UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA
DE
MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA TEHNOLOGIA INFORMAȚIEI

Lucrare de licență

PRELUCRAREA IMAGINILOR ÎN CADRUL APLICAȚIILOR MOBILE

Absolvent
Elena-Diana Tudosă

Coordonator științific
Conf.univ.dr. Radu Boriga

București, iulie 2020

Abstract

Este binecunoscut faptul că tehnologia avansează constant în aria dispozitivelor mobile, fapt determinat de portabilitatea acestora. Astfel, tot mai multe funcții devin integrate în aplicații, transformând telefon de odată într-un dispozitiv multifuncțional. Odată cu această evoluție, s-a căutat constant eficientizarea proceselor de dezvoltare a aplicațiilor mobile. Astfel, plecând de la ideea de economire de timp și bani, a apărut conceptul de aplicație hibridă, compatibilă cu cele două sisteme de operare principale: Android și iOS.

Lucrarea realizată este un studiu asupra modalităților de a obține aplicații mobile hibride, punând accent pe librăria dezvoltată de Facebook, React Native. Avantajul principal al acestei tehnologii îl reprezintă mecanismul de transformare al codului din Javascript în limbaj nativ și, odată cu aceasta, al elementelor de interfață în componente native. Astfel, aplicația dezvoltată va imita comportamentul unei aplicații native, din punct de vedere al aspectului, dar și al performanței.

React Native a fost scris astfel încât dezvoltarea unei aplicații să fie cât mai asemănătoare procesului realizat în cazul scrierii unei pagini web în React. Datorită acestui fapt, în primul capitol este prezentată istoria tehnologiilor utilizate pentru dezvoltarea aplicațiilor web, având ca focus Single Page Applications (SPA). Mai departe, este analizată biblioteca React, alături de funcționalitățile sale.

Expo reprezintă un framework construit special pentru dezvoltarea aplicațiilor în React Native. Acesta conține un set de tool-uri și servicii construite special pentru a ajuta dezvoltatorul în procesele de develop, build și deploy.

În practică, există numeroase aplicații care se ocupă de editarea fotografiilor, fapt datorat atracției omului față de frumos, dar și expansiunii rețelelor de socializare și influenței lor. Subiectul secundar abordat în cadrul lucrării îl reprezintă prelucrarea de imagini prin intermediul aplicării de filtre asupra matricei de pixeli asociată acesteia. Aplicația dezvoltată utilizează biblioteca PixiJS, ce vine cu metode și filtre integrate pentru a fi aplicate asupra unei fotografii.

În urma unei analize asupra filtrului Sepia, s-a dezvoltat un algoritm pe baza căruia să se creeze matrici de transformare particularizabile, având ton dominant o nunță dată. Pentru a demonstra

rezultatul obținut, s-au realizat trei exemple, pornind de la trei tonuri diferite de culoare, și au fost incluse în aplicație.

În finalul lucrării, este explicat modul în care a fost dezvoltată aplicația și cum au fost îmbinate tehnologiile descrise, urmând apoi prezentate perspective de dezvoltare ulterioare.

Abstract

It is a well known that technology has been consntantly evolving in the area of mobile devices, by virtue of their portability. Therefore, more and more features become integrated into the mobile devices, thus contribuiting to their versatility. Due to this evolution, efficienlty developing mobile apps was constanlty sought after. And so, having in mind the idea of saving energy and time, the concept of hybrid mobile application was born.

The thesis is a study regarding the ways of developing hybrid mobile applications, making an emphasize on Facebook's library, React Native. The main advantage of this technology is its engine that adapts Javascript code to native language and user interface components. Therefore, the developed app will mimic a native app, regarding its aspect and performance.

React Native was written in order for the developing process to be similar with the one developing a web page using React. Therefore, the history regarding developing methodologies for web applications is presented in the first chapter, with a main focus on Single Page Applications. Following the next chapter, the React library is presented alongside its functionalities.

Expo is an open-source framework built for React Native applications. It comes with a set of tools and services that help the developer în the processes of developing, building and deploying the app.

On the market, there are many applications built for editing images, due to the natural human's attraction for beauty, and also because of the growth of social media networks and their influence. The second point of the thesis is characterised by image processing using filters on the image's matrix of pixels. The application is making use of PixiJS library and its methods and filters.

As a result of analyzing the Sepia filter, an algorithm was developed based on which customizable matrix was obtained, having the main hue a given color. In order to prove the algorithm's accuracy, three cases were analized, given three different shades of color, that were included in the app.

Nearing the end of the thesis, the development process and the way of using the described technologies are presented, followed by possible new features.

Cuprins

I. Introducere....9
II. Single Page Application....11
II.1 Server Side Pages....11
II.2 Ajax....11
II.3 Single Page Application (SPA)....11
II.4 Progressive Web Applications (PWA)....13
III. React....14
III.1 Virtual Document Object Model....15
III.2 Lifecycle methods....15
III.2.1 ShouldComponentUpdate....16
III.2.2 ComponentDidMount....16
III.2.3 ComponentWillUnmount....16
III.2.4 Render....16
III.3 Javascript XML (JSX)....16
III.4 React Hooks....16
IV. Aplicații mobile....17
IV.1 Aplicațiile hibride....18
IV.1.1 Avantaje....18
IV.1.2 Dezavantaje....19
IV.2 Aplicații native....20
IV.2.1 Avantaje....20
IV.2.2 Dezavantaje....20
V. React Native....22
V.1 Performanța oferită de React Native....24
VI. Prelucrarea imaginilor....26
VI.1 Proprietăți....28
VI.1.1 Contrast....28
VI.1.2 Luminozitate....29

VI.1.3 Tonuri de gri....29
VI.1.4 Nuanță....30
VI.1.5 Saturație....31
VI.2 Filtre....32
VI.2.1 Browni (Maroniu)....32
VI.2.2 Dots (Puncte)....32
VI.2.3 Emboss (Gravare)....33
VI.2.4 Pixelate (Pixelare)....34
VI.2.5 Cross Hatch (Hașurare încrucișată)....34
VI.2.6 Noise (Zgomot)....35
VI.2.7 Old Film (Film vechi)....36
VI.2.8 RGB Split (Împărțire RGB)....37
VI.2.9 Bulge Pinch (Umflătură-ciupitură)....37
VI.2.10 Motion Blur (Neclaritate de mișcare)....38
VI.2.11 Advanced Bloom (Înflorire avansată)....39
VI.2.12 Blur (Estompare)....39
VI.2.13 Negative (Negativ)....40
VI.2.14 Night (Noapte)....41
VI.3 Studiu dezvoltat pe baza filtrului Sepia....42
VII. Soluția software dezvoltată....48
VII.1 Dependințe Node Packege Manager (NPM)....50
VII.1.1 Navigarea....50
VII.1.2 Importarea de fișiere din telefon....52
VII.1.3 Stocarea datelor aplicației în memoria dispozitivului....54
VII.1.4 Aplicarea filtrelor asupra imaginilor....55
VII.2 Structura aplicației....57
VII.2.1 App....57
VII.2.2 Home....58
VII.2.3 Gallery....59
VII.2.4 Edited Gallery....59
VII.2.5 Edit Photo....60

VII.2.6 Personalize.....61
VII.3 Alte aspecte ale aplicației.....62
VII.3.1 Enums.....63
VII.3.2 Filters.....63
VII.3.3 Style.....63
VIII. Concluzii.....65
IX. Bibliografie.....66

I. Introducere

În cadrul lucrării, sunt abordate două subiecte tehnologice actuale. Subiectul principal constă în dezvoltarea de aplicații mobile hibride, prin intermediul bibliotecii React Native. Subiectul secundar îl reprezintă prelucrarea imaginilor digitale, prin aplicarea de filtre.

React Native reprezintă o bibliotecă pentru Javascript, ce se ocupă de generarea elementelor drept componente native. Avantajul principal și motivul pentru care această bibliotecă diferă de celelalte soluții pentru a dezvolta aplicații hibride este performanța pe care o are aplicația construită. Codul de dezvoltare al unei aplicații scrise în React Native se realizează în Javascript. Platformele suportate de această bibliotecă sunt Android și iOS, prin urmare, engine-ul pe care îl oferă React Native, va transforma ulterior rezultatul în cod Kotlin/Java pentru Android, respectiv Swfit/Objective-C pentru platforma iOS.

O modalitate de a dezvolta aplicații în React Native este utilizarea platformei Expo. Aceasta vine cu un set de tool-uri și servicii construite special pentru a ajuta în procesele de dezvoltare, build și deploy.

Mai departe, în cadrul lucrării este abordat subiectul prelucrarării imaginilor. O imagine reprezintă o matrice de pixeli, iar un pixel este descris, de regulă, prin 3 canale de culoare: roșu, verde și albastru. Pentru a edita aspectul unei imagini, este suficientă modificarea valorilor acestei matrici, pe oricare canal. Un rezultat uniform presupune dezvoltarea mai multor reguli sau formule matematice aplicate asupra matricei de pixeli. WebGL vine cu soluții rapide și definite intern pentru prelucrarea imaginilor. Mai mult de atât, biblioteca PixiJS utilizează resursele oferite de WebGL și oferă o cale de a utliza funcționalitățile acestuia în contextul limbajului Javascript. Astfel, au fost extrase filtre definite de această bibliotecă pentru a fi mai departe analizate și utilizate în cadrul studiului.

Analizând clasicul filtru sepia, a fost dezvoltat un algoritm pe baza căruia să se genereze matricea de pixeli asociată unui filtru sepia particularizat, al cărui ton de culoare dominant să fie o nuanță dată. Pentru a exemplifica rezultatele obținute, s-au ales trei tonuri diferite după care s-au creat trei filtre.

Scopul aplicației dezvoltate este de a descrie modul în care se pot îmbina aceste tehnologii, React Native și PixiJS, de a evidenția performanța aplicației obținute și de a vizualiza modul în care se modifică imaginile digitale în urma aplicării filtrelor descrise. De asemenea, funcționalitatea distinctivă a aplicației, spre deosebire de cele deja existente pe piață, este posibilitatea utilizatorului de a-și crea propriile filtre, prin atribuirea de diferite valori matricei de transformare. Această funcționalitate poate fi considerată atât cu rol educativ, pentru a înțelege efectele pe care diferiți parametri îi au asupra unei transformări, cât și cu scopul de a oferi utilizatorului o gamă cât mai largă de filtre, având caracter unic și personalizat.

II. Single Page Application

Tehnologia a avansat exponențial în ultimii ani și, dat fiind acest fapt, s-au schimbat și tehnicile și modul de gândire în implementarea oricărei soluții. Astfel, s-au modificat treptat telefoanele clasice în smartphone-uri, plata cash în plata UPI/Paytm cash, monoliticul în microservicii etc. Aceste modificări sunt inevitabile și, totodată, foarte rapide.

Conform [6], atunci când paginile web au apărut, presupuneau doar HTML static, simplist. Ulterior, au apărut două cerințe principale care, odată cu ele, au ridicat o provocare în modalitatea de dezvoltare: vizualizarea datelor mereu actualizate și efectuarea de acțiuni din interfață.

În decursul timpului, au apărut mai multe soluții pentru implementarea acestor cerințe. Conform [1], există patru modalități de a dezvolta o aplicație web:

II.1 Server Side Pages

Folosind limbaje precum PHP, JSP, o variantă de implementare a vizualizării dinamice a site-ului constă în a aduce informația mereu din baza de date, încapsulată într-un template HTML. Pentru efectuarea de acțiuni, se face un request către server, se generează un nou template pe baza modificărilor realizate în baza de date, după care întregul HTML este trimis către client, iar pagina este reîncărcată astfel.

II.2 Ajax

Odată cu introducerea conceptului de AJAX, s-a eficientizat efectuarea acțiunilor prin adăugarea conceptului de call API asincron. Pe de altă parte, vizualizarea datelor a rămas în continuare generată de template-uri venite ca răspuns HTML de pe server.

II.3 Single Page Application (SPA)

Conceptul de SPA presupune, pe scurt, decuplarea front-end-ului de back-end, adică a view-ului de datele stocate în server. Toate template-urile de view și logica de front-end sunt încărcate odată cu pornirea aplicației web, urmând ca datele din back-end să fie aduse

doar în momentul în care trebuie afișate sau folosite. Astfel, există șansa ca o mare parte a datelor de pe server să nu fie solicitată pe parcursul unei navigări pe site.

