UNIVERSITATEA „ALEXANDRU IOAN CUZA” DIN IAŞI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

ARVert

propusă de

Ceicoschi Valentin-Gabriel

Sesiunea: *iulie, 2019*

Coordonator ştiinţific

Prof. Colab. Florin Olariu

UNIVERSITATEA „ALEXANDRU IOAN CUZA” DIN IAŞI

FACULTATEA DE INFORMATICĂ

ARVert

Ceicoschi Valentin-Gabriel

Sesiunea: *iulie, 2019*

Coordonator ştiinţific

Prof. Colab. Florin Olariu

Cuprins

Introducere

Motivaţie

ARVert este o aplicaţie, dar şi un concept – un concept care odată implementat cu succes va schimba modul în care se face publicitate în contextul actual. Motivaţia din spatele conceptului vine mai ales din nevoia clienţilor de a avea o experienţă cât mai plăcută cu un produs de marketing, experienţă care momentan lipseşte din majoritatea reclamelor din mediul online şi din cel offline. De asemenea, un motiv în plus se regăseşte şi în nevoia companiilor de a creşte **rata de conversie** a utilizatorilor şi de a reduce costurile reclamelor.

Conform *pagefair.com*[1], **615 milioane** de dispozitive (desktop şi mobile) folosesc una sau mai multe soluţii de *ad-blocking*, acest număr reprezentând 30% din totalul dispozitivelor - în România numărul atinge **2,4 milioane.** Aceste date pot reprezinta dovada că modul în care se face publicitate în online este uşor deranjant pentru utilizatori.

Pe de altă parte, în offline reclamele sunt mai puţin deranjate, dar vin cu costuri mult mai mari. Spre exemplu, doar în USA costurile prezise pentru anul acesta ajung până la **200 miliarde USD**, conform *statista.com*[2].

Prin urmare, aceşti clienţii se vor bucura de o experienţă personalizată, în timp ce ofertele companiilor vor fi mai atractive şi cu costuri mai reduse. Acestora li se mai alătură un al treilea actor, reprezentat de deţinătorii de clădiri şi spaţii publicitare – suportul pentru reclamele interactive.

Lista motivelor pentru care această temă va aduce un plus în contextul actual se poate lărgi. Nu pot fi neglijate nici aspecte legate de provocarea pe care o propune şi de gradul de inovaţie al soluţiei, acestea reprezentând aspecte cheie în alegerea temei.

Obiectivele generale ale lucrării

Cum orice element de noutate vine cu un oarecare scepticism din partea publicului larg, este necesară o introducere succintă prin care să se evidenţieze necesitatea sa, dar şi cum îşi poate lăsa amprenta în contextul actual. De asemenea, se încearcă o materializare a conceptului şi trecerea sa dintr-un cadru abstract, într-unul palpabil, cu beneficii evidente.

Astfel, această lucrare are drept obiectiv principal **evidenţierea nevoii conceptului** (fiind o abordare prin care utilizatorul final se află în prim-plan) – o nevoie justificată prin statisticile menţionate mai sus.

[1] https://pagefair.com/intel/
[2] https://www.statista.com/

Un al doilea obiectiv urmărit este **descrierea problemei** şi **propunerea unei modalităţi de implementare** a produsului final. Se încearcă obţinerea unei posibile strategii prin care se pot aduce cei trei actori în acelaşi cadru, şi anume portalul ARVert. De asemenea, se doreşte îmbunătăţirea experienţei utilizatorului deoarece acesta reprezintă un motiv cheie pentru care tehnologia nu este adoptată la scară largă (relatează perkinscoie.com[3]).

În plus, un obiectiv la fel de important este şi **prezentarea tehnologiilor** necesare în dezvoltarea produsului, se va ţine cont de oportunităţile pe care le oferă fiecare în parte şi se va motiva folosirea lor în detrimentul celorlalte. De asemenea, se va motiva de ce s-au folosit abordări noi şi cum pot fi scalate şi implementate pentru a ţinti nevoile beneficiarilor.

Metodologia folosită

Având în vedere gradul de inovaţie al temei propuse, s-a încercat adoptarea unei metodologii centrată pe utilizator. **Design Thinking**[4] este o metodologie de design care vine cu o abordare bazată pe soluţie.

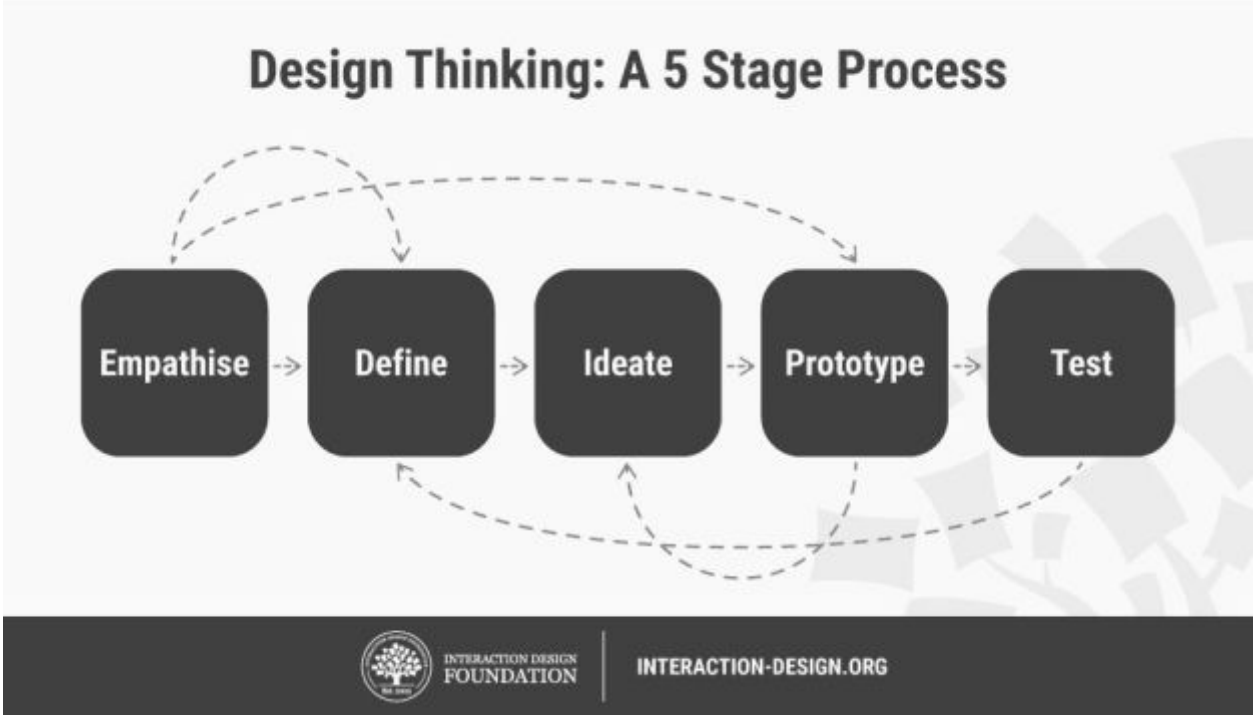

Figura 1 – Paşii metodologiei *Design Thinking* urmăriţi în acest proiect (Ilustraţia aparţine interaction-design.org cu drepturi depline).

[3] https://www.perkinscoie.com/en/
[4] https://www.interaction-design.org/literature/article/5-stages-in-the-design-thinking-process

Aceasta este extrem de utilă în situaţii în care problema este greu de definit sau este deosebit de complexă şi se referă la înţelegerea nevoilor umane prin remodelarea problemei, având utilizatorul în prim-plan, generarea de idei, testarea şi prototipizarea lor. Modelul general bazat pe cinci paşi este reprezentat în **Figura 1.**

Astfel, în pasul **Empathise** s-a încercat colectarea de informaţii despre experienţa actuală a utilizatorului. Acest lucru s-a realizat în trei moduri:

- Interviu prin **Google Forms**[5] în care principalele întrebări făceau referire la ultimele experienţe cu reclamele în mediul online şi offline – **Anexa 3**;
- Interviuri cu persoane care s-au declarat interesate de o astfel de idee;
- Studiul statisticilor publice care fac referire la motivele din spatele folosirii soluţiilor de ad-blocking şi a reach-ului pe care îl au reclamele – **Anexa 4**.

În urma acestui pas s-a constatat că premisele iniţiale se validează, publicul declarându-se nemulţumit, iar cei mai mulţi consideră că reclamele online lezează experienţa şi sunt deranjante. Dintre aceştia, 70% au declarat că utilizează o soluţie de ad-blocking, iar 66% au menţionat că prea puţine dintre reclamele din online sunt relevante nevoilor personale. Tot aici s-a constatat că utilizatorii sunt interesaţi de reclame care se pliază pe nevoile lor şi care nu sunt deranjante în viaţa de zi cu zi.

Acest pas a fost precedat de **Define (the problem)** în care s-a realizat o hartă a experienţelor şi s-a formulat **problema** având în centrul atenţiei utilizatorul, iar rezultatul a fost:
„Publicul, reprezentat de persoane din diverse medii şi categorii sociale, vrea să-şi satisfacă nevoia pentru noutăţi personalizate din segmentul marketing într-o manieră care nu deranjează, dar este asaltat de reclame atât în online, cât şi în offline."
Întrebarea care a precedat această problemă a fost:
„Cum am putea noi să satisfacem nevoia publicului pentru noutăţi din segmentul marketing într-o manieră personalizată şi care să nu afecteze calitatea experienţei?"
Următorul pas a fost **Ideate**, în care s-au adus diverse idei şi s-au selectat şi combinat cele cu cel mai mare potenţial. Astfel, s-a obţinut o primă soluţie care propune ca întreaga experienţă cu ad-urile să fie un hibrid între online şi offline, o soluţie care combină reclame digitale aplicate pe un suport fizic, proiectarea acestora folosind, practic, realitatea augmentată pe un suport (clădire, panou) din viaţa reală.

[5] https://www.google.com/forms/about/

În pasul **Prototype (Figura 2)** s-au realizat câteva modele mobile cu un flow foarte simplu, iar acesta a fost trimis către **Test**, unde s-a observat impactul soluţiei asupra utilizatorilor.