Single Page Applications utilizează AJAX si HTML5 pentru a construi aplicația propriu-zisă. În schimb, pentru a facilita utilizarea acestor concepte, s-au dezvoltat mai multe framework-uri sau biblioteci care folosesc acest concept: React.js, Angular, Vue, Ember.
Beneficiile utilizării Single Page Application:

- Crește semnificativ viteza website-ului; HTML-ul, CSS-ul și scripturile sunt încărcate o singură dată pe parcursul unei vizite
- Caching; un Single Page Application poate stoca în cache orice date. Printr-un singur request către server, poate salva toate datele necesare în cache, după care să aibă capacitatea de a funcționa offline
- User experience liniar; interfața poate fi construită astfel încât navigarea să nu presupună doar clickuri pentru schimbarea conținutului
- Debugging folosing Chrome: Chrome a dezvoltat developer tools care eficientizează procesul de debugging

Dezavantajele utilizării Single Page Application:

- Search Engine Optimization: dat fiind faptul că informațiile se încarcă doar atunci când sunt necesare, website-urile construite ca Single Page Application nu sunt optime din punct de vedere al Search Engine Optimization
- Istoricul browser-ului: o aplicație Single Page Application nu salvează vizitele utilizatorului între stări. Astfel, browser-ul redirecționează utilizatorul către pagina anterioară, nu către starea anterioară.
- Probleme de securitate: Single Page Applications sunt mai puțin imune la atacurile cross-site scripting (XSS) decât aplicațiile multi-page. Folosind XSS, hackerii pot injecta client-side scripturi în aplicație care să execute interogări la nivelul bazei de date

Single Page Applications se mulează perfect pentru construirea de platforme dinamice cu un volum mic de date. De asemenea, dacă aplicația urmează a fi dezvoltată ulterior pe mobile,

este din nou o bună variantă alegerea modelului Single Page Application. Dezavantajul principal pe care îll are este Search Engine Optimization slab.

II.4 Progressive Web Applications (PWA)

PWA este un concept relativ nou, dezvoltat în 2019. Acesta se ocupă în special de prelucrarea și stocarea datelor, astfel încât să fie evitate apeluri multiple către server pentru aceeași informație. Așa cum îi spune și numele, un PWA va încărca pe parcurs datele utilizate, le va stoca în cache, iar revenirea la ele va presupune doar extragerea lor de acolo. Acest concept este utilizat și în cadrul aplicațiilor mobile și permite utilizarea lor offline.

III. React

React reprezintă o bibliotecă JavaScript realizată pentru a construi interfețe pentru aplicații web. A fost concepută de Facebook și se află în continuă dezvoltare, fiind cea mai utilizată bibliotecă pentru realizarea interfețelor. Imediat după React, se situează Angular și View.

La baza aplicațiilor construite cu React stau componentele. O componentă reprezintă o "bucată" din interfață, izolată și reutilizabilă. În procesul de dezvoltare a unei aplicații web, se implementează mai multe componente ce ulterior sunt compuse pentru a construi interfața.

Așa cum este amintit și în [5], fiecare aplicație React are minimum o componentă, numită Root Component. Aceasta este echivalenta întregii aplicații și conține mai multe componente-copil. Astfel, structura rezultată poate fi asemănată cu cea de arbore, astfel descrisă în Figura III.1.

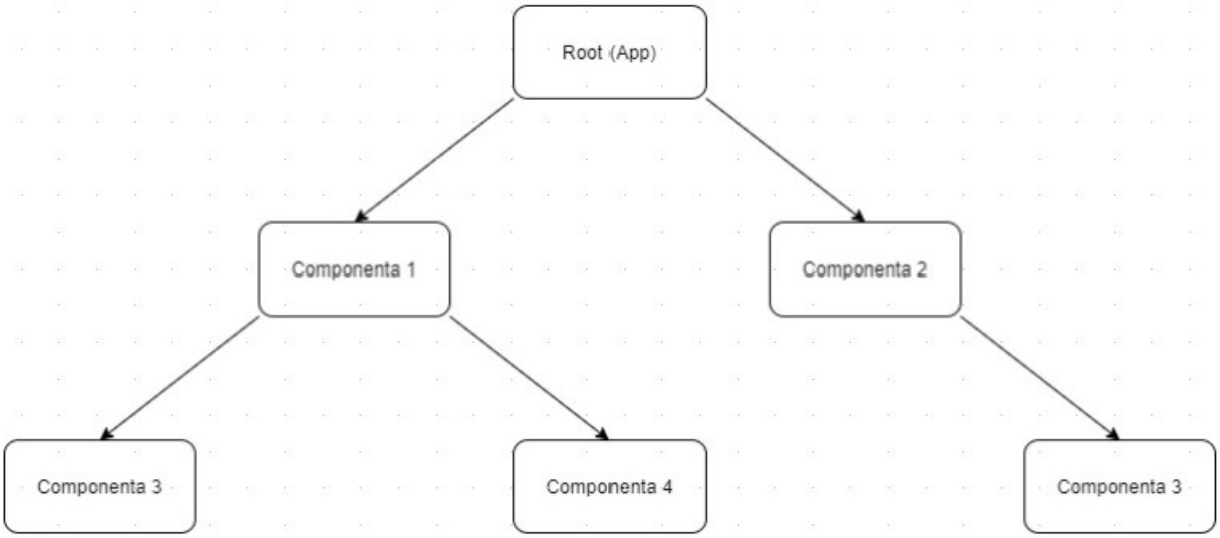

Figura III.1: Arhitectura unei aplicații React Native

React se ocupă de afișarea datelor, fiind necesare biblioteci adiționale pentru construirea unei pagini web complete care să includă rutare, să managerieze starea aplicației și să realizeze cererile către server.

React presupune crearea de entități, numite componente, care sunt ulterior generate ca elemente HTML în Document Object Model (DOM). Aceste componente pot primi valori, cunoscute drept "props", care ulterior să fie afișate drept conținut.

Există două modalități de a crea componente în React:

- Componente funcționale

  Declarate într-o funcție care returnează cod Javascript XML

```
const Greeting = (props) => <div>Hello, {props.name}!
</div>;
```

Figura III.1: Exemplu de componentă declarată prin cod JSX [13]

- Componente clase

  Declarate folosind clase ES6. Se mai numesc componente "stateful" deoarece state-ul lor poate stoca valori care ulterior să fie pasate componentelor copil prin proprietăți

```
class ParentComponent extends React.Component {
  state = { color: 'green' };
  render() {
    return (
      <ChildComponent color={this.state.color} />
    );
  }
}
```

Figura III.2: Exemplu de componentă declarată drept clasă [13]

III.1 Virtual Document Object Model

O funcționalitate pe care React o aduce și o utilizează este Virtual Document Object Model (Virtual DOM). În cadrul acestuia, doar componentele care își modifică starea sunt generate din nou, eficientizând modalitatea de reîncărcare a unei pagini cu datele actualizate.

III.2 Lifecycle methods

Lifecycle methods reprezinta hook-uri care permit executarea acțiunilor pe componente pe durata lor de viață.

Printre acestea, se numără:

III.2.1 ShouldComponentUpdate

Permite dezvoltatorului să prevină recrearea nenecesară a unei componente.

III.2.2 ComponentDidMount

Este apelată în momentul în care o componentă este creată în interfața utilizatorului, adică i se realizează un nod Document Object Model.

III.2.3 ComponentWillUnmount

Este apelată imediat înainte ca o componentă să își încheie durata de viață și se utilizează adesea pentru a curăța aplicația de resursele pe care componenta respectivă le utiliza în particular.

III.2.4 Render

Reprezintă cea mai utilizată și importantă metodă de lifecycle, obligatorie oricărei componente. Este apelată de fiecare dată când starea (state-ul) unei componente este actualizată și această modificare trebuie reflectată în interfața cu utilizatorul.

III.3 Javascript XML (JSX)

JSX reprezintă o extensie a sintaxei limbajului Javascript ce furnizează o cale de a structura generarea componentelor.

Având în vedere că HTML reprezintă cea mai utilizată cale de a realiza acest fapt, asemănarea JSX-ului cu acesta oferă developerilor ușurința utilizării unei sintaxe familiare. De asemenea, componentele pot fi generate și utilizând JavaScript pur, însă această metodă nu este tocmai folosită.

III.4 React Hooks

Hook-urile reprezintă funcții care permit dezvoltatorilor să se agațe, pe parcursul duratei unei funcții sau componente React, de aceasta.

Cel mai întâlnit hook îl reprezintă useState care permite adăugarea de stări într-o componentă funcțională

IV. Aplicații mobile

De la momentul apariției primului apel telefonic (1973) [12] și până în prezent, tehnologia a crescut la o scară exponențială. Acest fapt se datorează avantajului principal pe care telefonul mobil îl prezintă: portabilitatea. Într-un dispozitiv de dimensiunea unui telefon mobil s-au reușit a fi înglobate sute de servicii, pornind de la funcția sa de bază, apel telefonic, și extinzându-se ulterior spre GPS, aparat foto, portofel electronic etc. Extinderea funcționalităților pe care un dispozitiv mobil le poate avea este imensă, dar principala funcție modernă a acestuia a devenit cu siguranță accesul la internet.

Există trei tipuri de aplicații mobile:

- Web
- Native
- Hibride

Acestea sunt prezentate, alături de avantaje și dezavantaje, în articolul [11].

Prima categorie menționată, aplicațiile web, reprezintă site-uri web construite folosind tehnologiile specifice, HTML / CSS / Javascript. Acestea rulează pe un browser, precum Safari sau Chrome, și sunt construite pentru a fi accesate de pe calculator. Cu toate acestea, se pot adăuga clase de stilizare special dedicate astfel încât aspectul lor să fie unul plăcut și experienței cu telefonul mobil. Această tehnică este utilizată în cadrul oricărei pagini web de actualitate și a devenit o cerință esențială. Cu toate acestea, tipul acesta de aplicații este îndepărtat de termenul propriu-zis de aplicație mobile, întrucât nu accesează nicio funcționalitate specifică unui telefon mobil (cameră foto, gestures etc).

Primul lucru de menționat când vine vorba despre comparație între aplicații mobile native și aplicații hibride este acela că aplicațiile native sunt dezvoltate special pentru o platformă, iOS, Android etc. Acestea sunt scrise în limbaje de programare care sunt suportate de platformele respective (Swift/Objective-C pentru iOS, respectiv Java/Kotlin pentru Android).

Pe de altă parte, aplicațiile hibride sunt scrise prin intermediul tehnologiilor web, cum ar fi Javascript, CSS și HTML. Practic, acestea reprezintă aplicații web înglobate într-un înfășurător (wrapper) nativ ce permite comunicarea cu platforma dispozitivului și accesul la funcționalitățile

sistemului de operare. Rezultatul dezvoltării reprezintă o aplicație compatibilă cu ambele platforme mobile.

Un lucru pe care îl au în comun ambele modalități de dezvoltare este că aplicațiile rezultate pot fi distribuite pe market place-urile oficiale, precum App Store și Google Play.

Printre aplicațiile hibride întâlnite adesea pe piață se numără Uber, Instagram și Twitter. Distincția între cele două tipuri de aplicații nu este una vizibilă utilizatorului, ci afectează strict partea de dezvoltare și investire.

IV.1 Aplicațiile hibride

După cum am spus și mai devreme, aceste aplicații sunt inițial website-uri împachetate în containere native. Pentru a accesa funcționalitățile device-ului, aplicațiile hibride utilizează API-uri specializate. Acest fapt conduce la limitarea posibilităților de dezvoltare, deoarece trebuie să existe un terț care să acceseze funcționalitatea respectivă.

Aplicațiile hibride sunt potrivite atunci când produsul se bazează pe conținut. Pentru funcționalități complexe, există riscul de a crește exponențial dificultatea de implementare.

IV.1.1 Avantaje

- Aplicația dezvoltată rulează atât pe Android, cât și pe iOS
    - Acesta reprezintă avantajul principal atunci când vine vorba de aplicații hibride. Practic, același cod sursă generează o aplicație funcționabilă pentru ambele platforme
    - Atunci când obiectivul principal este de a ținti o piață cât mai largă de clienți, aceasta poate reprezenta o soluție optimă
- Timp de dezvoltare mai scăzut
    - Având în vedere că se dezvoltă o singură soluție software în schimbul a două, timpul de livrare este mai scăzut

    - Asemenea este și timpul de testare, deoarece nu toate funcționalitățile vor trebui verificate pe ambele platforme de către inginerii de calitate

      Mai ușor de modificat și de updatat
    - Modificarea unui feature sau rezolvarea unui bug se realizează într-un singur loc
    - Fiind un webview, fiecare update realizat asupra aplicației va fi preluat automat la următorul refresh. Prin urmare, nu este necesar update-ul aplicației din Store de către utilizator, așa cum se procedează pentru aplicațiile native
- Costurii de dezvoltare mai scăzute
    - Costul unei aplicații hibride care țintește ambele platforme este aproximativ același cu cel pentru dezvoltarea unei singure aplicații native
    - Deoarece costul unei soluții software se calculează în funcție de timpul petrecut pentru dezvoltarea sa, automat costul este diminuat
    - De exemplu, în țările Est-Europene, printre care și România, prețul unei ore de dezvoltare este între 40-50$, iar timpul aproximativ de dezvoltare al unei aplicații de complexitate medie este de 600 de ore, costul dezvoltării ar ajunge la $300k

IV.1.2 Dezavantaje

- Capabilități de dezvoltare limitate
    - Este necesar un third-party pentru a integra funcționalități
    - Calitate User Experience scăzută
    - Deoarece interfața trebuie să fie undeva la mijloc între mediul Android și iOS, realizarea unei bune experiențe care să satisfacă utilizatorii ambelor platforme este o provocare pentru dezvoltatori
    - Există funcții specifice iOS ce sunt diferite de cele de Android și invers, iar acest fapt poate conduce la evitarea folosirii lor. Un exemplu poate fi detectarea și însemnătatea gestures

IV.2 Aplicații native

IV.2.1 Avantaje

- Calitate User Experience ridicată
    - Deoarece focusul este într-o singură parte, se poate realiza un design custom, definit prin setarea gesturilor, elemente proprii și unice
    - Pot fi folosite cu totul în modul offline
- Performanță ridicată
    - Consum de energie și memorie mai scăzute
    - Codul nativ rulează mult mai rapid, prin urmare și aplicația
- Securitate
    - Crearea de aplicații native este singura cale de a garanta utilizatorilor protecția datelor
    - Întreaga putere a hardware-ului trebuie să proceseze taskuri, iar aplicațiile hibride nu pot accesa această resursă la maximum
- Paletă de capabilități largă
    - Acces integral la baza de date și funcționalitățile hardware ale device-ului
    - Nu este necesară integrarea unui third-party
    - Personalizare
    - Dezvoltarea nativă este singura cale garantată de a menține un layout stabil și la un nivel ridicat de complexitate

IV.2.2 Dezavantaje

- Costul de dezvoltare ridicat
    - Procesul de dezvoltare al unei aplicații native este mai complex și necesită resursă umană experimentată

- Timpul de dezvoltare ridicat
    - Dacă se dorește implementarea de soluții pentru ambele platforme, trebuie dezvoltate două aplicații, prin urmare timpul va fi mai îndelungat și este recomandată dezvoltarea lor în paralel

În concluzie, aplicațiile hibride sunt o soluție perfectă pentru proiecte simple și orientate pe afișarea de conținut. Această variantă este de asemenea de luat în considerare atunci când bugetul este limitat, dar se dorește implementarea unei soluții pentru ambele platforme într-un timp limitat. Pe de altă parte, dacă funcționalitățile personalizate și User Experience-ul sunt factori determiniști pentru succesul proiectului, soluția ideală este implementarea aplicațiilor native.

V. React Native

React Native este o bibliotecă care extinde funcționalitățile pe care le oferă React și permite construirea aplicațiilor mobile native (care rulează atât pe IOS, cât și pe Android), prin generarea elementelor în componente native.

Paradigma React Native este “Learn once, write everywhere” [7] și se referă la flexibilitatea unui dezvoltator de a crea atât aplicații web, cât și mobile, utilizând aceeași tehnologie.

O aplicație dezvoltată în React Native nu este o pagină web ce rulează pe browser-ul dispozitivului și nu este nici o aplicație web gazduită într-un webview (ex: Ionic). În acest ultim caz, s-ar aplica un wrapper peste aplicație pentru a fi flexibilă pe cele două platforme, dar în schimb nu ar converti și codul. Astfel, performanța acestor aplicații este scăzută. Avantajul este disponibilitatea tuturor tehnologiilor web în procesul dezvoltării.

React Native construiește o aplicație hibridă în care tot codul este convertit în cod nativ.

Conform [2], ceea ce diferențiază această tehnologie de restul framework-urilor de dezvoltare de aplicații hibride este modul în care elementele sale sunt generate drept componente native. Asemenea bibliotecii React pentru aplicații web, aplicațiile sunt scrise în React Native folosind o combinație de JavaScript și XML, cunoscută drept Javascript XML (JSX). Acest cod este convertit de React Native, utilizând API-urile native de generare din Objective-C/Swift, respectiv Java/Kotlin, astfel încât aplicația rezultată va genera componente de interfață native platformei, iar aspectul și experiența va fi asemenea unei aplicații native.

Pentru construirea interfeței, dezvoltatorii folosesc elementele DOM. Acestea sunt componente aflate în pachetul ‘react-native’. Spre exemplu, elementul devine <View>, iar și <p> au ca echivalent <Text>. Acestea sunt doar blocuri de bază, pentru cele mai complexe se construiesc componente particularizate de dezvoltator sau se adaugă pachete de pe Node Package Manager. Pentru stilizare, se utilizează un obiect StyleSheet, care conține obiecte drept ‘clase’ cu proprietăți similare CSS.

Expo este un framework si o platformă open-source utilizată pentru a construi aplicații hibride pentru telefon, valabile atât pe Android, cât și pe iOS. Conține un set de tool-uri și servicii

construite special pentru React Native ce ajută dezvoltatorul în procesele de develop, build și deploy.

Fiind scrisă pentru React Native, limbajul utilizat în dezvoltare este Javascript sau Typescript. Printre serviciile oferite de Expo, se regăsesc seturi de API-uri ce oferă acces la resursele dispozitivului, precum camera foto, fișiere media, haptics (interacțiuni cu ecranul) și așa mai departe. Prin intermediul utilizării Expo, sunt introduse noi opțiuni de configurare a aplicației, opțiuni care, printr-o dezvoltare utilizând doar React Native, presupuneau modificări la nivel de proiect iOS și Android simultan. Pentru aceste cazuri, Expo vine cu soluții și componente gata construite, iar timpul de dezvoltare poate fi redus de la 1-2 zile la 10 minute. Printre funcționalitățile menționate mai sus, se numără push notifications, splash screen, Facebook login etc.

Un subiect adesea abordat în cadrul găsirii soluției celei mai potrivite pentru dezvoltarea de aplicații web este cross-platform. Dezvoltarea unei aplicații mobile este un proces costisitor, prin urmare, beneficiile dezvoltării pe mai multe platforme este imens. Se dorește reutilizarea abilităților, a codului și a resurselor umane. Ideea principală este că cross-platform poate face dezvoltarea mobile mult mai productivă, mai rapidă. Cheia în dezvoltarea closs-plaftorm este Javascript, datorită compatibilității sale cu cele mai populare trei platforme: web, iOS și Android. Acest concept nu este unul nou, adus de React Native, ci utilizat încă din 2009 de către Cordova și PhoneGap. În schimb, modul de abordare adus de acestea din urmă nu a fost unul care să fi prins. Motivul eșecului a fost performanța scăzută, lucru critic pentru o aplicație.

În cadrul React Native, conceptul de cross-platform a fost dezvoltat într-un mod diferit. În figura V.1 este o aplicație scrisă în React Native pur. Principala observație este că în ecran nu este generat un webview cu HTML, ci fiecare view reprezintă un element nativ platformei. De exemplu, pentru navigare este folosit UINavigationController din iOS, image view-ul este UIImageView etc. Prin urmare, interfața este nativă, iar Javascript instanțiază view-uri native, în spate.

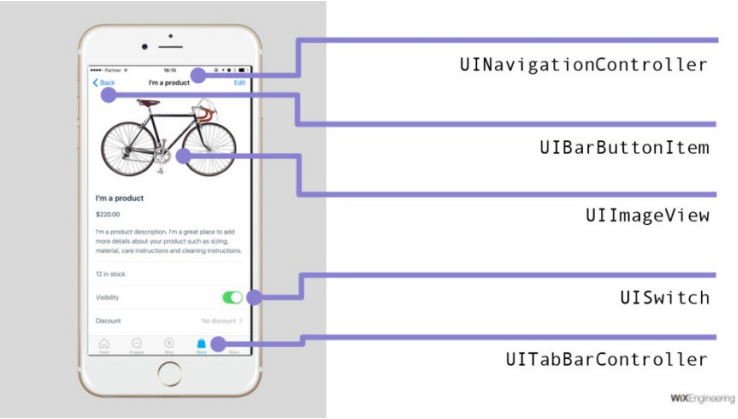

Figura V.1: Aplicație dezvoltată în React Native și generată pe un dispozitiv iOS [9]

V.1 Performanța oferită de React Native

React Native susține promisiuni puternice în ceea ce privește experiența de dezvoltare cu Javascript și reutilizarea codului între platforme. Principala întrebare este - aceste beneficii aduc un cost dramatic în ceea ce privește performanța? Cât de bine poate React Native să-și stăpânească propriile implementări native? Arhitectul în aplicații mobile de la Wix.com, Tal Kol, în cadrul unui tech talk pentru Wix Engineering Tech Talks, [8] , abordează acest subiect și discută despre cum este obținută această performanță în cadrul aplicațiilor dezvoltate în React Native și despre ce se află în spatele unei astfel de aplicații.

Pentru a înțelege limitările de performanță în React Native, este necesar să înțelegem ce se întâmplă în spatele acestui proces. Una dintre premisele principale în React Native este livrarea experienței native la care utilizatorii mobile se așteaptă, în timp ce focusul se axează pe eficiența dezvoltării în Javascript și React. Altfel spus, chiar dacă codul aplicației este scris în Javascript, interfața aplicației rezultate este complet nativă.

Acest fapt implică existența a două diferite în care aplicația rulează:

1. Domeniul nativ - Regatul pentru Objective-C/Swift în iOS și Java/Kotlin în Android. Aici se realizează interacțiunea cu sistemul de operare și View-urile sunt generate. Interfața este manipulată exclusiv pe thread-ul principal, însă pot exista și thread-uri în fundal care

să se ocupe de alte calcule. Aici React Native realizează cea mai grea parte pentru dezvoltatori.

2. Domeniul Javascript - Javascript este rulat pe propriul său thread, separat, de către un motor dedicat. Logica de business, de la ce View să fie afișat și până la stilizarea componentelor este, de regulă, implementată aici

Este de menționat faptul că variabilele definite într-un domeniul nu pot fi accesate în mod direct de către celălalt domeniu. Astfel, toate comunicările între cele două domenii trebuie realizate prin intermediul unui pod (bridge). Acest concept este asemănător cu modul în care clientul și serverul comunică în cadrul unei aplicații web - datele trebuie să fie serializate pentru a trece pe pod.

Aici se află una dintre cheile principale pentru înțelegerea performanței React Native. Fiecare domeniul în sine este foarte rapid. Blocajul în performanță apare atunci când trebuie să trecem de pe un domeniul în altul. Pentru a obține performanța cât mai ridicată în cadrul aplicației, numărul acestor treceri trebuie păstrate minimum.