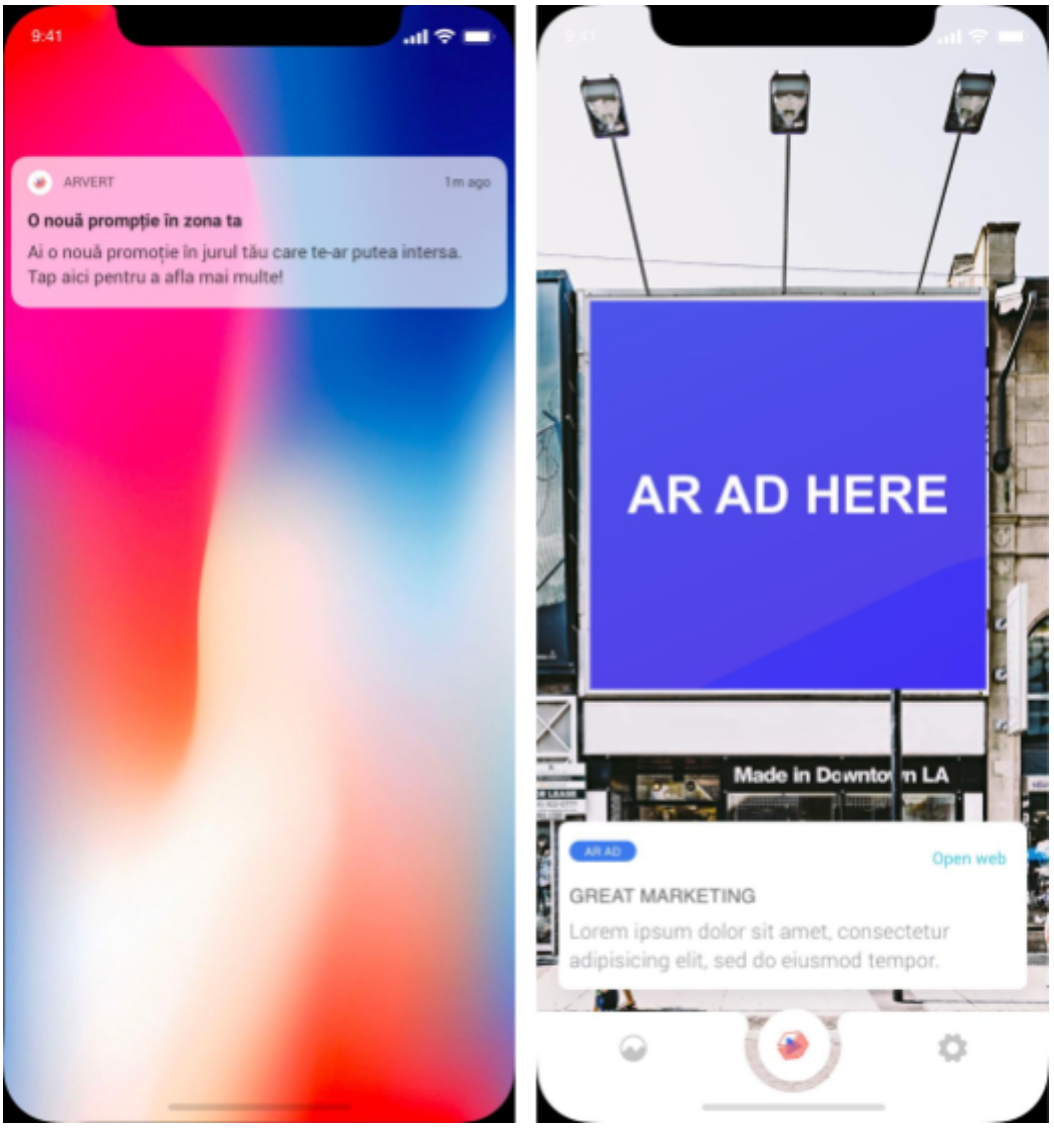

Figura 2 – Prototipul aplicaţiei (O parte din elementele grafice nu aparţin autorului, ci brand-ului **Apple**[6]).

Descrierea sumară a soluţiei

ARVert este o aplicaţie de mobile care odată instalată va aduce **utilizatorului final** notificări despre noutăţile şi reclamele din jurul său. Aceste notificări sunt personalizate în funcţie de profilul utilizatorului şi de locul în care se află. În acelaşi timp, soluţia tratează şi nevoia **companiilor** de a înregistra reclame. Acestea au posibilitatea selecţiei dintr-o gamă de spaţii publicitare şi de a încărca un model tridimensional care să respecte o serie de specificaţii.

[6] https://www.apple.com/

Totodată, **deţinătorii de spaţii publicitare**, pot solicita ca un anumit spaţiu sa fie utilizat drept suport pentru o reclamă AR.

Structura lucrării

Această lucrare este împărţită în trei capitole principale (care includ mai multe sub-capitole), precedate de concluzii, bibliografie şi anexe.

În prima parte, adică în capitolul **Tehnologiile folosite,** s-a propus un stack de tehnologii care a fost folosit în realizarea aplicaţiei mobile, a API-ului şi a integrării cu soluţiile oferite de **Google Cloud**[7] şi **Firebase**[8] şi s-au tratat aspecte legate de tehnologia AR (**ARKit**[9]).

Următorul capitol, **Arhitectura proiectului**, este împărţit în mai multe titluri:

- Context şi Componente – clarifică structura generală a soluţiei şi cum se integrează aceasta în contextul actual (diagrame C4 nivelul 1 şi nivelul 2);
- Baze de date – se surprind detaliile din structura bazei de date (diagrame de baze de date);
- Structura API – cum au fost integrate serviciile Google Cloud şi Firebase astfel încât API-ul să-şi îndeplinească funcţiile descrise;
- Structura front-end – noţiuni care cuprind arhitectura unei aplicaţii mobile, interacţiunea cu API-ul şi cu tehnologia AR
- Securitate – aspecte legate de manipularea datelor cu caracter sensibil (date personale, parole, date bancare, statistici etc.)

În capitolul **Descrierea soluţiei** s-au discutat detalii din implementarea efectivă a soluţiei, tratând fiecare componentă menţionată în **Arhitectură,** dar şi componente particulare.

Finalul lucrării este structurat în **Concluzii** – în care se aduce în prim-plan părerea autorului despre această lucrare, viitoare direcţii şi dificultăţile întâlnite, **Bibliografie** – se menţionează toate sursele de inspiraţie pentru această lucrare, autori şi articole care au făcut posibil acest articol ştiinţific, iar în **Anexe** – se însumează o serie studii proprii, studii publice şi statisticile care au pus bazele şi premisele lucrării.

[7] https://cloud.google.com/
[8] https://firebase.google.com/
[9] https://developer.apple.com/augmented-reality/

Capitolul 1: Descrierea problemei

Tehnologiile folosite

ARVert reprezintă un concept relativ nou care are la bază o tehnologie încă în dezvoltare – acesta este motivul pentru care tehnologiile folosite în dezvoltarea sistemului sunt relativ noi şi se regăsesc în dezvoltarea multor aplicaţii recente. Se poate observa că soluţia are la bază servicii de **Cloud Computing** care cresc stabilitatea şi scalabilitatea acesteia, dar şi portabilitatea şi modularizarea sistemului. Pe lângă serviciile cloud, aplicaţia este scrisă integral în **Javascript** deoarece reduce costurile de mentenanţă.

Firebase este o platformă destinată dezvoltării aplicaţiilor pentru mobile şi web. Aceasta oferă programatorilor o suită de servicii şi unelte care cresc calitatea aplicaţiilor, ajută la scalarea acestora şi reduc necesitatea configurării unor soluţii de hosting (conform hackernoon.com[10]). S-a optat pentru această abordare în detrimentul variantei clasice, ce are la bază un server, datorită stabilităţii, timpului redus pentru configurare, viteza de răspuns (deoarece fişierele 3D ating dimensiuni mari) şi posibilităţii de a aduce aplicaţiei viitoare îmbunătăţiri cu un efort redus; mai multe detalii se găsesc în **Anexa 1.**

NodeJS este folosit în funcţiile firebase deoarece permite instalarea de pachete, este uşor de configurat şi folosit şi aduce un plus la mentenanţa aplicaţiei; mai multe detalii se găsesc în **Anexa 2**.

React Native este un framework pentru realizarea de aplicaţii mobile folosind JavaScript şi care are la bază **ReactJS**. Aplicaţia de mobile ARVert este construită folosind React Native deoarece s-a mizat pe portabilitate, astfel codul sursă poate fi refolosit pentru aplicaţii Android, cât şi IOS – *write once, run anywhare*[11].

Expo este un mediu de dezvoltare a aplicaţiilor mobile. Expo poate simula rularea aplicaţiilor native, poate realiza build-uri finale ale aplicaţiei pentru ambele platforme şi poate facilita testarea aplicaţiei. De asemenea, Expo vine la pachet cu un SDK care oferă posibilitatea de a interacţiona cu componentele native de sistem.

[10] https://hackernoon.com/introduction-to-firebase-218a23186cd7
[11] https://medium.com/@vaibhavsingh_56357/react-native-web-write-once-run-anywhere-12a3aab326fc

ARKit reprezintă motorul de bază al sistemului ARVert şi cel care poate face această soluţie funcţională. ARVert se bazează pe tehnologia oferită de ARKit pentru a crea o experienţă unică pentru utilizatori. Mai multe informaţii se găsesc în **Anexa 2**.

Arhitectura proiectului

În cele ce urmează s-a încercat explicarea arhitecturii generale a sistemului, având în prim-plan elemente precum **Context şi componente**, reprezentate prin diagramele C4, structura bazei de date, structura generală a API-ului şi ce funcţii îndeplineşte acesta în cadrul aplicaţiei, structura front-end (aplicaţia de mobile), aspecte legate de securitate şi cum s-a realizat integrarea cu ARKit-ul.

Pe baza premiselor iniţiale şi a rezultatelor metodologiei Design Thinking, s-au identificat trei tipuri de clienţi pentru care este destinat ARVert (**Figura 3**):

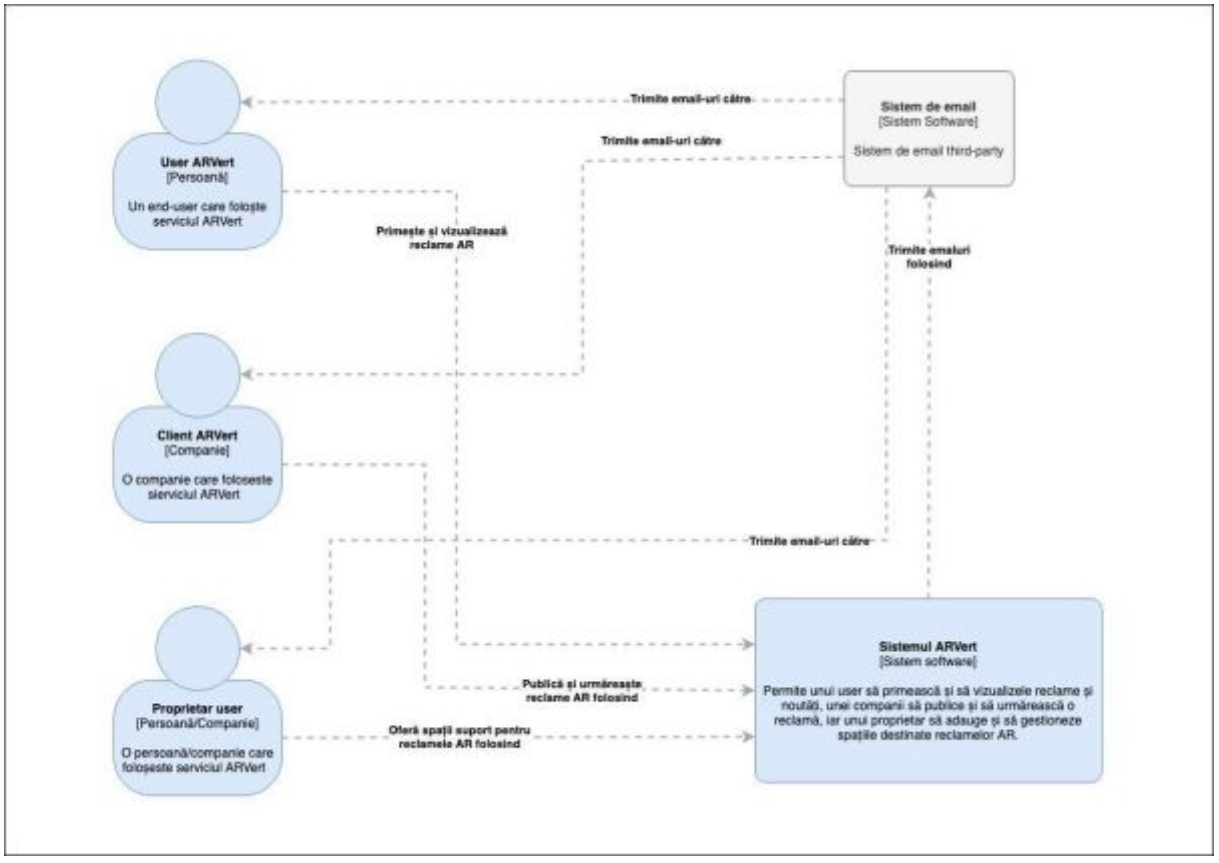

Figura 3 – Diagrama C4 (nivel 1) care explică cum sistemul ARVert interacţionează cu utilizatorii săi.

- **Utilizatorul Final** – o persoană care este interesată de noutăţile şi reclamele din jurul său şi foloseşte aplicaţia ARVert pentru a fi la *curent* cu acestea. Prin urmare, ARVert trimite notificări utilizatorului, acesta având posibilitatea de a vizualiza şi interacţiona cu reclama AR.

- **Client ARVert** – o companie care foloseşte serviciile ARVert pentru a publica reclame şi anunţuri cu interes public în platforma ARVert.
- **Partener ARVert** – un utilizator care dispune de spaţii ce pot fi folosite drept suport pentru reclamele AR. Acesta are acces la înregistrarea de spaţii noi şi gestionarea celor înregistrate deja.