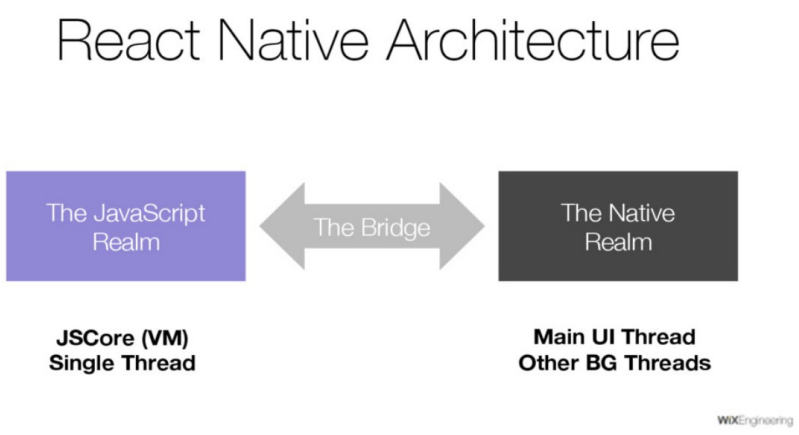

Figura V.2: *Cele două domenii ce stau la baza arhitecturii React Native și podul ce realizează legătura între acestea [9]*

VI. Prelucrarea imaginilor

Frumosul a reprezentat dintotdeauna un obiectiv pentru om în procesul de realizare al mediului ce îl înconjoară. Acesta se manisfestă prin căi personalizate pentru fiecare individ și determinate, în general, de mediul său înconjurător. Ca oameni, suntem ființe culturale, deci suntem influențați de societate. Luăm o parte din acea societate și o încorporăm în comportamentul nostru. Preferințele estetice sunt stabilite la nivel psihologic prin dezvoltare, expunere și inovație individuală. Suntem cu toții condiționați de cultură, în funcție de context. Astfel, de exemplu, modul de a privi o pictură diferă de la om la om, ba chiar de la aceeași persoană aflată în două momente diferite ale vieții.

Chit că vorbim de structura unui obiect, de materialul unei haine sau de culoarea sa, omul a urmărit întotdeauna ca bunurile să reflecte personalitatea lui sau să îi fie plăcute la vedere.

Odată cu evoluția tehnologiei, acest aspect a fost vizat și în imagini, iar astfel s-a construit un întreg subdomeniu al programării ce se ocupă de prelucrarea imaginilor.

Prelucrarea imaginilor este o metodă de a aplica operații asupra unei imagini pentru a îi modifica conținutul sau pentru a extragere caracteristici/ feature-uri referitoare la aceasta. Este un tip de procesare pe bază de semnal, în care input-ul este reprezentat de imagine, iar output-ul poate fi o altă imagine sau o caracteristică/ un feature.

Pentru a putea discuta despre prelucrarea imaginilor, este importantă înțelegerea conceptului de imagine la nivel digital.

Pentru început, ce este un pixel?

Un pixel este un punct din ecran, iar numărul de pixeli este dat de rezoluția ecranului. Fiecare pixel este compus din trei canale de culoare: canalul roșu, canalul verde și cel albastru (RGB - red, green, blue). Aceste canale sunt stocate pe 8 biți, prin urmare pot lua valori între 0 și 255, 0 reprezentând lipsa de culoare, iar 255 intensitatea maximă. Astfel, un pixel cu cele 3 canale 0-0-0 va avea culoarea negru, iar, la pol opus, 255-255-255 va fi echivalentul culorii alb. Pe același principiu, 255-0-0 va fi culoarea roșu, 255-255-0 galben și așa mai departe.

O imagine reprezintă o matrice NxM de pixeli, deci o structură NxMx3.

Există mai multe tipuri de imagini, din punct de vedere al canalelor de culoare. Un tip adesea întâlnit în practică îl reprezintă imaginile grayscale. Acestea sunt imagini ce conțin numai tonuri de gri. În cadrul unei astfel de imagini, pixelii nu mai sunt reprezentați prin cele 3 canale de culoare, ci prin unul singur ce ia valori între 0 și 255, adică de la negru la alb.

Un alt format de îl reprezintă cel al imaginilor alb-negru. Ele se diferențiază de cele grayscale prin faptul că un pixel poate fi alb sau negru, fără nuanțe de gri. Pixelii acestor imagini pot avea, prin urmare, doar două valori: 0 (negru) și 1 (alb).

Cele două tipuri de imagini prezentate anterior pot fi stocate și asemenea imaginilor color, însă această abordare este o modalitate ineficientă. Să spunem că avem o imagine complet albă de dimensiune 100x100 pixeli. Aceasta, salvată ca imagine în tonuri alb-negru, ar fi echivalentă unei matrici 100x100 biți. Aceeași imagine, dar salvată în format grayscale, va ocupa 100x100x8 biți, în timp ce varianta ei color 100x100x8x8x8 biți. Prin urmare, alegerea corectă a formatului aduce îmbunătățiri în stocarea memoriei, dar și rapiditate în aplicarea algoritmilor despre care vom vorbi în paginile ce urmează.

Fiecare filtru utilizat în aplicație este extras din biblioteca “expo-pixi”, despre care vom detalia în capitolul următor. Până atunci, vom enumera impactul fiecărui filtru asupra imaginilor; mai exact, cum se modifică intensitatea pixelilor unei matrici corespunzătoare unei imagini. Exemple interactive pot fi găsite cu ușurință în linkul atașat (https://pixijs.io/pixi-filters/tools/demo/).

În continuare, vom prezenta filtrele utilizate în contextul aplicației dezvoltate. Pentru evidențierea efectelor, s-a ales utilizarea fotografiei standard pentru procesarea imaginilor, Lena, utilizată în acest domeniu încă din 1973.

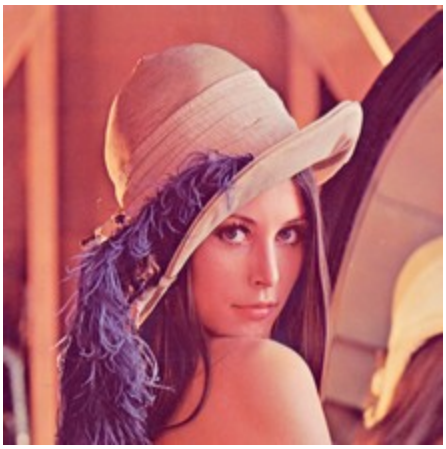

Figura VI.1: Lena, fotografia de testare standard pentru procesarea de imagini (Dwight Hooker, 1972)

VI.1 Proprietăți

VI.1.1 Contrast

Vizibilitatea componentelor scenei este, în general, determinată în cea mai mare parte de contrastul zonei din imagine; contrastul este o masură proporțională cu diferența dintre luminozitatea anumitor pixeli

Figura VI.2: Rezultatul obținut în urma creșterii contrastului

Modificarea contrastului presupune modificarea proporțională a intervalului valorilor pixelilor unei imagini. De exemplu, conform [4], o imagine ce are pixelii cu valori cuprinse între 40 și 90 poate avea contrastul crescut prin maparea proporțională a pixelilor în raza 0-255 (40 devine 0, 90 devine 255 și, analog 75 - mijlocul intervalului - va fi 127) [3]

VI.1.2 Luminozitate

Pentru a lumina conținutul unei imagini, este suficientă mărirea valorii fiecărui pixel, pe fiecare canal de culoare, cu aceeași constantă. Cu cât este setată o constantă mai mare, cu atât luminozitatea va fi mai ridicată, iar o valoare negativă va determina întunecarea imaginii. Trebuie luat în considerare faptul ca intensitatea unei pixel este cuprinsă între 0 și 255. Prin urmare, orice rezultat ce iese din acest interval va fi mapat la 0, respectiv 255.

Figura VI.3: Rezultatul obținut în urma creșterii luminozității

VI.1.3 Tonuri de gri

Un filtru adesea utilizat în practică și confundat cu așa-zisul filtru alb-negru este filtrul greyscale. Acesta oferă fotografiilor unor aspect vechi și scade impactul obositor pe care îl poate avea prezența prea multor detalii pe o fotografie

Așa cum am descris și la începutul capitolului, un pixel RGB devine un pixel greyscale, cu un singur canal de culoare de intensitate cuprinsă între 0 și 255 și cu valoarea egală cu media aritmetică a celor trei canale de culoare a imaginii originale.

Figura VI.4: Rezultatul obținut în urma aplicării filtrului greyscale

VI.1.4 Nuanță

Pentru nuanțarea conținutului unei imagini se utilizează filtrul Hue. Acesta primește ca input o culoare care va fi apoi evidențiată în imaginea rezultată. Spre exemplu, dacă este selectată culoarea roșie, pixelii imaginii vor avea intensitatea canalului roșu cu atât mai intensă cu cât roșul-input este mai puternic. În mod analog, dacă se dorește accentuarea unei culori ce este rezultatul combinării a două nuanțe (mov = roșu + albastru), va crește intensitatea pe canalele roșu și albastru a imaginii.

Figura VI.5: Rezultatul obținut în urma aplicării filtrului de nuanță verde

VI.1.5 Saturație

Efectul de saturare este asemănător celui Hue, operând asupra intensității pixelilor. O cale de a obține creșterea saturației este folosind extrapolarea imaginii existente cu versiunea ei în tonuri de gri. Procesul invers, de desaturare, se obține prin interpolare.

Figura VI.6: Rezultatul obținut în urma creșterii saturației

VI.2 Filtre

VI.2.1 Browni (Maroniu)

Este un subfiltru al celui de nuanță (Hue) ce oferă imaginii un aspect așa-zis ruginit. Culorile accentuate sunt pe canalele roșu și ușor verde, ambele de intensități scăzute astfel încât combinația lor să rezulte în maro (6:3:0)

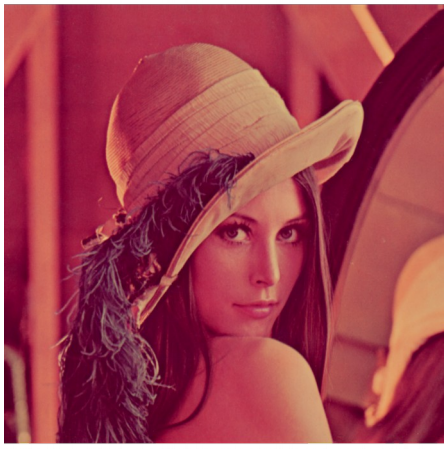

Figura VI.7: Rezultatul obținut în urma aplicării filtrului Browni

VI.2.2 Dots (Puncte)

Filtrul transformă imaginea primită în tonuri de alb și negru, după care definește efectu de “dots” (de maximum dimensiunea scale) prin punctarea conținutul alb cu puncte negre și punctarea conținutului negru cu puncte albe.Cu cât parametrul scale al funcției este mai scăzut, cu atât dimensiunea punctelor este mai scăzută, iar imaginea va avea o claritate mai mare

Figura VI.8: Rezultatul obținut în urma aplicării filtrului Dots

VI.2.3 Emboss (Gravare)

Acest filtru transpune imaginea în trei culori: alb, negru și gri pe baza unui algoritm:

- identifică muchiile din imagine și le marchează cu negru. În funcție de parametrul de intrare, strength, aceste muchii sunt mai groase sau mai subțiri

- tonurile aprinse (de intensitate ridicată) din zona muchiilor sunt evidențiate prin culoarea albă, urmând ca restul conținutului imaginii să fie gri.

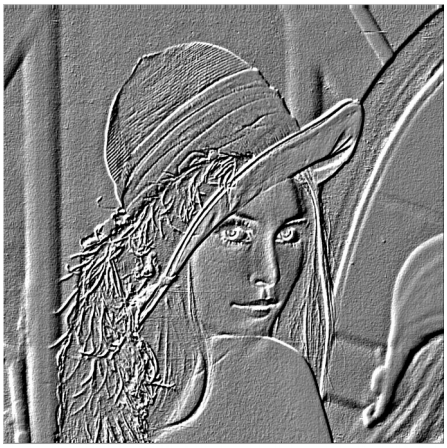

Figura VI.9: Rezultatul obținut în urma aplicării filtrului Emboss

VI.2.4 Pixelate (Pixelare)

Filtrul Pixelate oferă imaginii un efect pixelat, bazându-se pe un parametru, size, oferit funcției

Întreg conținutul imaginii se împarte în pătrate de latura size cărora le este înlocuit ulterior fiecare pixel cu valoarea dominantă din pătratul respectiv

Spre exemplu, să spunem că avem un pătrat de 3x3 cu valorile: roșu, roșu, gri, alb, roșu, roșu, negru, negru, negru. În această matrice, culoarea dominantă este roșu, prin urmare toți pixelii vor deveni roșii.

În cadrul acestui filtru, claritatea imaginii este invers proporțională cu dimensiunea parametrului size, iar valoarea acestuia egală cu 1 va oferi ca output imaginea inițială

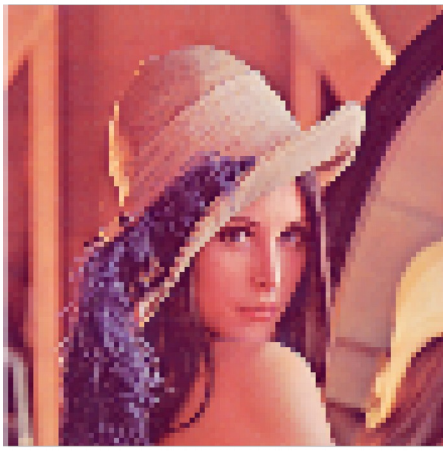

Figura VI.10: Rezultatul obținut în urma aplicării filtrului Pixelate

VI.2.5 Cross Hatch (Haşurare încrucișată)

Imită efectul de haşurare prin culorile alb și negru

- Sunt identificate muchiile din imagine, însă nu sunt evidențiate în modul în care sunt în cadrul efectului Emboss

- Din direcțiile muchiilor sunt trasate linii cu panta 1 sau -1 (+/-45 grade), a căror lungime și densitate este dată în funcție de intensitatea culorii pe care o înlocuiesc

Figura VI.11: Rezultatul obținut în urma aplicării filtrului Cross Hatch

VI.2.6 Noise (Zgomot)

Constă în modificarea intensității pixelilor unei imagini și adăugarea unui așa zis efect de zgomot

- Primește ca parametru un număr între 0 și 1, corespondenți ai procentului de 0%, respectiv 100% (analog, 0.5 va corespunde valorii de 50%)]
- Acest procent reprezintă numărul de pixeli care vor fi modificați în cadrul aplicării filtrului
- Pentru fiecare pixel selectat, se alege random între -127 și 127, număr ce va fi ulterior adăugat pe fiecare canal de culoare al pixelului
- Spre exemplu, dacă pixelul selectat are valorile (127, 200, 227), iar valoarea random este -127, pixelul va avea valoarea rezultată (0, 74, 100)

Figura VI.12: Rezultatul obținut în urma aplicării filtrului Noise

VI.2.7 Old Film (Film vechi)

Are rolul de a oferi imaginii un aspect vintage. În cadrul acestui filtru, se realizează o serie de de operații, astfel :

- Imaginea este convertită în tonuri sepia
- Se aplică filtrul Vignette, pentru întunecarea graduală a imaginii, valoarea maximă fiind în colțuri, iar minimă în centru
- Se adaugă puncte și linii aleatoare albe cu rolul de a sugera deteriorarea fotografiei, asemenea unei imagini vechi

Figura VI.13: Rezultatul obținut în urma aplicării filtrului Old Film

VI.2.8 RGB Split (Împărțire RGB)

- Realizează împrăștierea fiecărui pixel prin fiecare canal de culoare, primind ca parametri 3 perechi de numere ce vor reprezenta coordonatele fiecărui canal

Figura VI.14: Rezultatul obținut în urma aplicării filtrului RGB Split

VI.2.9 Bulge Pinch (Umflătură-ciupitură)

Având centrul și raza reglabile, filtrul bulge pinch realizează pe cercul determinat o evidențiere a conținutului prin aplicarea unui zoom gaussian. Astfel, cu cât pixelii sunt mai apropiați de centrul cercului, cu atât mai mult vor fi măriți. În mod analog, îndepărtarea spre capetele cercului setat mărește tot mai puțin conținutul, pe marginea cercului fiind valoarea inițială, asemenea exteriorului cercului.

Figura VI.15: Rezultatul obținut în urma aplicării filtrului Bulge Pinch

VI.2.10 Motion Blur (Neclaritate de mișcare)

Oferă imaginii aspectul unei fotografii realizate în mișcare, asemenea efectului obținut atunci când se produce o mișcare rapidă în timpul fotografierii sau timpul de expunere al aparatului este mult prea ridicat.

Pentru obținerea acestui aspect la nivel de procesare, se procedează imaginea prin intermediul operației de translatare gaussiană (în cazul nostru, pe diagonala secundară), în ambele sensuri (sus și jos).

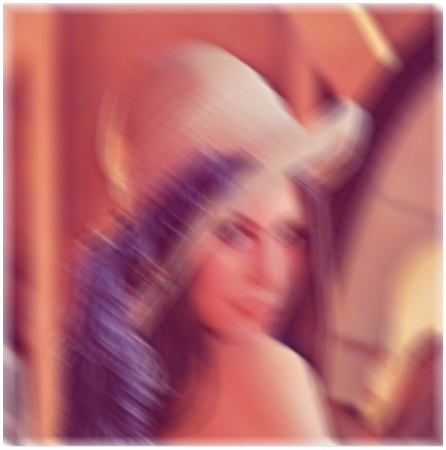

Figura VI.16: Rezultatul obținut în urma aplicării filtrului Motion Blur

VI.2.11 Advanced Bloom (Înflorire avansată)

Este un filtru format prin combinația filtrelor de contrast și luminozitate și are rolul de a evidenția elementele imaginii.

Figura VI.17: Rezultatul obținut în urma aplicării filtrului Advanced Bloom

VI.2.12 Blur (Estompare)

Are rolul de a realiza contururi mai vagi în imagine, oferind un efect neclar conținutului. Pentru fiecare pixel se calculează media aritmetică a sa cu a celor 9 vecini, iar rezultatul reprezintă noua sa valoare.

Figura VI.18: Rezultatul obținut în urma aplicării filtrului Blur

VI.2.13 Negative (Negativ)

Transformă fiecare pixel al unei fotografii în corespunzător său complementar.

Figura VI.19: Exemple de culori și perechile lor complementare [14]

Culoarea dominantă în fotografia Lena este roșu, prin urmare rezultatul obținut în urma aplicării filtrului Negative peste imagine trebuie să aibă culoarea dominantă albastru deschis.

Figura VI.20: Rezultatul obținut în urma aplicării filtrului Negative

VI.2.14 Night (Noapte)

Presupune eliminarea tonurilor de roșu și verde din imagine.

Figura VI.21: Rezultatul obținut în urma aplicării filtrului cu efect de noapte

VI.3 Studiu dezvoltat pe baza filtrului Sepia

Fiecare dintre filtrele prezentate este descris printr-o matrice de pixeli aplicată pe fotografia curentă pentru a obține rezultatul vizualizat. În continuare, vom studia efectele pe care îl au diferite matrici asupra fotografiei.

Vom lucra cu matrici de dimensiuni 4x5, astfel încât:

- prima linia corespunde canalului roșu
- a doua linie corespunde canalului verde
- a treia linia este dedicată canalului albastru
- ultima linia, a patra, este pentru canalul alpha, cel de opacitate.

Primele trei coloane reprezintă coeficienții de înmulțire cu cele trei canale de culoare ale pixelului, iar cele două coloane din urmă sunt dedicate opacității.

Matricea identitate, cea care nu aplică nicio modificare asupra fotografiei, este:

1, 0, 0, 0, 0

0, 1, 0, 0, 0

0, 0, 1, 0, 0

0, 0, 0, 1, 0

Vom lua mai departe spre analiză filtrul sepia, a cărui matrice de transformare clasic utilizată este:

0.39 0.769 0.189 0 0

0.349 0.686 0.168 0 0

0.272 0.534 0.131 0 0

0 0 0 1 0

Un pixel asupra căruia se aplică această transformare se modifică astfel:

Presupunem pixel P(100,150,200)

tr = 100 * 0.39 + 150 * 0.769 + 200 * 0.189 = 192.45 = 192

Analog,

tg = 100 * 0.349 + 150 * 0.686 + 200 * 0.168 = 171

tb = 100 * 0.272 + 150 * 0.534 + 200 * 0.131 = 133

Pixelul devine P(192, 171, 133)

Observăm cum canalul roșu a crescut semnificativ, canalul verde a crescut ușor, iar valoarea canalului albastru a scăzut. Acest rezultat este cauzat de faptul că suma pe liniile matricei este 1.348, 1.2, respectiv 0.937 și depinde dacă numărul este supraunitar sau subunitar. Scopul filtrului sepia este de a reducere tonurile de albastru din imagine și de a avea dominant tonuri calde.

În continuare, vom analiza ce transformări putem aplica asupra fotografiei astfel încât să obținem efectul sepia alături de luminozități diferite. Am luat drept referință fotografia din figura VI.22.

Figura VI.22: Turnul din Pisa

http://mikestravelguide.com/wp-content/uploads/2012/11/Pisa-tower-2-800x1066.jpg

Din filtrul sepia original, observal proportiile aproximative R:G:B ~ 2:4:1 ce se asigură de păstrarea tonurilor calde pe care efectul sepia le realizează. În elaborarea filtrelor particularizate, vom avea în vedere această proporție pentru a menține un aspect uniform de intensitate.

Nuanța sepia poate fi definită prin diferite tonuri calde. Printre acestea, se numără culoarea rgb(112,66,20), maro. De asemenea, fiind o nuanță întunecată, luminozitatea fotografiei va scădea.

Pentru a realiza un filtru sepia a cărui ton dominant să fie acesta, am dezvoltat algoritmul următor:

- Se mapează valorile canalelor prin regula de trei-simplu, având drept referință 127 la 1
- Fiecare dintre cele trei valori obținute se distribuie după regula 2:4:1

Având culoarea rgb(112, 66, 20), în urma aplicării primul pas obținem valorile (0.88, 0.07, 0.02), iar, după distribuirea pe coloane a valorilor, descrisă în pasul al doilea, matricea de transformare va deveni:

0.25 0.5 0.125 0 0

0.15 0.3 0.07 0 0

0.04 0.06 0.02 0 0

0 0 0 1 0

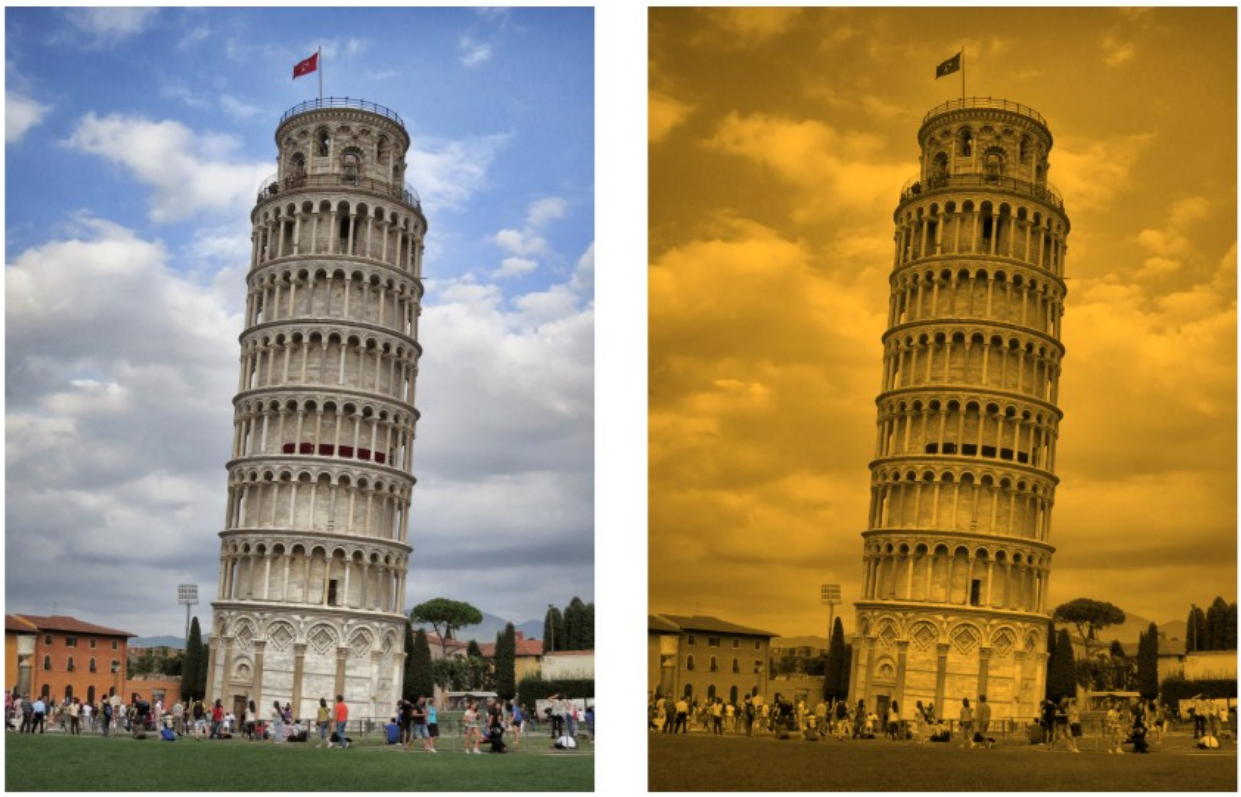

Figura VI.23: Rezultatul obținut în urma aplicării filtrului particularizat în tonuri maronii Sepia

O altă nuanță aparținând familiei sepia este roșu-caramiziu, descris prin rgb(165, 42, 42).