Tot aici se explică succint modul în care sistemul ARVert va satisface nevoile utilizatorilor şi care sunt funcţiile sale. Astfel, ARVert va putea înregistra şi autentifica utilizatori, va putea trimite notificări despre promoţiile şi reclamele din zona utilizatorului, va permite vizualizarea lor, va facilita crearea de reclame noi şi de spaţii suport.

Pentru a intra în profunzimea sistemului, s-a realizat o diagramă C4 (nivel 2) – **Figura 4**, care reduce gradul de abstractizare al soluţiei. Aici se pot identifica patru componente care au fost implementate şi o componentă care a fost integrată după cum urmează:

- **Aplicaţia de Mobile** – care are la bază React Native şi Expo şi care reprezintă portalul prin care utilizatorul are acces la sistemul ARVert. Aici se îndeplinesc funcţii diverse: de la cele mai simple, cum ar fi autentificarea şi înregistrarea utilizatorilor, sistemul de notificări în timp real, gestionarea reclamelor şi a spaţiilor suport, până la funcţia de afişare şi interacţionare cu reclamele AR.
- **API** – reprezintă o componentă stand-alone care este modelată sub metodologia RESTful API[12], o componentă care rulează în Cloud prin intermediul Firebase Functions şi este scrisă în NodeJS. Acest API reprezintă o sumă de rute care vor fi declanşate de apeluri HTTPS din cadrul aplicaţiei de mobile. Comunicarea cu acest modul se va face prin JSON-uri.
- **Database** – baza de date este structurată sub forma unui JSON, NoSQL şi este construită folosind Firestore. Aici sunt grupate date despre utilizatori (preferinţe, istoric, statistici), date despre reclamele şi anunţurile AR, legăturile către modele 3D, dar şi date despre locaţii. În baza de date se efectuează operaţiuni CRUD[13] din partea API-ului folosind framework-ul Firebase.
- **Storage** – Firebase Storage, acesta permite stocarea fişierelor media şi a modelelor 3D pentru reclame.

[12] https://profs.info.uaic.ro/~gcalancea/cloud-computing/Week3.pdf
[13] Create, Read, Update, Delete – https://www.codecademy.com/articles/what-is-crud

- **Autentificare** – Firebase Authentication este apelat în mod direct de către aplicaţia de mobile prin intermediul metodelor expuse. Acest modul se ocupă de autentificarea utilizatorilor şi returnarea unei sesiuni persistente.

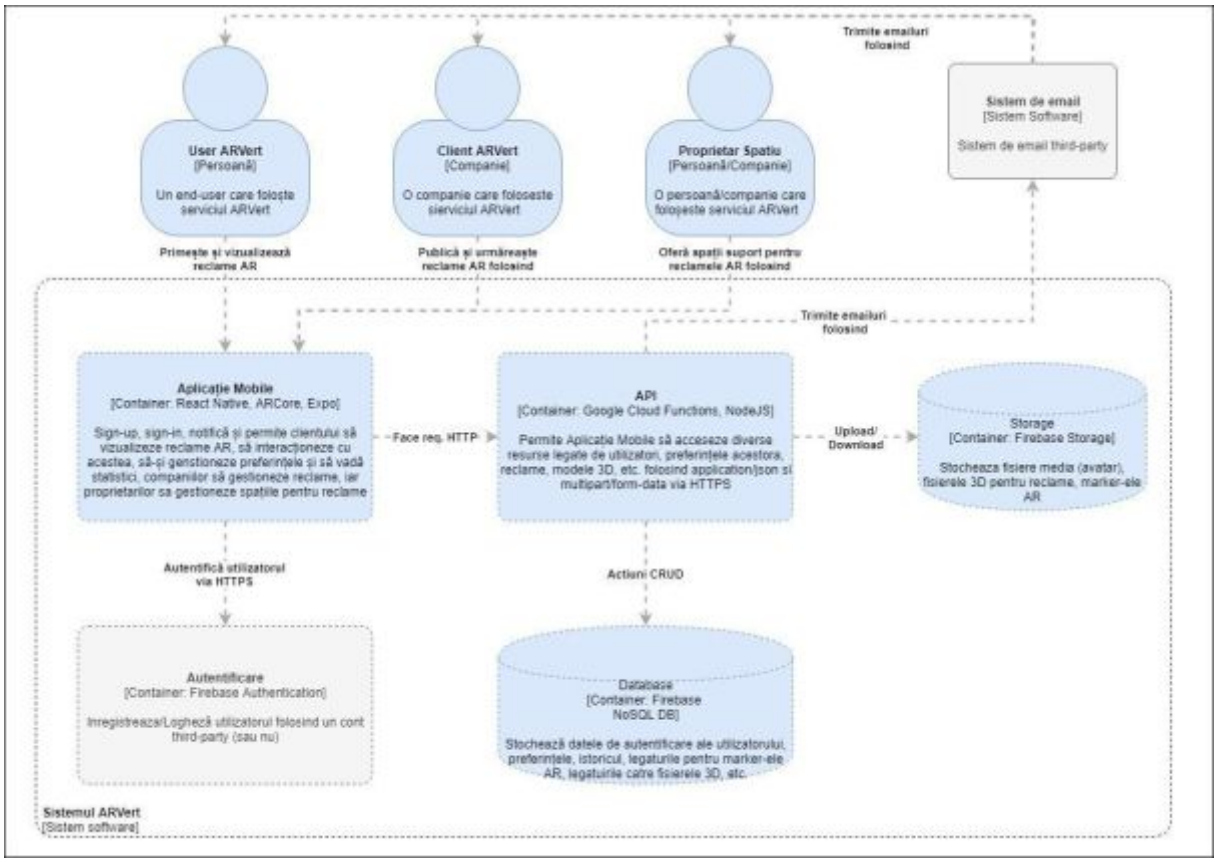

Figura 4 – Diagrama C4 (nivel 2) explică cum sistemul ARVert interacţionează cu utilizatorii săi şi cum interacţionează componentele din sistemul ARVert unele cu celelalte.

Pentru **baza de date**, s-a propus o structură simplă **(Figura 5)**, care are în centrul atenţiei utilizatorul:

- **Users** – tabela în care se vor reţine date despre utilizator cum ar fi informaţiile personale, parola, data de naştere etc.;
- **Roles** – orice utilizator va avea un rol la nivel de sistem, fie că este client sau administrator;
- **Ads** – tabela în care se găsesc datele importante despre reclame. Aici putem găsi căile către model şi alte resurse, dar şi identificatorul categoriei din care face parte, pentru ce gen şi interval de vârstă este destinată reclama etc.;
- **Categories** – reprezintă tabela în care vor fi listate categoriile pentru fiecare reclamă;
- **Visits** – această tabelă va trata fiecare vizualizare a unei reclame; aici se vor înregistra data şi ora când a fost vizitată, dar şi ce utilizator a executat acţiunea.

- **Locations** – această tabelă va conţine toate locaţiile care au fost listate drept suport pentru reclamele AR. Aici se găsesc date despre locaţia în care se află, cine o deţine şi indicii estimativi ai valorii.

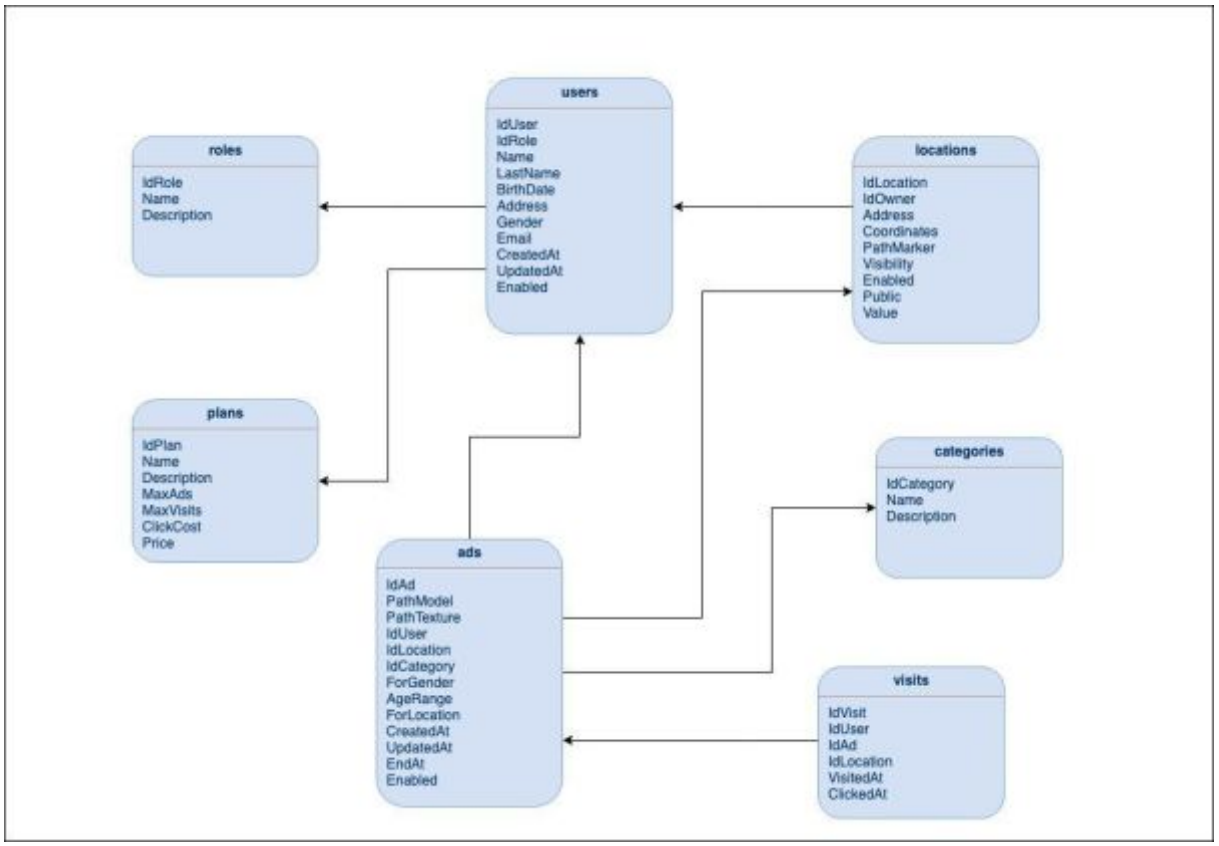

Figura 5 – Diagrama Bazei de Date (Firebase Database – NoSQL).

Structura API-ului se pliază pe structura bazei de date. Astfel printre rutele **(Figura 6)** principale se numără:

- **GET /users/{id}** – se pot accesa date despre un utilizator;
- **POST /users** – se creează un utilizator nou la înregistrarea cestuia în platformă;
- **GET /ads** – se pot accesa date despre reclame în funcţie de filtre: locaţie, interval de vârstă, sex etc.;
- **POST /ads** – adăugarea unei reclame în baza de date (aici e necesară încărcarea de fişiere);
- **POST /locations** – adăugarea unei locaţii suport pentru reclamele AR (adresă, marker, etc.).