Prin regula de trei-simplu, obținem valorile (1.3, 0.33, 033), iar distribuind pe coloane, rezultă matricea de transformare:

0.36 0.72 0.18 0 0

0.08 0.16 0.04 0 0

0.08 0.16 0.04 0 0

0 0 0 1 0

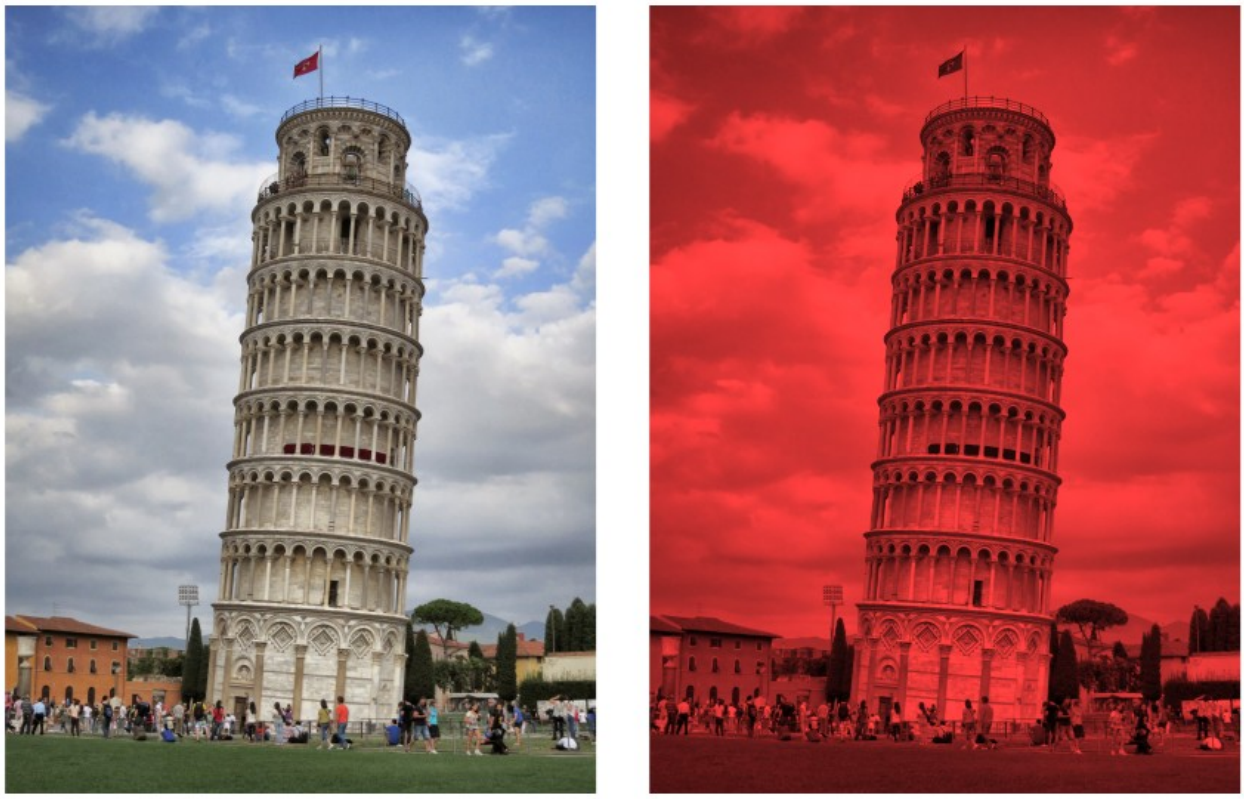

Figura VI.24: Rezultatul obținut în urma aplicării filtrului particularizat în tonuri roșii Sepia

A treia culoare pe baza căruia am aplicat algoritmul este rgb(230, 168, 23), auriu.

În urma mapării, am obținut valorile (1.81, 1.32, 0.18),

iar după distribuirea pe coloane, a rezultat matricea:

0.5 1 0.25 0 0

0.38 0.76 0.19 0 0

0.05 0.1 0.025 0 0

0 0 0 1 0

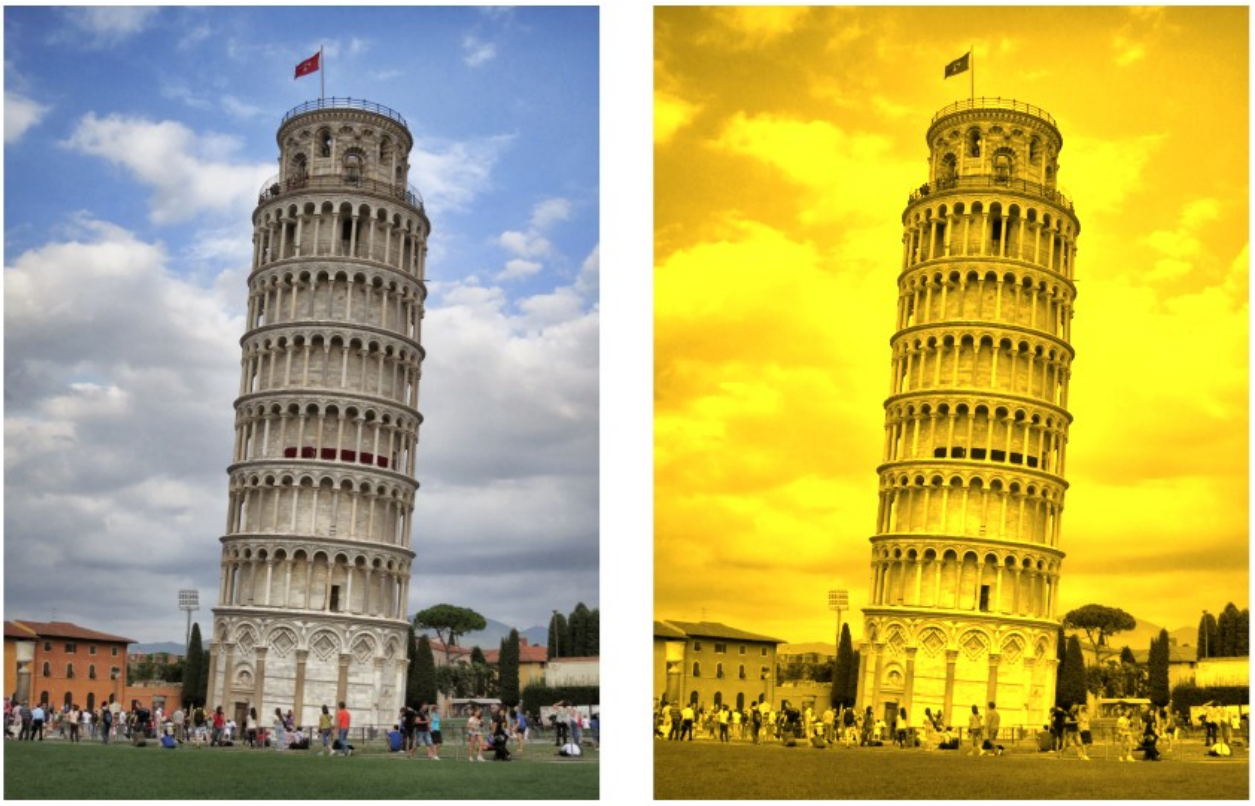

Figura VI.25: Rezultatul obținut în urma aplicării filtrului particularizat în tonuri aurii Sepia

În concluzie, astfel de filtre se pot realiza plecând de la orice nuanță pe care dorim să o oferim fotografiei. Algorimul dezvoltat asigură păstrarea unor proporții de luminozitate, astfel încât să nu apară anomalii în conținutul fotografiei. Dacă se dorește, totuși, creșterea luminozității în fotografie, se poate modifica maparea lui 1 într-o valoare mai mare decât 127. Analog, pentru scăderea luminozității, se poate opta pentru maparea spre o valoare mai mică decât 127.

VII. Soluția software dezvoltată

Pentru evidențierea conceptelor prezentate, a urmat construirea unei soluții software care să le înglobeze. Aplicația mobile dezvoltată, Smart Pixel, realizează editarea de imagini prin prelucrarea pixelilor și aplicarea filtrelor enumerate. Scopul acestei soluții este înfrumusețarea și/ sau modificarea aspectului unei fotografii digitale. Exemple de astfel de aplicații se regăsesc adesea pe piață, printre ele numărându-se VSCO Camera, PicsArt, Snapseed etc. Cu toate că numărul lor este destul de mare, cererea pe piață continuă să crească. Factorul determinant al acestei creșteri îl reprezintă expansiunea rețelelor de socializare și influenței lor asupra omului.

Ceea ce deosebește aplicația Smart Pixel de restul aplicațiilor aflate deja pe piață este un ecran dedicat creării propriilor filtre de către utilizator. În cadrul acestuia, este afișată o matrice 4x5 de dezvoltare, având valoarea inițială egală cu matricea neutră. Utilizatorul are posibilitatea de a modifica valorile acesteia și să observe în timp real rezultatul transformării. De asemenea, sunt prezentate instrucțiuni sumare de scriere a matricei de dezvoltare, astfel încât utilizatorul să poată realiza o transformare cât mai apropiată de ceea ce rezultatul dorit. Rolul acestui ecran poate fi considerat atât cu rol educativ, pentru a înțelege efectele pe care diferiți parametri îi au asupra unei transformări, cât și pentru a oferi utilizatorului o gamă cât mai largă de filtre, având caracter unic și personalizat.

În procesul de hotărâre a tehnologiei utilizate, un prim aspect a constat în analiza dependințelor externe pe care aplicația le-ar utiliza. Principala funcționalitate este, în mod evident, aplicarea de filtre asupra imaginilor. Ca soluție pentru aplicațiile realizate în Javascript s-a utilizat PixiJS, disponibil atât pentru React Native, cât și pentru Expo. Cum nu a fost necesară integrarea de funcționalități specifice unei anumite platforme, iar aspecte precum memoria cache, camera foto, galeria cu imagini sunt include în core-ul oricărui framework de dezvoltare de aplicații hibride,

această soluție este una realizabilă. Totodată, s-a ținut cont de avantajul principal pe care îl oferă dezvoltarea unei aplicații hibrid: disponibilitatea pe mai multe platforme mobile.

Dintre framework-urile de dezvoltare, s-a ales React Native datorită stabilității sale, fiind o platformă open-source și cu suport constant. De asemenea, documentația este una foarte bine elaborată, iar dezvoltarea este una intuitivă, odată venit din mediul web. Tot datorită popularității sale, numărul de pachete de pe node manager compatibile este unul foarte mare.

Navigând și mai adânc în mediul React Native, a apărut posibilitatea folosirii platformei Expo. Aceasta vine, la randul ei, cu limitări în privința dezvoltării, dar cu avantajul propriului SDK (Software Development Kit). Practic, Expo se ocupă de o mare parte din complexitatea build-ului unei aplicații mobile. Acest aspect este regăsit în documentație sub titlul de Managed Workflow și asigură dezvoltatorului că tot codul scris va fi în mediul Javascript, nefiind necesară intervenția sa în partea de iOS sau Android, caz întâlnit în situații particulare când dezvoltarea este realizată prin React Native.

De asemenea, prin API-uri specifice, Expo permite accesul aplicației la resursele device-ului , cum ar fi camera foto, galeria media, detectarea de gesturi etc. Aceste API-uri sunt updatate constant, fiind o platformă actuală cu suport tehnic la zi. Astfel, dependințele dintre pachete nu vor avea de suferit pe parcursul timpului și nu este necesară actualizarea constantă a aplicației de către dezvoltator.