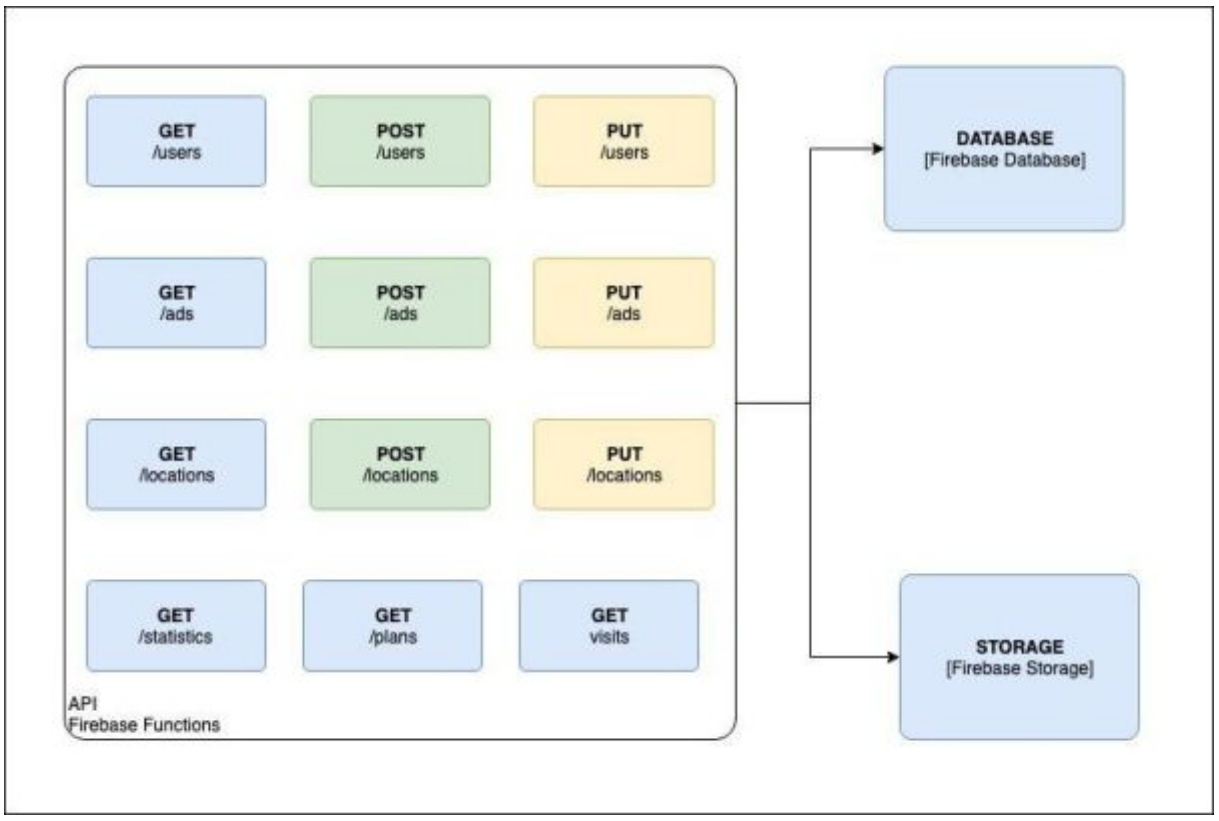

Figura 6 – Structura de end-point-uri API.

Aplicaţia va fi structurată în pagini diverse care vor interacţiona cu utilizatorul după nevoile sale. Ea va fi instalată pe dispozitivele utilizatorilor şi se va conecta în mod dinamic la API şi la sistemul de autentificare.

După instalarea aplicaţiei, utilizatorul va avea posibilitatea de a se conecta folosind un cont existent sau aceea de a-şi crea un cont nou. După conectarea cu succes a utilizatorului, acesta poate să parcurgă un tutorial sau poate să ajungă direct la meniul aplicaţiei. Din meniu va avea posibilitatea să selecteze pasul următor: poate deschide camera pentru a vizualiza o reclamă AR, să actualizeze detaliile contului (nume, prenume, poza, parola) sau să acceseze Centrul AR. Aici va putea adăuga reclame noi accesând formularul de creare reclamă sau să adauge o locaţie în sistem. Tot aici va putea vizualiza reclamele şi locaţiile sale. În final, utilizatorul se poate deconecta din contul său – vezi Flowchart-ul din **Figura 7**.

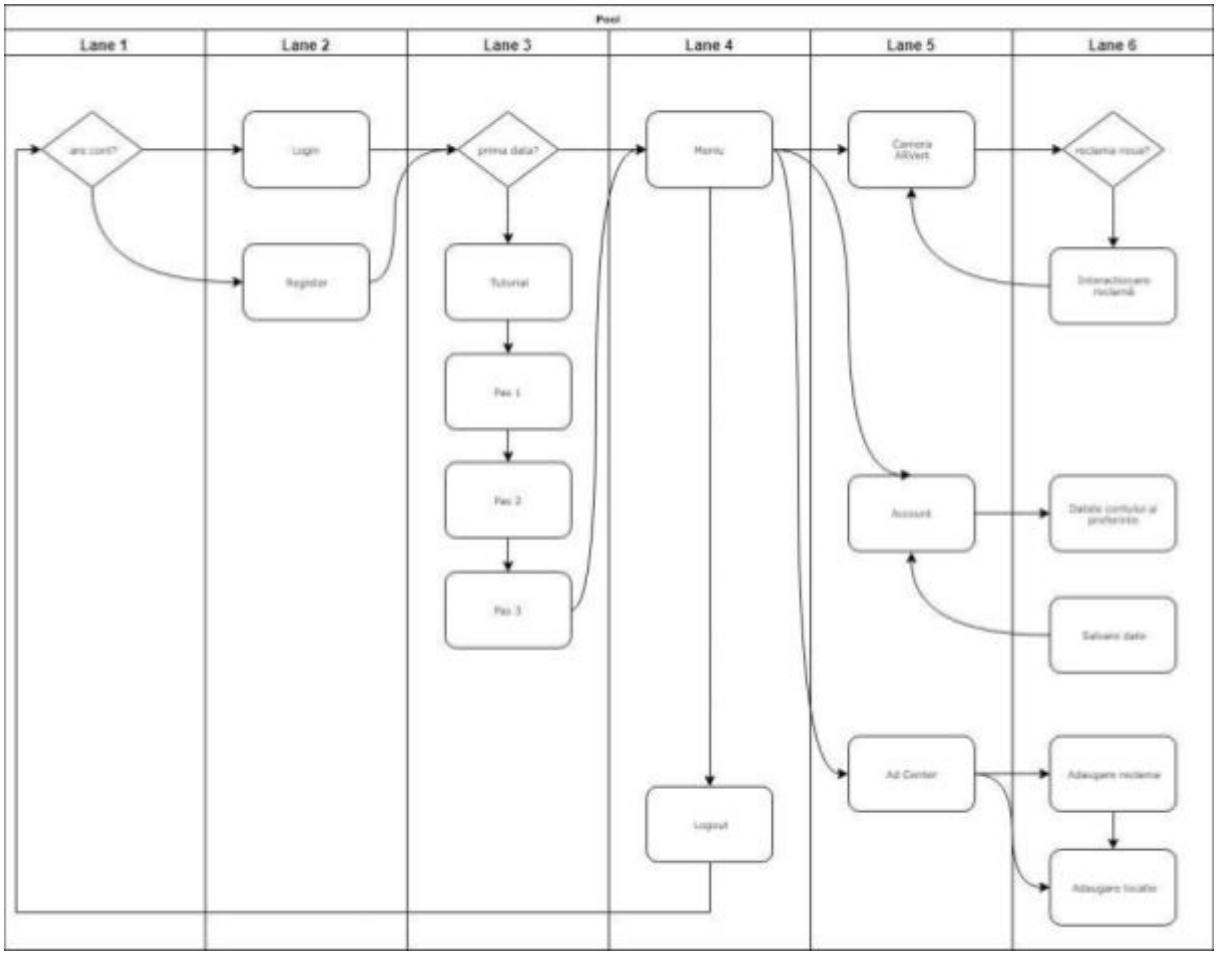

Figura 7 – Flowchart-ul aplicaţiei şi principalele funcţionalităţi.

Siguranţa utilizatorului este gestionată de sistemul de autentificare. Astfel, Firebase Authentication gestionează conectarea utilizatorului şi salvează o sesiune validă cu datele sale. Acest proces are la bază o cheie unică, specifică aplicaţiei şi care va fi folosită pentru autentificare şi înregistrare. Tot aici s-au tratat clauzele **GDPR**[14], datele confidenţiale ale utilizatorului fiind codate folosind **bcrypt**[15].

Pe de altă parte, comunicarea dintre server şi aplicaţie se va realiza via HTTPS şi pe baza unui token de acces unic pentru fiecare utilizator generat de către Firebase Authentication, după autentificarea acestuia.

S-a optat pentru un API pentru ca aplicaţia să nu acceseze în mod direct baza de date; astfel apelurile neautorizate sunt uşor de filtrat.

Concluzii

Acest capitol aduce în prim-plan tehnologiile folosite în realizarea sistemului ARVert, se motivează beneficiile, provocările şi funcţiile pe care acestea le îndeplinesc. În partea de început se vorbeşte de serviciile Google Cloud ca o alternativă viabilă la clasicul web-server, urmat de limbajul sub care este scrisă soluţia (Javascript), despre NodeJS la nivel de API şi React Native la nivel de aplicaţie mobile. De asemenea, se pune accent pe ARKit, drept nucleul aplicaţiei în jurul căreia se poate aduce o soluţie viabilă pentru această problemă.

Tot aici se tratează modul în care s-a realizat arhitectura sistemului ARVert. Se menţionează cei trei actori pentru care este destinat sistemul, se surprinde structura bazei de date (tabele şi dependenţele dintre acestea), se aduc informaţii despre rutele API şi tratează aspecte legate de securitate şi GDPR.

[14] General Data Protection Regulation – https://eugdpr.org/
[15] https://www.npmjs.com/package/bcrypt

Capitolul 2: Abordări anterioare

Cu toate că primele sisteme funcţionale care aveau la baza realitatea augmentată au fost folosite încă de la începutul anilor 1990, tehnologia a devenit cunoscută în rândurile consumatorilor abia în 2014, odată cu lansarea **Google Glasses** şi cu un apogeu de popularitate după lansarea jocului **Pokémon GO** în 2016, relatează colocationamerica.com[16]. Astfel, până în prezent, tehnologia a putut fi exploatată la nivel larg, remarcându-se în zona de marketing prin produse precum IKEA AR[17], UBER AR Experience[18], Pepsi – bus shelter[19], A/R Jordan[20] etc.

Pe plan autohton, tehnologia a fost abordată cu succes de către CreativeVR[21] – o echipă care realizează experienţe AR/VR pentru diverse companii şi cauze şi care aduce AR-ul în segmentul publicitar.

Dintre acestea, **A/R Jordan** este una dintre abordările care se asemănă cu soluţia propusă în această lucrare. Practic, este prima experienţă comercială realizată prin intermediul realităţii augmentate[22]. Astfel, brandul Jordan, împreună cu Snapchat şi Shopify, a realizat o experienţă inedită prin care utilizatorul putea să cumpere noua pereche de încălţăminte sport marca Jordan, direct din aplicaţia Snapchat. Aici era expus un model 3D al sportivului Michael Jordan, iar la accesarea sa se putea comanda produsul dorit direct din aplicaţie. Campania a avut un succes neaşteptate, cu lichidări de stocuri în primele ore (sursa rga.com).

Produsele menţionate mai sus au fost folosite într-un context redus şi care vizau un număr restrâns de părţi – în schimb, ARVert propune o abordare generalizată a tehnologiei în zona de marketing şi aduce toţi actorii în acelaşi context, fiind uşor de personalizat pentru fiecare cauză, produs sau brand, dar şi o soluţie cu integrare rapidă în orice campanie publicitară şi care se pliază pe nevoile utilizatorilor.

Pe de alta parte, ARVert aduce inovaţie pe scena advertising şi propune o nouă manieră prin care utilizatorii ajung la informaţie – manieră ieftină, care nu lezează experienţa utilizatorului şi care se bucură de entuziasmul tehnologiei.

[16] https://www.colocationamerica.com/blog/history-of-augmented-reality
[17] https://www.ikea.com/gb/en/customer-service/ikea-apps/
[18] https://www.youtube.com/watch?v=bCcvEVyAXQ0
[19] https://www.youtube.com/watch?v=Go9rf9GmYpM
[20] https://vimeo.com/273224288
[21] http://creativevr.ro/
[22] https://www.rga.com/work/case-studies/a-r-jordan

Cu aceste lucruri în minte, se poate deduce că conceptul este predispus succesului odată cu evoluţia tehnologiei şi a sectorului AR/VR, dar şi răspândirea largă a ochelarilor AR (tractica.com[23] anunţă că numărul ochelarilor AR va atinge 22.8 milioane până în 2022).