Tool-ul pe care Expo îl oferă dezvoltatorilor pentru rularea aplicației în timpul dezvoltării este Expo Client. Acesta generează un URL de pe care se poate realiza debugging. Există posibilitatea de a rula aplicația pe un emulator, pe un device real sau chiar și în browser. Cele două din urmă sunt disponibile exclusiv datorită platformei Expo și reprezintă modalitatea în care s-a realizat dezvoltarea și debugging-ul aplicației.

VII.1 Dependințe Node Packege Manager (NPM)

Deoarece React și React Native nu sunt framework-uri web, respectiv mobile, ci doar librării care se ocupă de generarea conținutul paginii, pentru construirea unei aplicații complete este necesară adăugarea de librării care să satisfacă celelalte nevoi.

Instalarea oricărui pachet se realizează prin deschiderea unui terminal cu locația curentă în folder-ul proiectului și introducerea comenzii "npm install" urmată de numele pachetului și, opțional, versiunea sa.

VII.1.1 Navigarea

Navigarea permite unui utilizator să schimbe conținutul vizualizat în aplicație prin configurarea de rute, fiecare specifică unei anumite componente sau a unui container.

Pentru realizarea rutării este utilizat pachetul react-navigation.

Comanda utilizată pentru instalarea pachetului:

npm install react-navigation

Acesta conține componente React și funcții ce ajută la configurarea rutării. Aplicația Smart Pixel utilizează funcția createBottomTabNavigator() ce returnează un obiect care poate fi configurat astfel încât să se construiască un meniu de tipul bottom tab, specific aplicațiilor mobile.

Fiecare Tab Screen face referire către un Stack Navigator care conține două componente, cea de editare a unei imagini și una specifică stack-ului.

```
function GalleryStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen  name={Routes.GALLERY_MAIN} component={Gallery} />
      <Stack.Screen name={Routes.GALLERY_EDIT} component={EditPhoto} />
    </Stack.Navigator>
  );
}
```

Figura VII.1: Funcția care returnează stiva construită pentru galeria foto a aplicației

Construcția astfel a rutării este un aspect caracteristic aplicațiilor mobile și utilizează conceptul de stivă, cu operațiile specifice de pop și push. Deoarece din oricare dintre cele trei din cele patru ecrane din Bottom Tab Navigator se poate ajunge în componenta de editare de fotografii, s-a creat câte un Stack fiecărei astfel de componente care să conțină și ecranul de editare.

Totodată, fiecare componentă primește intern un obiect navigation cu ajutorul căruia se poate realiza rutarea între ecrane. Rutarea este permisă doar între componente de același nivel.

În exemplul de mai sus este redat modul în care a fost utilizat acest obiect.

```
async function redirectToEditPhoto() {
    try {
        AssetUtils.fromUriAsync(image.uri).then(fromUri => {
            fromUri.localUri = fromUri.uri;
            AssetUtils.resolveAsync(fromUri).then(uriResolved => {
                navigation.navigate(Routes.NEW_EDIT, {photo: uriResolved})
            });
        });
    } catch (error) {
        console.log(error);
    }
}
```

Figura VII.2: Funcția de redirectare din ecranul Home în ecranul Edit Photo, prin pasarea imaginii ce urmează a fi editată

Metoda navigate, pe lângă executarea de navigare, permite și trimiterea de parametri între componente. În acest caz, s-a trimis, din componenta de galerie către cea de editare, fotografia selectată pentru a fi ulterior prelucrată.

Construirea unui meniu de tipul Tab Bar este o soluție tot mai populară în rândul metodelor de navigare în aplicațiile mobil. O bună navigare trebuie să înștiințeze utilizatorul constant două lucruri: unde se află și unde poate ajunge. De asemenea, posibilitatea de deplasare de la o pagină la alta se realizează într-un singur pas, asigurând o experiență plăcută la nivel de utilizare. Un alt avantaj al acestui meniu este dat de poziția sa pe ecran. Deoarece telefoanele mobile tind să aibă dimensiuni tot mai mari, cea mai mare parte a interacțiunii utilizatorului cu ecranul trebuie să se realizeze în jumătatea inferioară a display-ului, această zonă fiind mult mai ușor accesibilă.

Această soluție ar prezenta o problemă în cazul în care am avea un număr mai mare de 5 ecrane principale, deoarece nu pot fi așezate într-o singură linie păstrând vizibilitatea și lizibilitatea. Într-o astfel de situație, o soluție optimă ar putea fi meniul de tip Drawer.

VII.1.2 Importarea de fișiere din telefon

Printre specificațiile aplicației, se numără și importarea de imagini din galeria telefonului sau direct din camera foto. Pentru a avea acces la aceste fișiere, dar și la metode de manipulare a lor, s-a instalat pachetul “expo-image-picker”.

Comanda utililzată pentru instalarea pachetului:

npm install expo-image-picker

Vom aborda, pentru început, cazul când se dorește importarea de fișiere din Camera Roll. Pentru a avea acces la acestea, este nevoie de oferirea de acces din partea utilizatorului. Astfel, în primul rând este cerut accesul la galerie, urmând apoi deschiderea propriu-zisă a bibliotecii foto.

```
async function pickImageFromGallery() {
    try {
        let permission = await ImagePicker.getCameraRollPermissionsAsync();

        if (permission.granted == false) {
            ImagePicker.requestCameraRollPermissionsAsync();
            permission = await ImagePicker.getCameraRollPermissionsAsync();
        }

        if (permission.granted == true) {
            let result = await ImagePicker.launchImageLibraryAsync( options: {
                mediaTypes: ImagePicker.MediaTypeOptions.Images,
                allowsEditing: true,
                quality: 1
            });

            if (!result.cancelled) {
                setImage(result);
                // @ts-ignore
                storeDataToStorage(result.uri);
                setShowOptions( value: false);
            }
        }
    } catch (e) {
        console.log(e)
    }
}
```

Figura VII.3: Funcția utilizată pentru a accesa galeria foto a dispozitivului

În figura VII.3, implementarea este realizată astfel: se verifică dacă aplicația are acces la galeria foto, iar, dacă nu, este solicitat acest lucru. După ce accesul este permis, se deschide galeria foto prin funcția launchImageLibraryAsync.

Prin parametrii funcţiei se specifică tipul fișierelor media care să fie încărcate, permiterea editării prin decupare și rotație, precum și calitatea fotografiei încărcate, 1 fiind maximum.

Încărcarea unei imagini direct din camera foto este un proces asemănător celui descris anterior.

```
async function pickImageFromCamera() {
    try {
        let permission = await ImagePicker.getCameraPermissionsAsync();

        if (permission.granted == false) {
            ImagePicker.requestCameraPermissionsAsync();
            permission = await ImagePicker.getCameraPermissionsAsync();
        }

        if (permission.granted == true) {
            let result = await ImagePicker.launchCameraAsync( options: {
                mediaTypes: ImagePicker.MediaTypeOptions.Images,
                allowsEditing: true,
                quality: 1
            });

            if (!result.cancelled) {
                setImage(result);
                // @ts-ignore
                storeDataToStorage(result.uri);
                setShowOptions( value: false);
            }
        }
    } catch (e) {
        console.log(e)
    }
}
```

Figura VII.4: Funcția utilizată pentru a accesa camera foto a dispozitivului

Diferența o reprezintă faptul că se cere accesul la camera foto printr-o funcție dedicată acestui scop, iar în locul deschiderii galeriei, se utilizează funcția launchCameraAsync ce va porni camera foto a dispozitivului mobil, cu aceeași parametri definiți și pentru încărcarea din galeria foto.

Obiectul rezultat la încheierea ambelor acțiuni este de același tip și conține un câmp, cancelled, care este fals atunci când o imagine a fost încărcată. Dacă operația de selectare și încărcare s-a încheiat cu succes, imaginea este salvată în storage-ul intern al aplicației și într-o variabilă pentru a fi afișată pe ecran.

VII.1.3 Stocarea datelor aplicației în memoria dispozitivului

Async Storage este o alternativă pentru Local Storage, ce stochează datele asincron și persistent în forma cheie-valoare, fără a le cripta. Acest obiect se află în biblioteca react-native, nefiind necesară adăugarea unei noi dependințe npm. Valoarea stocată în orice cheie din Async Storage este în mod exclusiv de tipul string, prin urmare orice obiect trebuie parsat înainte de a fi stocat. Pentru acest procedeu, se utilizează metoda JSON.stringify(), iar atunci când se va extrage fișierul din cache, i se va aplica metoda inversă, JSON.parse(), care convertește un string în obiect.

În aplicația creată, există două chei dedicate stocării imaginilor. Deoarece fiecare fotografie încărcată de pe Camera Roll sau direct de pe camera foto se dorește a fi păstrată într-o galerie internă a aplicației, s-a construit cheia “gallery”, iar pentru imaginile editate și prezentate în ecranul Edited Gallery, există cheia “edited-gallery”.

```
export enum StorageTypes {
    'GALLERY' = 'gallery',
    'EDITED_PHOTOS' = 'edited-gallery',
}
```

Figura VII.5 Cheile utilizate pentru stocarea imaginilor

În fiecare dintre aceste două chei este stocat un array ce conține căile imaginilor salvate în memoria cache.

Dată fiind constrângerea dată de tipul valorii stocate, un array este de fapt un string care are fiecare cale către imagini separată prin virgulă. Pentru a stoca o nouă imagine în cheia ‘gallery’, se extrage mai întâi vechiul conținut, i se concateneaza o virgulă, iar apoi se adaugă noua imagine convertită ca string. Tot acest nou șir de caractere înlocuiește apoi vechea valoare aflată pe cheia gallery.

```
async function storeDataToStorage(image) {
    try {
        let storedValue = await AsyncStorage.getItem(StorageTypes.EDITED_PHOTOS);
        const newImage = JSON.stringify(image);
        let result = storedValue ? storedValue.concat(",").concat(newImage) : newImage;
        await AsyncStorage.setItem(StorageTypes.EDITED_PHOTOS, result);

    } catch (error) {
        console.log(error);
    }
}
```

Figura VII.6 Funcția de stocare a unei imagini în șirul existent prin conversia în string

Pentru a obține galeria propriu-zisă, se extrage valoarea din Async Storage stocată pe cheia gallery și se prelucrează astfel: se split-uiește valoarea cheii la fiecare virgulă întâlnită prin construirea un array care conține valorile dintre virgule. Aceste valori sunt apoi parsate în obiect și astfel se obține un șir de obiecte de căi ale fotografiilor stocate în memoria cache, care pot fi afișate în componenta Gallery.

```
async function getDataFromStorage() {
    try {
        var value = await AsyncStorage.getItem(StorageTypes.GALLERY);
        if (value !== null) {
            let newValue = value.split( separator: ",").map( callbackfn: item => JSON.parse(item))
                .filter( callbackfn: item => item !== null);
            setImages(newValue);
        }
    } catch (error) {
        console.log(error);
    }
}
```

Figura VII.7 Funcția care extrage imaginile de la o cheie și le convertește în șir

VII.1.4 Aplicarea filtrelor asupra imaginilor

Pentru filtrarea imaginilor s-a adăugat un pachet adițional, “expo-pixi”, care oferă un set de filtre și de funcții dedicate prelucrării imaginilor. Acest pachet este construit pe baza bibliotecii PixiJS și a fost adaptat astfel încât să poată fi utilizat pe platforma Expo.

PixiJS este o bibliotecă ce se ocupă de generarea conținutului, permițând dezvoltarea de imagini grafice interactive, bogate în efecte, utilizate ulterior în jocuri sau aplicații. PixiJS este un wrapper peste WebGL. WebGL este o bibliotecă pentru Javascript de generare a conținutul grafic

și are acces la GPU (Graphics Processing Unit), permițând realizarea operațiilor și randării conținutului foarte rapid. PixiJS dispune de suport WebGL și se încadrează perfect în canvas-ul oferit de HTML5.

Printre avantajele oferite de PixiJS și care au afectat aplicația, se numără:

- Suport pe o plajă largă de platforme conținut interactiv disponibil pentru desktop, mobil și chiar mai mult, printr-o singură bază a codului. Acest lucru se datorează scrierii sale în Javascript, limbaj universal pentru orice domeniu
- Ușurință în utilizarea API-urilor
- Detectarea randării conținutului

Comanda utililzată pentru instalarea pachetului:

npm install expo-pixi

Componenta pe care “expo-pixi” o oferă este Filter Image. Aceasta extinde funcționalitatea de afișare de imagini a componentei Image și vine în plus cu parametrul “filters”. “Filters” este un șir de obiecte de tipul ColorMatrixFilter. Fiecare astfel de obiect este echivalentul matricei de intensități care este aplicată în compunerea imaginii filtrate. Astfel, pot fi adăugate mai multe filtre care să prelucreze imaginea, iar rezultatul acestei compuneri va fi generat în timp real pe ecran.