[23] https://www.tractica.com/

Capitolul 3: Descrierea soluţiei

Serverless API

La nivel de API se iniţializează aplicaţia, folosind metoda expusă de către librăria firebase-admin, *initializeAPP()*. Pentru a putea accesa datele înregistrate în baza de date este necesar un fişier de autentificare generat din consola Google. După iniţializarea aplicaţiei prin metoda *firestore()*, se oferă acces efectiv la acele date care vor putea fi prelucrate după nevoile programatorului. Un exemplu poate fi observat în **Figura 8**.

```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
var serviceAccount = require("./arvert-data");
admin.initializeApp({
 credential: admin.credential.cert(serviceAccount),
 databaseURL: "https://arvert.firebaseio.com"
});
let db = admin.firestore();
```

Figura 8 – Exemplu de iniţializare a bazei de date la nivel de API.

De asemenea, aici se iniţializează serviciul folosind *express()*, se mapează ruta principală */api* şi se conectează celelalte rute prin *app.use()*. Constanta *webApi* este de fapt funcţia cloud la care se fac apelurile (**Figura 9**).

```
const app = express();
const main = express();
main.use('/api/', app);
main.use(bodyParser.json());
export const webApi = functions.https.onRequest(main);
app.use('/', router);
```

Figura 9 – Exemplu de iniţializare serviciului API.

Tot aici sunt scrise rutele platformei, iar în **Figura 10** se găseşte un exemplu de funcţie care răspunde la un apel GET la /ads. Se aduce din baza de date colecţia necesară prin metoda *collection()*, apoi se trimit datele obţinute într-un răspuns de tip JSON.

```
exports.getAds = functions.https.onRequest(async (request, response) => {
 response.header('Content-Type','application/json');
 response.header('Access-Control-Allow-Headers', 'Content-Type');
 let ads = [];
 try {
   const snapshot = await db.collection(ads).get()
   snapshot.forEach((doc) => {ads.push(doc.data())})
 } catch (err) {response.json(err)}
 if (ads.length) {response.json(ads);
 } else {response.status(404).json('Error');});
```

Figura 10 – Exemplu al metodei **getAds**, care returnează toate reclamele din baza de date.

Autentificare

Autentificarea are la bază Firebase Authentication, acest serviciu dispunând de metode de autentificare cu email şi parolă, dar şi printr-o reţea de socializare. În **Figura 13** s-a tratat cazul în care se doreşte autentificarea cu Google – prin *onAuthStateChanged()* se verifică dacă s-a schimbat starea curentă a utilizatorului (login, logout), apoi prin *isUserEqual()* se verifică dacă utilizatorul este deja logat în sistem. *GoogleAuthProvider.credential()* generează token-ul de acces al utilizatorului, token ce va fi folosit pentru a valida apelurile la API. Tot aici se salvează sau actualizează datele primite de la Google în Firestore. În **Figura 14** este surprinsă pagina de login cu email şi parolă, dar şi modelul de autentificare folosind un cont google.

```
function onSignIn(googleUser, props) {
 let unsubscribe = firebase.auth().onAuthStateChanged((firebaseUser) => {
   unsubscribe();
   if (!isUserEqual(googleUser, firebaseUser)) {
     let credential =
firebase.auth.GoogleAuthProvider.credential(googleUser.idToken,googleUser.a
ccessToken);
firebase.auth().signInWithCredential(credential).then(async (result)=>{
    const user = {...};
    if (result.isNewUser) {props.saveUser(user, result.user.uid)}
     else {props.updateUser(user, result.user.uid)}
});}});}
```

Figura 13 – Exemplu de autentificare a utilizatorului folosind Firebase şi GoogleAuthProvider.

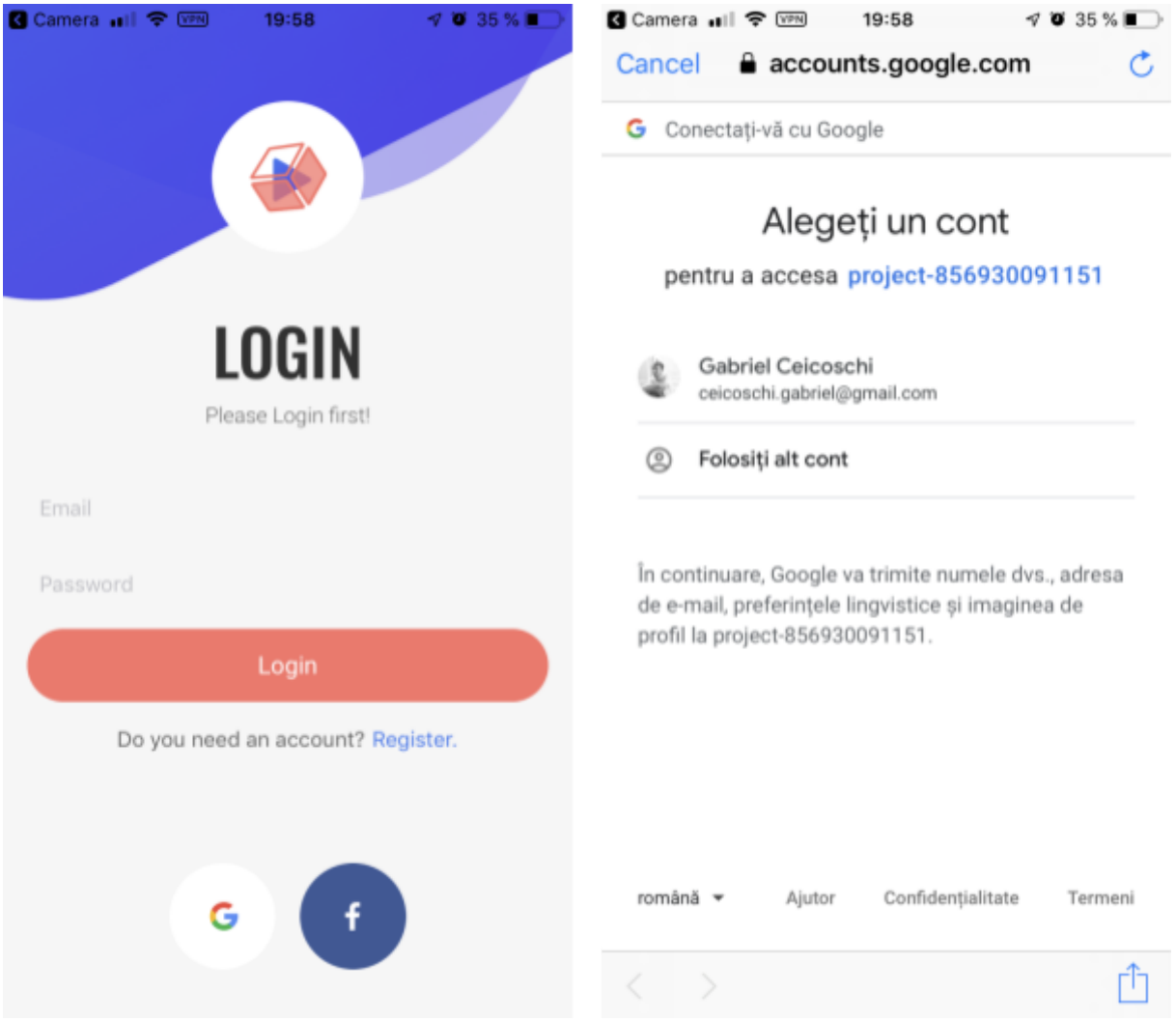

Figura 14 – Paginile pentru autentificare prin email şi autentificare cu Google. Elementele grafice cum ar fi iconiţele, fundalul, fonturile, logourile Facebook şi Google nu aparţin autorului.

Navigare

Aplicaţia este împărţită în patru pagini principale. Printre acestea se numără **LoginScreen** şi **RegisterScreen** – componentele care tratează autentificarea utilizatorului, **LoadingScreen** – componentă care tratează tranziţia dintre utilizatorul neautentificat şi cel autentificat, dar şi pagina **Main**. În **Main** se găsesc trei componente care vizează aspectele cheie ale platformei:

- Camera AR – interacţionarea cu reclamele AR şi camera foto;
- Centrul AR – vizualizarea, actualizarea şi crearea de locaţii şi reclame noi;
- Setări – aspecte legate de utilizatorul curent, profil şi setările aplicaţiei.

Camera AR

La nivelul aplicației s-a realizat camera AR (**Figura 15**). Aceasta are rolul de a afișa utilizatorului reclamele interactive și a fost construită peste librăria expo-three[24]. Expo-three expune clasa *Renderer* care reprezintă canvas-ul în care aplicația desenează modele 3D. Tot aici se încarcă modelul și se expun metode precum *Scene()* – inițializează scena curentă, *Camera()* – inițializează camera dispozitivului, *Light()* – controlează aspecte legate de lumina ambientului, contraste, umbre etc.

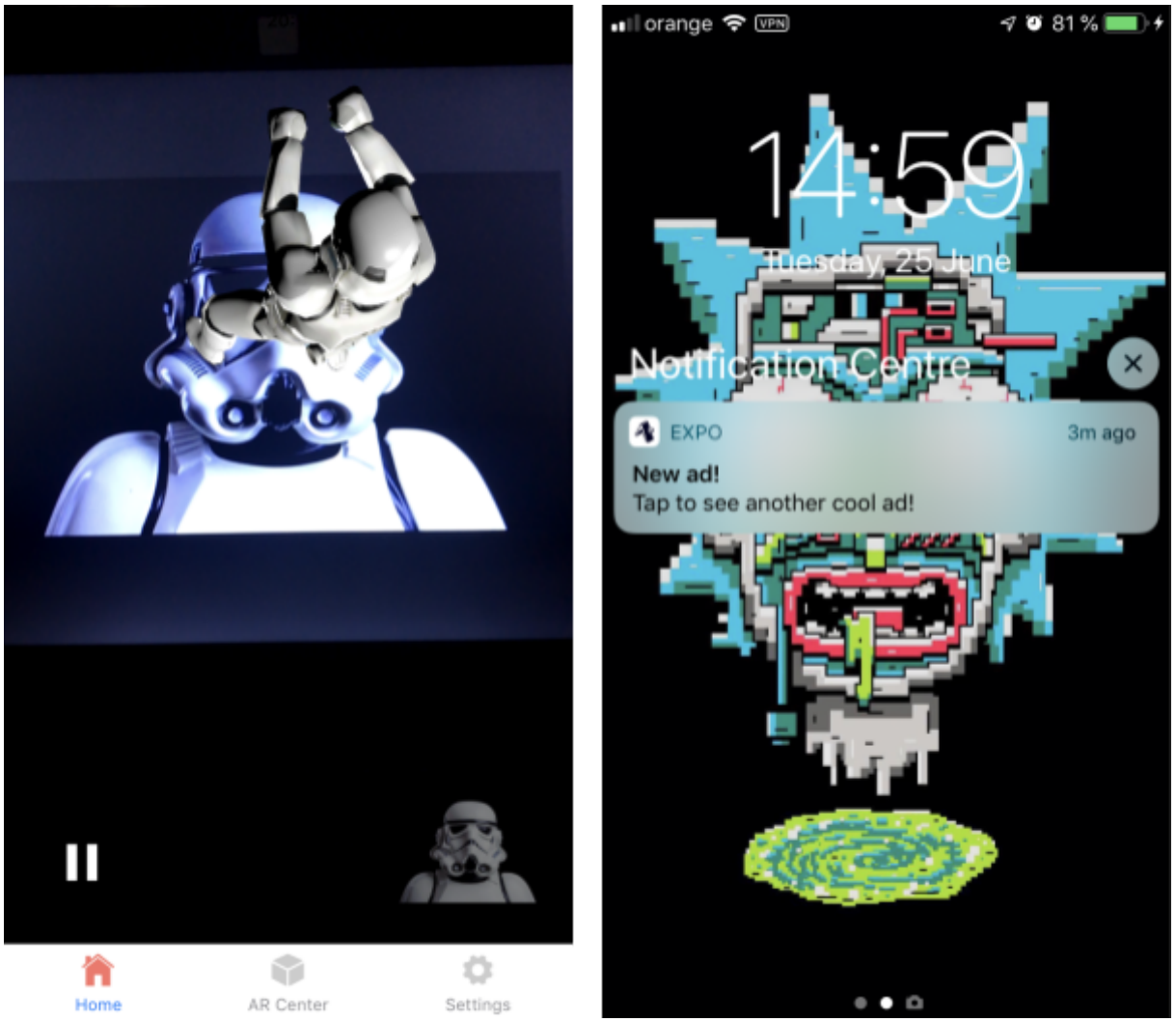

Figura 15 – În stânga, Camera AR și desenarea unui model 3D; în dreapta, a, un exemplu de notificare pentru reclama curent (atât textura, modelul cât și imaginea marcaj fac parte din exemplele librăriei expo-three https://github.com/expo/expo-three/tree/master/example/assets, iar imaginea de background din dreapta nu aparține autorului).

[24] https://github.com/expo/expo-three

În **Figura 16** este reprezentat un exemplu de inițializare a camerei și a modelului 3D.

```
onContextCreate = async ({ gl, scale: pixelRatio, width, height }) => {
   AR.setPlaneDetection(AR.PlaneDetectionTypes.Horizontal);
   await this.addDetectionImageAsync(Assets[marker]);
    this.renderer = new ExpoTHREE.Renderer({ gl, pixelRatio, width, height
});
   this.scene = new THREE.Scene();
   this.scene.background = new ThreeAR.BackgroundTexture(this.renderer);
   this.camera = new ThreeAR.Camera(width, height, 0.01, 1000);
   this.ambient = new ThreeAR.Light();
   await this.loadModel();
 };
```

Figura 16 – Exemplu de inițializare a modulului AR responsabil de afișarea modelelor 3D.

Acest modul centralizează imaginile primite de la cameră, modelul 3D și marcajul primite de la Storage. Astfel, aici se face matching-ul dintre cele trei și se oferă utilizatorului experiența așteptată. Funcția *loadModel()* tratează încărcarea dinamică a modelului și creează o scenă nouă pentru sesiunea curentă.

Centrul AR

A doua pagină importantă din cadrul aplicației este **Centrul AR (Figura 18).** Aici sunt listate reclamele și locațiile înregistrate de utilizator cu posibilitatea de a crea locații și reclame noi. Principalele metode ale componentei se găsesc în **Figura 17**. Funcțiile *setNewLocation()* și *setActiveLocation()* fac parte din modulul **Gestionare date** care va fi explicat ulterior, iar funcția *navigate()* este o metodă a modulului Navigare descris anterior.

```
function onNewLocationPress (props) {
 props.setNewLocation();
 props.navigation.navigate("LocationScreen");
}
function onLocationPress (props, location) {
 props.setActiveLocation(location);
 props.navigation.navigate("LocationScreen");
}
```

Figura 17 – Metodele principale din Centrul AR.

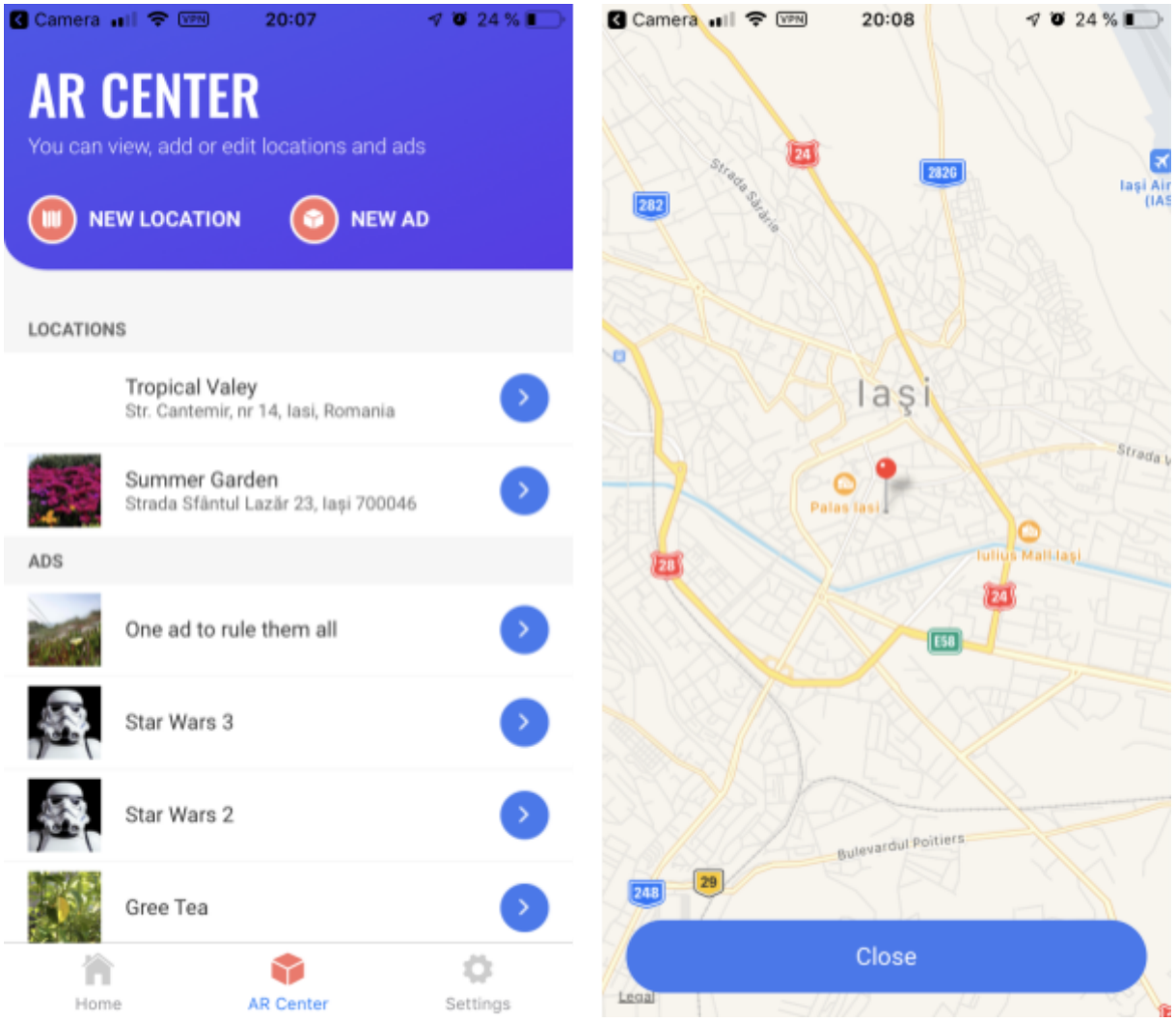

Figura 18 – În stânga, Listarea reclamelor şi locaţiilor; în dreapta, vizualizarea pe hartă a locaţiei. Elementele grafice cum ar fi iconiţele, fundalul, fonturile nu aparţin autorului.

Vizualizarea şi adăugarea unei locaţii sunt tratate în componenta **LocationScreen (Figura 19 stânga)**. Aici se poate încărca marcajul AR care va fi folosit de Camera AR pentru a identifica o clădire sau spaţiu publicitar, se pot seta numele şi adresa exactă şi se poate vizualiza locaţia pe hartă.

Asemănător a fost tratată vizualizarea şi adăugarea unei reclame în componenta **AdScreen (Figura 19 dreapta)**. În această pagină se poate încărca o imagine reprezentativă a reclamei, dar se pot seta şi detalii precum numele, categoria, publicul, ţinta (sex, segment de vârstă) şi se poate selecta o locaţie disponibilă pentru plasarea reclamei. Tot aici se pot seta căile către modelul 3D, dar şi către texturile folosite în desenarea acestuia.

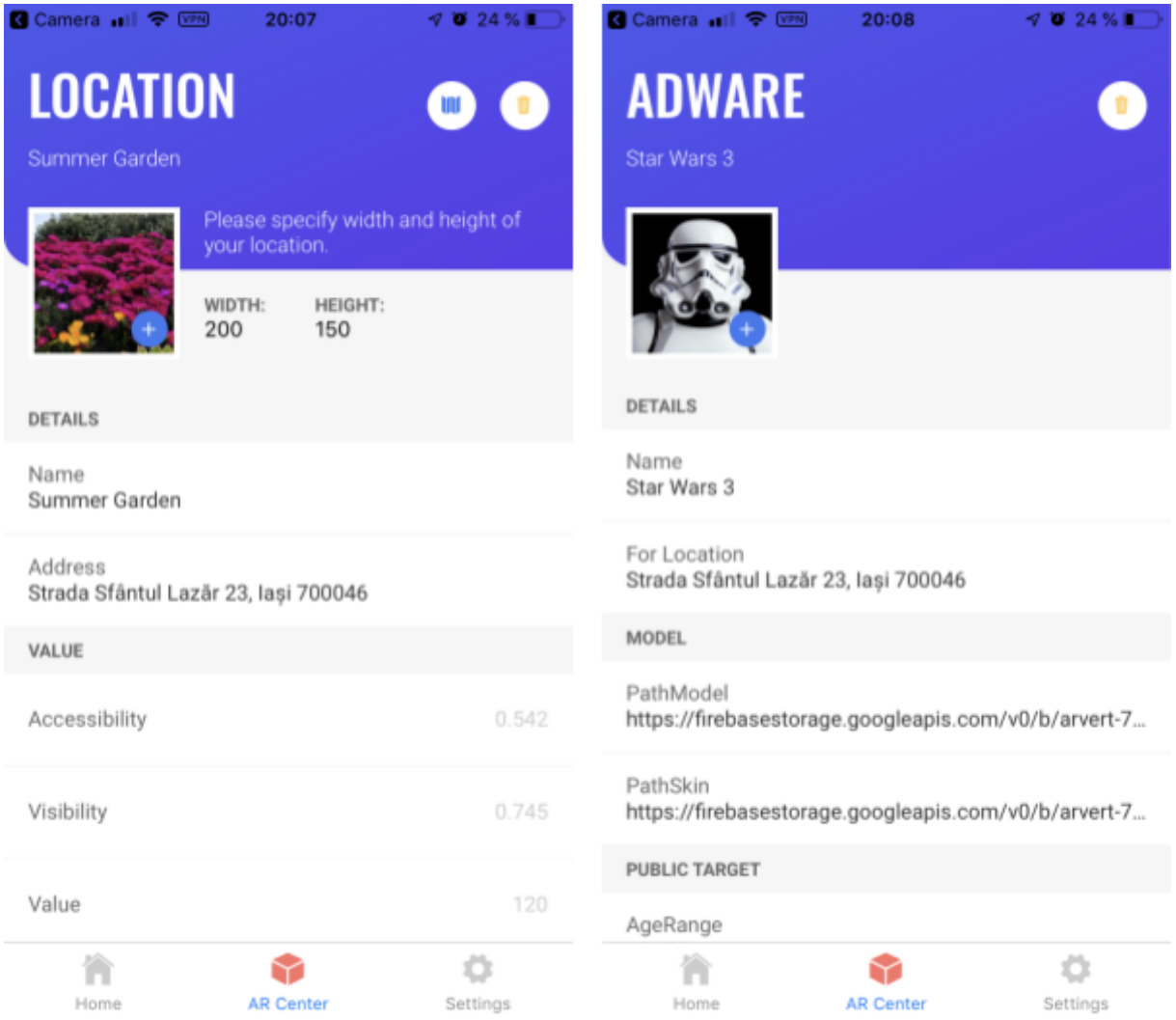

Figura 19 – Paginile responsabile cu vizualizarea şi editarea unei locaţii, respectiv ale unei reclame. Elementele grafice cum ar fi iconiţele, fundalul, fonturile nu aparţin autorului.

Setări

În Setări (**Figura 20**), utilizatorul îşi poate actualiza detaliile şi preferinţele precum data naşterii, sexul, adresa şi numărul de telefon. Tot aici îşi poate actualiza poza de profil sau să-şi şteargă contul (**Figura 20 dreapta**) în cazul în care nu mai doreşte să folosească aplicaţia.

Datele de mai sus sunt folosite pentru a oferi utilizatorului o experienţă care se pliază pe nevoile sale şi de a filtra, astfel, reclamele pe care le primeşte.

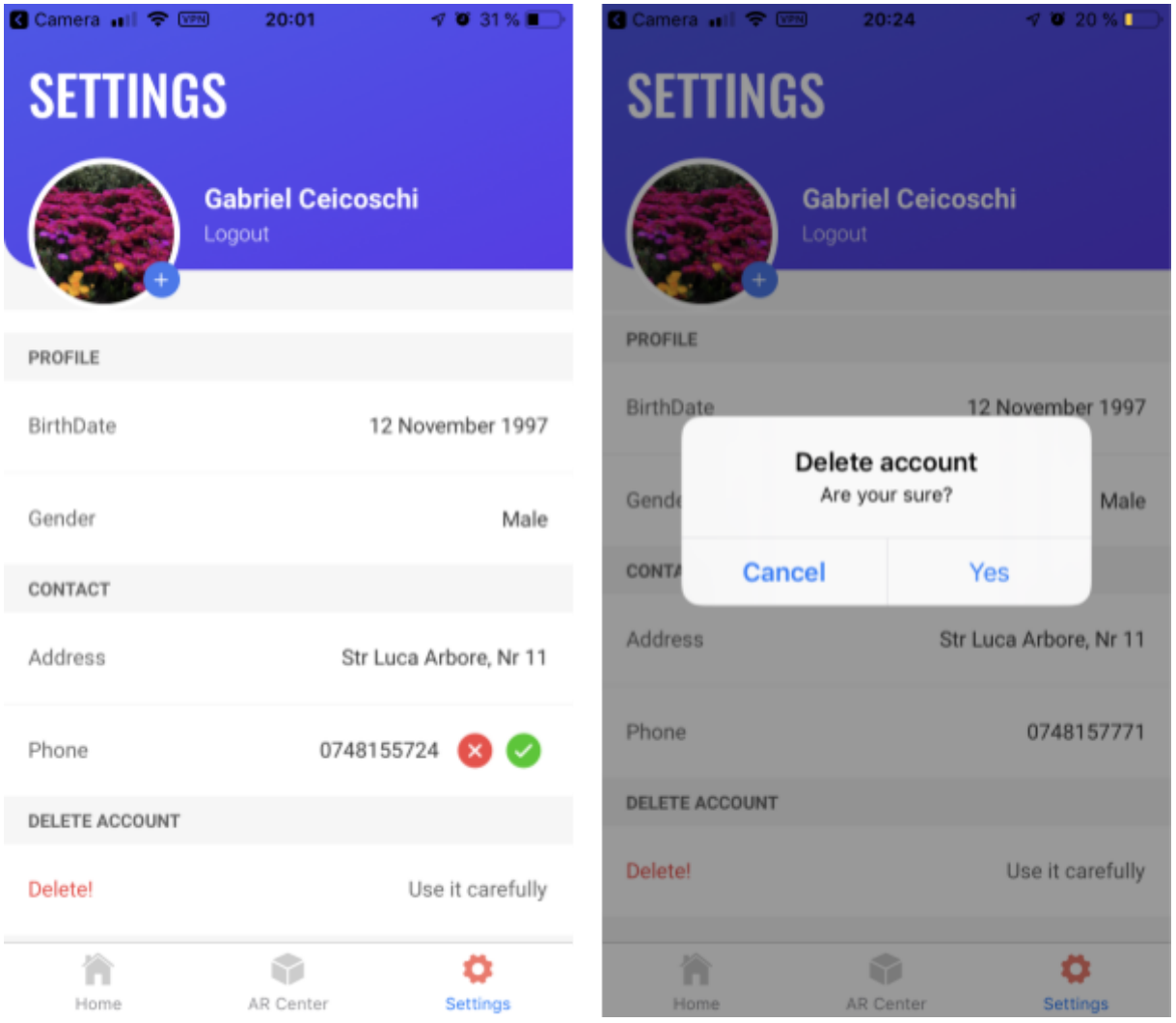

Figura 20 – Pagina de setări şi confirmarea dezactivării contului. Elementele grafice cum ar fi iconiţele, fundalul, fonturile nu aparţin autorului.

Gestionare date

Aplicaţia ARVert necesită gestionarea unor volume relativ mari de date – numeroase reclame şi locaţii, dar şi modele 3D şi date despre utilizatori. Pentru a face acest lucru cât mai eficient, s-a folosit **react-redux**[25]**,** o librărie care poate distribui în mod dinamic datele primite de la API către fiecare modul al aplicaţiei.

Astfel, s-a realizat un arbore principal cu date numit **store (Figura 21),** care cuprinde mai multe ramuri specifice fiecărei tabele din baza de date:

- **userReducer** – aici sunt aduse datele utilizatorului, identificatorul unic al sesiunii curente, legăturile către poza de profil, dar şi detaliile menţionate în **Setări**;
- **adReducer** – se reţin reclamele înregistrate de utilizator, dar şi reclama vizualizată;
- **locationReducer** – asemănător cu adReducer;

[25] https://github.com/reduxjs/react-redux

- **helpReducer** – un arbore care reţine date legate de categoriile de reclame, planuri, roluri etc.

```
export default combineReducers({userReducer,adReducer,
               locationReducer,helpReducer})
```

Figura 21 – Ramurile specifice tabelelor cu date.

Un exemplu de ramură care reţine datele utilizatorului poate fi găsit în **Figura 22.** Aici se poate identifica o stare iniţială care este un obiect cu valori nule. În cazul în care datele sunt aduse cu succes, acest obiect este actualizat cu date specifice.

```
const INITIAL_STATE = {userId: null,userData: null};
export default function userReducer(state = INITIAL_STATE, action) {
 switch (action.type) {
   case GET_USER: {return {...state,};}
   case GET_USER_SUCCESS: {
     return {
       ...state,
       userId: action.payload.id,
       userData: action.payload.data,
     };
   }
}
```

Figura 22 – Exemplu de ramură a arborelui principal cu date.

Fiecare caz în parte este atribuit unei acţiuni. Această acţiune este apelată ori de câte ori sunt necesare datele din Firestore. Un exemplu de acţiune care face un apel GET la ruta /users poate fi găsit în **Figura 23.** Dacă datele sunt primite cu succes, acestea sunt salvate în obiectul explicat mai sus.

```
export const getUser = (id) => {return async (dispatch) => {
   dispatch({type: GET_USER})
   axios.get(`${APIURL}/users/${id}`).then(res => {
       if (res.data) {
         dispatch({
```

```
          type: GET_USER_SUCCESS,
          payload: res.data,
        });
      }
  }).catch(error => {
      dispatch({type: GET_USER_FAIL,
        payload: error
      })
  });}};
```

Figura 23 – Exemplu de acţiune prin care se primesc datele despre utilizatorul curent.

Gestionarea permisiunilor

Pentru ca utilizatorul să aibă o experienţă completă, aplicaţia necesită aprobarea unor permisiuni pentru a accesa camera, galeria, contul Google etc. Acest lucru se realizează prin metoda *askAsync()*, expusă de către modulul *expo-permissions*. Aici, printr-un flag, se cere o permisiune anume. În **Figura 24** este tratat cazul în care se doreşte accesul la galeria dispozitivului, iar flag-ul specificat este *CAMERA_ROLL*. Tot aici se gestionează şi permisiunile pentru locaţie, dar şi pentru notificări. Acestea din urmă fiind folosite pentru a notifica utilizatorul că în apropierea sa se află o reclamă.

```
export async function _getPermissionAsync () {
 const { status } = await Permissions.askAsync(Permissions.CAMERA_ROLL);
 if (status !== 'granted') {
   console.log('Sorry, we need camera roll permissions to make this
work!');}}
```

Figura 24 – Exemplu prin care se solicită permisiunea de a accesa galeria dispozitivului.

Serviciu de email

Pentru ca utilizatorii să fie la curent cu noile oferte, s-a folosit un serviciu prin care se trimit mesaje despre reclamele înregistrate pe platformă. De asemenea, acest serviciu este folosit şi pentru confirmarea email-ului, dar şi pentru resetarea parolei.

Acest serviciu este realizat la nivel de API folosind SendGrid[26] API; un exemplu se găseşte în **Figura 25.** Pentru a face conectarea la SendGrid este necesară o cheie unică generată în consola serviciului.

[26] https://sendgrid.com/docs/

```
export const sendMail = async (req, res) => {
   sgMail.setApiKey(keys.sendGrid);
     if (req.body) {
         const msg = {
             to: req.body.email,
             from: 'noreply@arvert.co',
             subject: req.body.subject,
             text: req.body.subject,
             html: '...'
         };
         if (sgMail.send(msg)) {res.json("Success")
         } else {res.json("Can't send email due empty data")
         }}};
```

Figura 25 – Exemplu prin care se trimit emailuri către utilizatori.

Concluzii

Capitolul 3 însumează cele mai importante aspecte din realizarea soluţiei prezentate şi se aduc exemple semnificative pentru fiecare modul (sau componentă) a platformei ARVert. Astfel, aici este surprins modul de implementare al API-ului şi se oferă exemple de rute şi cum sunt acestea procesate folosind NodeJs şi Express. De asemenea, se explică modul de funcţionare al aplicaţiei mobile - printre care se numără autentificarea, gestionarea datelor utilizatorului, modul de funcţionare al camerei AR, dar şi cum s-au realizat cererile la API. Tot aici se explică cum s-a implementat serviciul de email şi cum s-au gestionat permisiunile de acces la dispozitiv.

Concluziile lucrării

Cu toate aceste lucruri conturate anterior, se poate întări ideea iniţială ca ARVert este o platformă, un concept şi o necesitate – ARVert reprezintă o posibilă soluţie la modul invaziv prin care se face publicitate, iar această lucrare îşi atinge scopul de a explica posibile abordări în materializarea soluţiei.

Prin concentrarea atenţiei asupra utilizatorului final, cercetarea statisticilor din domeniul publicitar şi a abordărilor anterioare, soluţia propusă vine cu un grad de autenticitate ridicat. De asemenea, însumarea de tehnologii noi şi servicii cloud cresc stabilitatea generală a platformei, scalabilitatea soluţiei, dar şi securitatea acesteia. Trebuie menţionat şi faptul că soluţia propusă aduce în acelaşi context diverşi actori – lucru care pune bazele unui ecosistem în care toate părţile au beneficii clare.

Chiar şi aşa, parcursul soluţiei va fi întâmpinat şi de viitoare îmbunătăţiri, iar cel mai important aspect va fi cumpărarea de produse din aplicaţie, dar şi achiziţionarea de planuri tarifare pentru utilizatorii care aleg să posteze reclame. Se va încerca realizarea unei strategii de recompensare a utilizatorilor activi care vizualizează reclame şi a celor care înregistrează locaţii noi.

Pe lângă acestea, un pas important va fi şi adăugarea unui algoritm de învăţare automată care va permite clasarea locaţiilor în funcţii de anumite criterii, dar şi recomandarea de reclame care sunt mai apropiate de comportamentul şi nevoile clienţilor.

Finalizând într-o notă optimistă, se poate spune că ARVert a devenit o adevărată provocare, un obiect inovativ şi un traseu marcat cu şanse mari de succes.

Bibliografie

1. Augmented Reality – The Past, The Present and The Future – https://www.interaction-design.org/literature/article/augmented-reality-the-past-the-present-and-the-future
2. Getting Started With Node.js : A Beginners Guide – https://codeburst.io/getting-started-with-node-js-a-beginners-guide-b03e25bca71b
3. Getting to know Expo – https://docs.expo.io/versions/v33.0.0/
4. Cursul Cloud Computing – https://profs.info.uaic.ro/~adria/teach/courses/CloudComputing/
5. Making an Augmented Reality App with React Native, Expo, and Google Poly – https://medium.com/@ericmorgan1/making-an-augmented-reality-app-with-react-native-expo-and-google-poly-d2acad175a7b
6. React Native: How to Setup Your First App – https://hackernoon.com/react-native-how-to-setup-your-first-app-a36c450a8a2f
7. ARKit – https://developer.apple.com/documentation/arkit
8. Building a REST API with Google Cloud Functions – https://medium.com/@andyhume/building-a-rest-api-with-google-cloud-functions-e0acdf1b2620
9. Building a “Serverless” RESTful API with Cloud Functions, Firestore and Express – https://itnext.io/building-a-serverless-restful-api-with-cloud-functions-firestore-and-express-f917a305d4e6
10. Augmented Reality — A Simple Technical Introduction – https://medium.com/deemaze-software/augmented-reality-a-simple-technical-introduction-83d5e77206b9
11. Getting Started with Firebase on the Web - Firecasts – https://www.youtube.com/watch?v=k1D0_wFlXgo
12. serverless-rest-api – https://github.com/dalenguyen/serverless-rest-api
13. Beginner's Guide for Creating a Serverless REST API using NodeJS over GCF – https://dev.to/levivm/creating-a-serverless-rest-api-using-google-cloud-functions-firebasefirestore-in-10-min-37km
14. fitgoal—The UI Kit that checks itself out in the mirror https://www.invisionapp.com/inside-design/design-resources/fitgoal-ui-kit/

15. Expo-google-login-firebase – https://github.com/nathvarun/Expo-Google-Login-Firebase/blob/master/screens/LoginScreen.js
16. vector-icons – https://expo.github.io/vector-icons/
17. 20 Augmented Reality Stats to Keep You Sharp in 2019 – https://techjury.net/stats-about/augmented-reality/
18. Fonturile Oswald și Roboto – https://fonts.google.com
19. Getting started with SendGrid API – https://sendgrid.com/docs/for-developers/sending-email/api-getting-started/
20. ARKit in React Native Tutorial: The Basics – https://blog.expo.io/arkit-in-react-native-tutorial-the-basics-9f839539f0b9
21. expo-three – https://github.com/expo/expo-three/blob/master/example/screens/AR/Model.js

Anexe

Anexa 1 – Din suita **Firebase** în aplicaţie s-au folosit servicii precum: Firebase Authentication, Firebase Storage, Firebase Realtime Database, Firebase Cloud Functions.

- **Firebase Authentication** – acest serviciu oferă o modalitate rapidă de a autentifica utilizatorul în aplicaţie. Poate realiza autentificarea folosind conturi din reţelelor social-media într-un mod sigur şi foarte eficient. De asemenea, se pot face operaţii complexe cum ar fi *account merging* şi multe altele.
- **Firebase Storage** – este o soluţie s*tand-alone* de a încărca conţinut generat de către utilizatori cum ar fi poze sau clipuri video. Acest serviciu este destinat pentru o mai bună scalare a aplicaţiei, pentru securitate, dar şi pentru o conexiune mai stabilă şi mai rapidă în timpul transferului de fişiere mari.
- **Firebase Realtime Database** – reprezintă o bază de date NoSQL, *cloud-hosted*, care permite stocarea şi sincronizarea datelor în timp real. Are forma unui fişier JSON pe care programatorul îl poate controla după nevoile aplicaţiei.
- **Firebase Cloud Functions** – funcţionează drept backend (API) pentru sistemul curent. Acesta are la bază evenimente care declanşează un răspuns HTTPS. În practică, serviciul poate fi asemănat cu un API în care fiecare funcţie reprezintă un end-point la care se conectează aplicaţia pentru a realiza diverse acţiuni. Întregul mecanism este stocat în cloud, este uşor de menţinut şi poate fi scris în numeroase limbaje printre care se număra şi **JavaScript (NodeJS).**

Anexa 2 – NodeJS reprezintă un mediu pentru a rula cod JavaScript în afara browser-ului web. Acesta are la bază engine-ul Chrome V8 JavaScript şi este o soluţie *ușoară* şi eficientă de a realiza aplicaţii. De asemenea, acesta oferă un package-manager, **NPM**, care este una dintre cele mai largi biblioteci de module *open-source* JavaScript (conform codeburst.io[27]). NodeJS are o arhitectură bazată pe evenimente care se execută într-un singur fir şi este optimizat pentru aplicaţii web şi *server-side* care necesită aţiuni în timp real. În cadrul sistemului ARVert,

ARKit este un framework dezvoltat de **Apple** şi este utilizat în aplicaţiile de iPhone şi iPad pentru a crea experienţe bazate pe realitatea augmentată. ARKit foloseşte camera şi senzorii (giroscopul şi accelerometrul) dispozitivului pentru a înţelege mediul înconjurător astfel încât utilizatorul să se bucure de o experientă simulată a unui model tridimensional. ARKit

[27] https://codeburst.io/all-about-node-js-you-wanted-to-know-25f3374e0be7

Anexa 3 – Rezultatele interviul menţionat în capitolul **Metodologia Soluţie** https://docs.google.com/spreadsheets/d/1uCfWtjHlNDF31zx2hRCP8HeOc3dWtuiOYvKJ2-gXn28/edit?usp=sharing

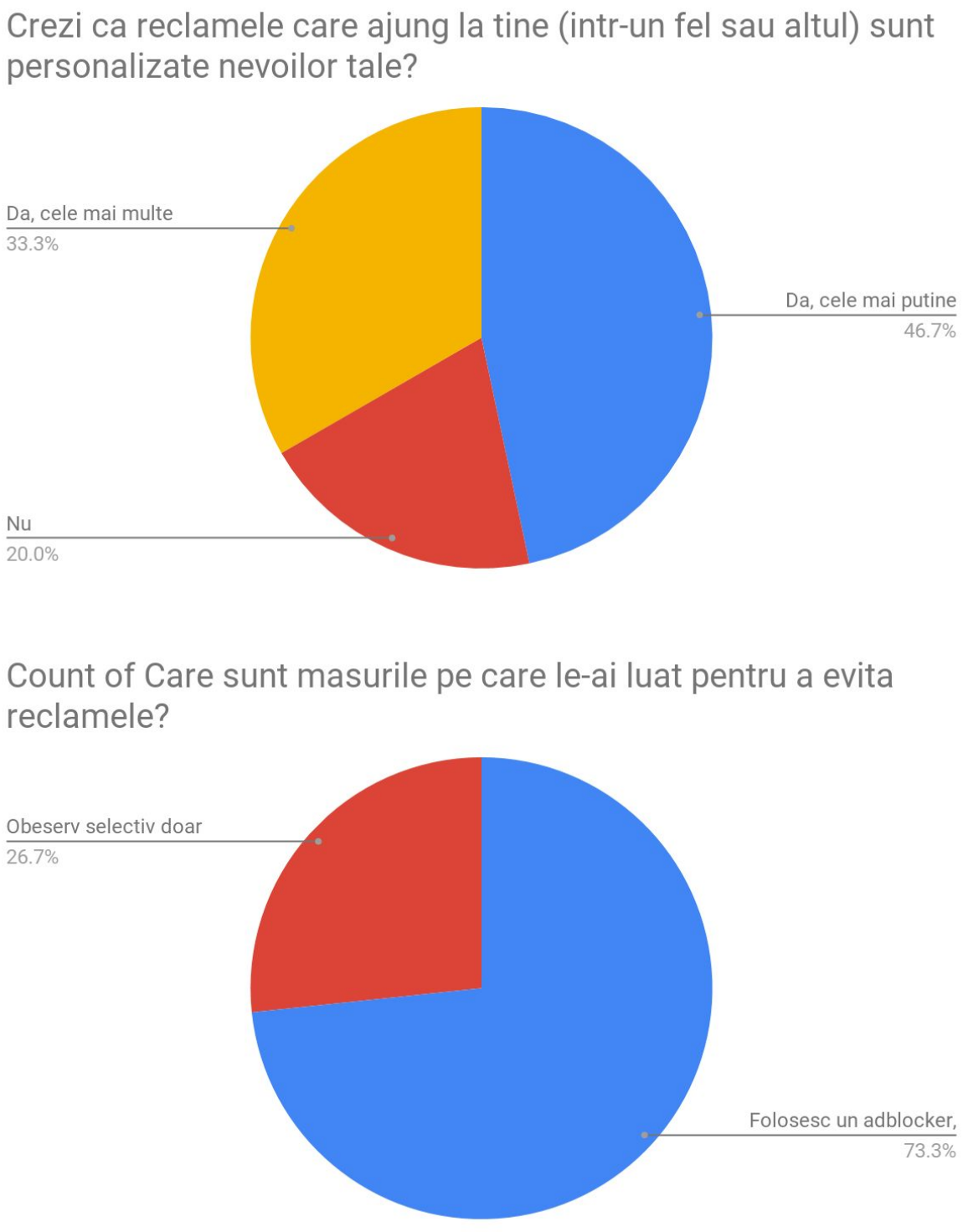

Figura 26 – Graficul studiului individual asupra unor persoane interesate de această soluţie

Anexa 4 – Statistici privind folosirea soluțiilor de blocare a reclamelor (Sursa: globalwebindex.com)

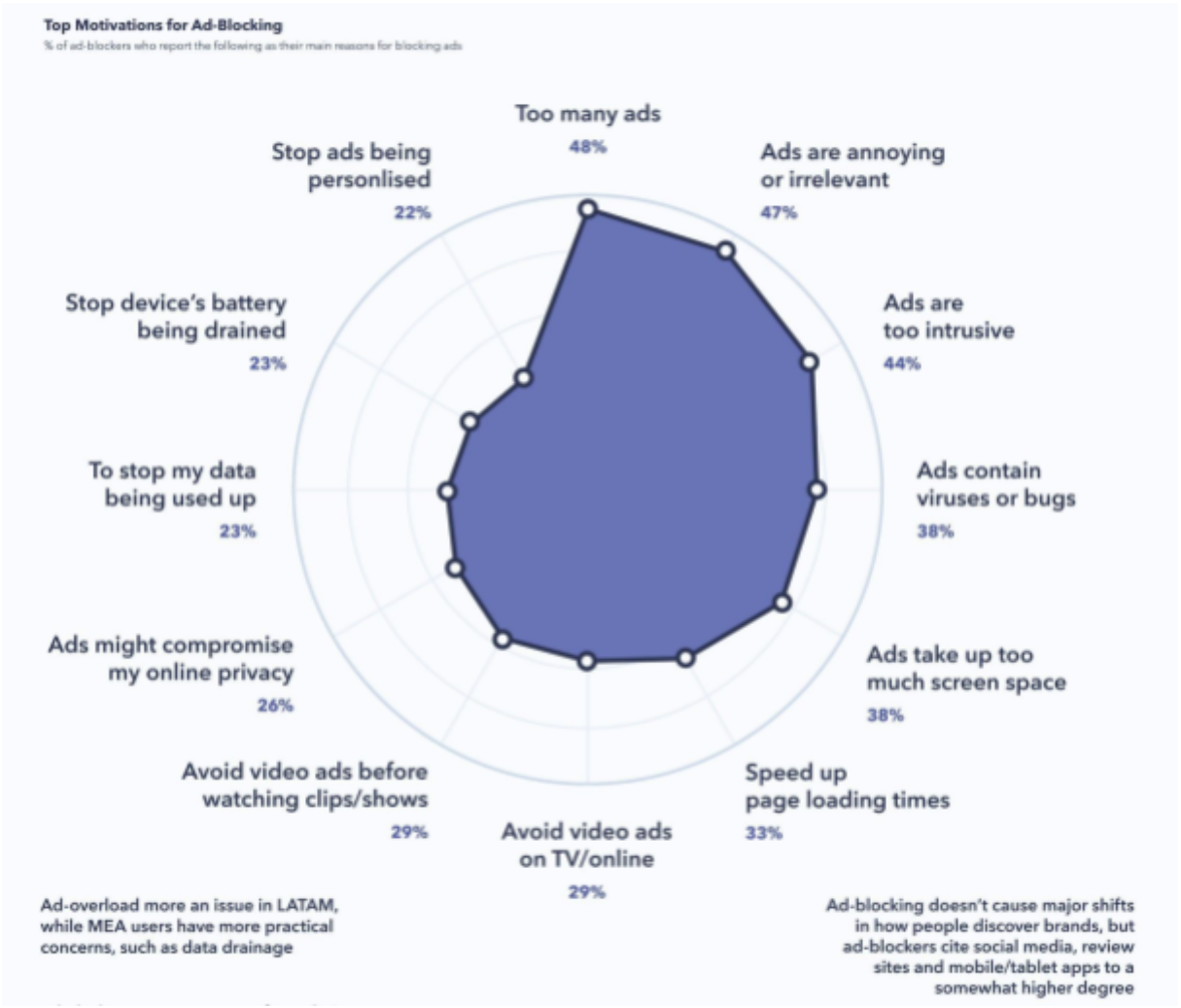

Figura 27 – Graficul motivelor pentru care se folosesc soluții de blocare a reclamelor