La nivel de aplicație, au fost declarate două tipuri de filtre: basic și custom. Filtrele basic sunt cele ce țin de proprietățile fotografiei: saturație, contrast, luminozitate etc., iar cele custom sunt efectele găsite în aplicație sub denumirea de Filters.

```
<ExpoPixi.FilterImage
    source={photo}
    ref={(c) => setFilterImageRef(c)}
    style={styles.image}
    resizeMode={"cover"}
    filters={filter}
/>
```

Figura VII.8: Modul de utilizare al componentei Filter Image

VII.2 Structura aplicației

Fiind scrisă în React Native, aplicația este structurată în componente, fiecare dedicată unui ecran. Astfel, avem 4 componente cu rol de ecran și componenta rădăcină a aplicației. Arhitectura aplicației, ilustrată în figura VII.9, este formată din componentele ecran și este dictată de modul în care a fost configurat React Navigation.

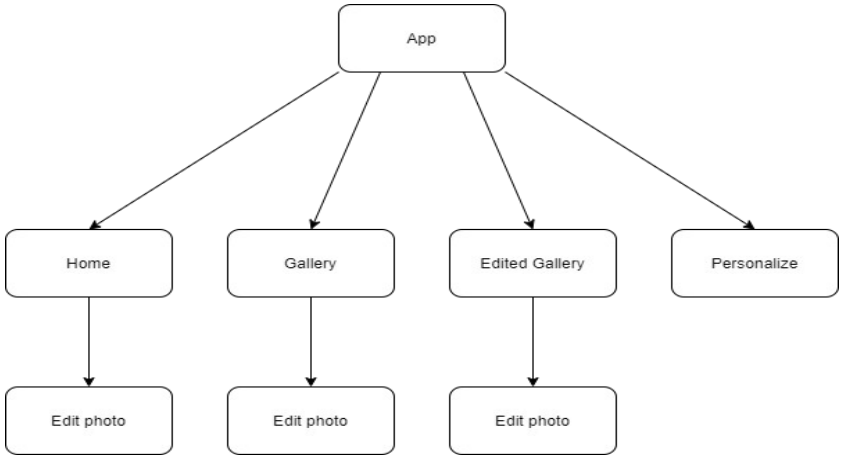

Figura VII.9: Arhitectura sub formă de arbore a aplicației dezvoltate

VII.2.1 App

- componentă esențială fiecărei aplicații React Native. Este locul unde se realizează încărcarea aplicației și se configurează setările

- aici, sunt declarate stivele de rutare și este afișată componenta de navigare împreuna cu tema ei particularizată
- este declarată componenta al cărei conținut să fie afișat după încărcarea aplicației: Home

VII.2.2 Home

- din ecranul acesta, utilizatorul poate încărca o fotografie nouă în galeria sa cu imagini ce urmează a fi editate și poate naviga spre ecranul de editare a unei fotografii cu noua poză adăugată
- există două modalități de încărcare a unei fotografii în aplicației: accesând galeria foto a telefonului sau prin realizarea unei fotografii din interiorul aplicației

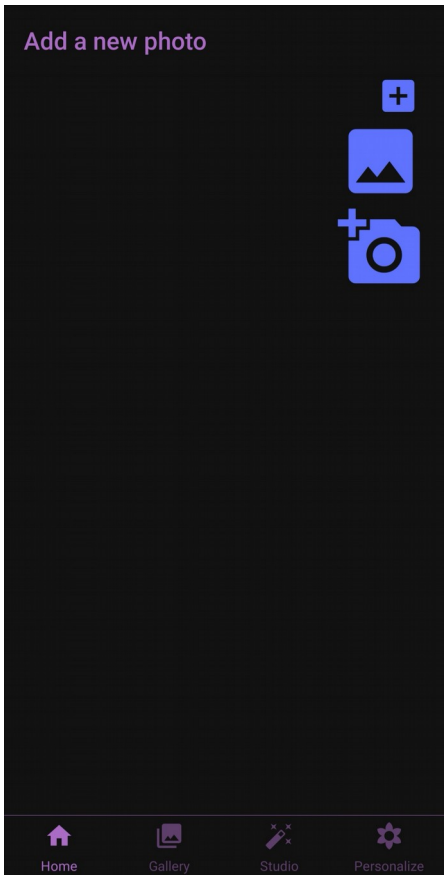

Figura VII.10: Ecranul unde se realizează importarea fotografiilor

VII.2.3 Gallery

- acesta este ecranul unde sunt afișate fotografiile care au fost încărcate în cadrul ecranului Home și stocate în memoria cache
- odată adăugată o nouă fotografie, aceasta va persista în memoria aplicației și va fi afișată la fiecare sesiune de utilizare a aplicației
- este posibilă ștergerea fotografiilor

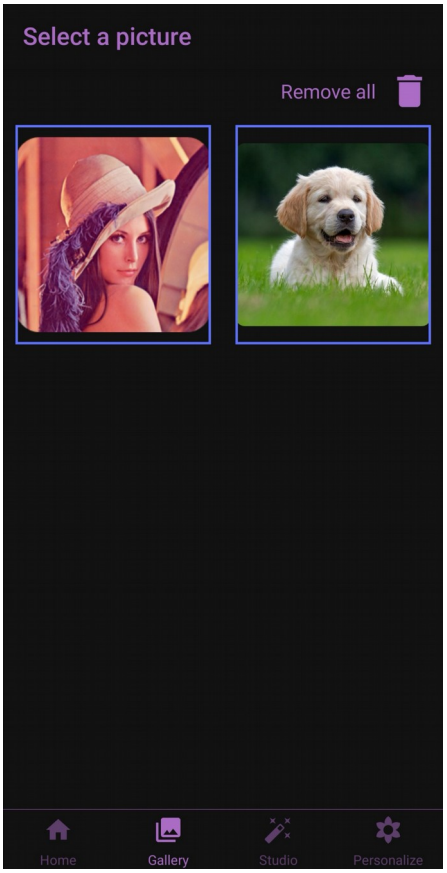

Figura VII.11: Galeria fotografiilor importate

VII.2.4 Edited Gallery

- asemenea ecranului menționat anterior, această componentă se ocupă de afișarea unei galerii de imagini
- fiecare fotografie rezultată în urma operației de editare este afișată aici

- utilizatorul are posibilitatea de a distribui o fotografie, direct din aplicație, pe platformele de socializare disponibile pe dispozitivul său mobil
- asemenea ecranului Gallery, fotografiile pot fi șterse și se poate opta pentru reeditarea unei fotografii, creându-se o nouă instanță a sa, fără a fi suprascrisă

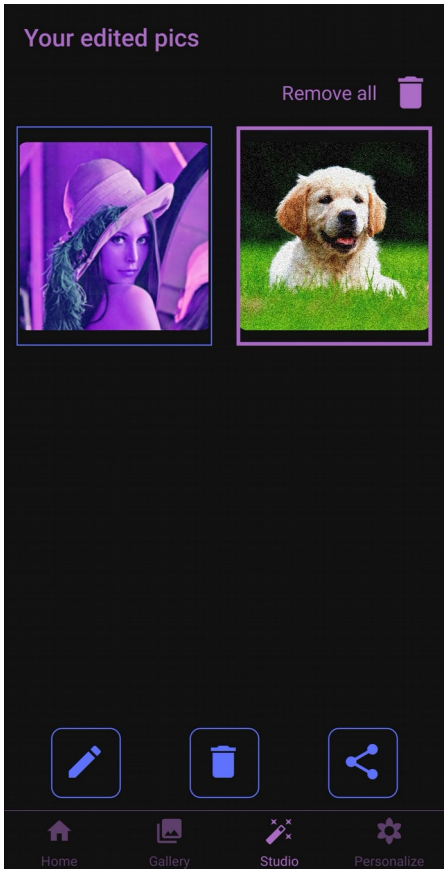

Figura VII.12: Galeria fotografiilor editate

VII.2.5 Edit Photo

- reprezintă ecranul în care se realizează editarea fotografiilor.
- primește de fiecare dată o variabilă ce conține calea imaginii pe care urmează să o prelucreze

- fotografiei îi sunt aplicate în timp real filtrele și proprietățile alese de utilizator, cu posibilitatea setării intensității lor, atunci când este cazul
- fiecare filtru poate fi eliminat individual prin butonul din dreptul barei de intensitate.
- la încheierea procesului de editare, se poate opta pentru salvarea fotografiei în galeria telefonului și a aplicației

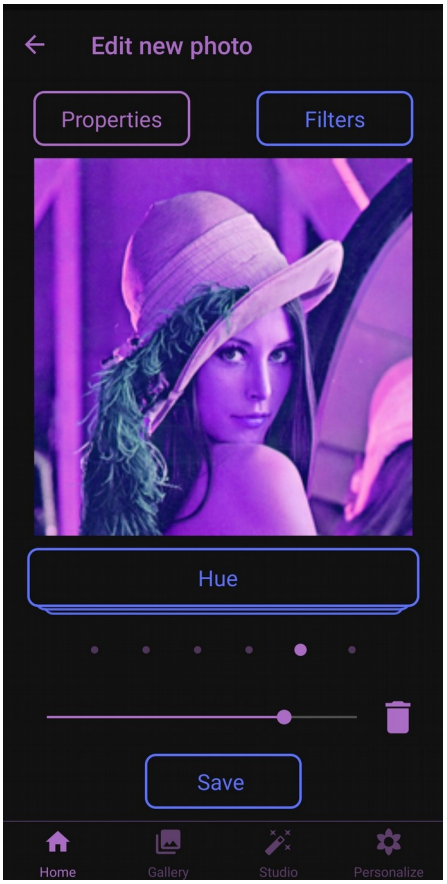

Figura VII.13: Ecranul de editare a unei fotografii

VII.2.6 Personalize

- ecranul în care utilizatorul are posibilitatea de a-și crea propriile filtre personalizate
- sunt afișate reguli și explicații sumare cu privire la modul în care acționează matricea de transformare asupra unei fotografii

- utilizatorul poate completa cu diferite valori matricea și să vizualizeze rezultate în timp real
- în figura VII.14 au fost introduse valorile descrise în cadrul filtrului auriu elaborat în cadrul capitolului VI.3 (Golden Sepia)

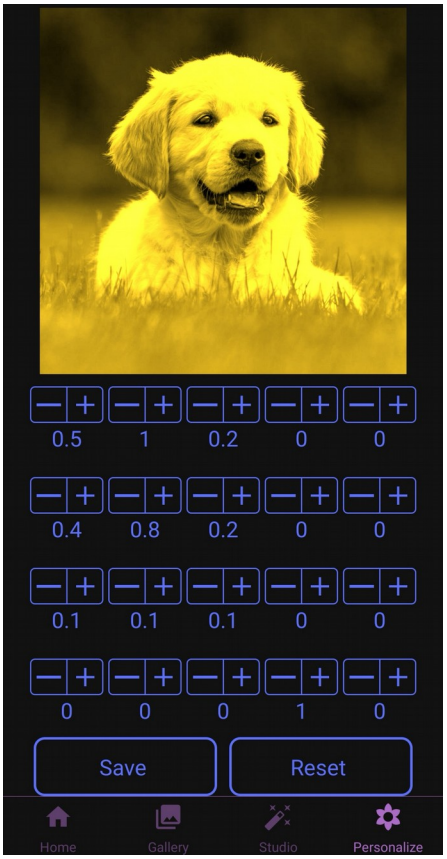

Figura VII.14: Ecranul dedicat customizării matricei unui filtru

VII.3 Alte aspecte ale aplicației

Pe lângă componentele specifice fiecărui ecran, aplicația conține un director, “utils”, unde sunt stocate constante cu uz global:

VII.3.1 Enums

- este un fișier dedicat enum-urilor și ajută la claritatea și ușurința de modificare a codului scris
- există două enumerări în proiect, cel dedicat cheilor din AsyncStorage și cel ce conține denumirea fiecărei rute a aplicației
- astfel, oricând se dorește a modificare a acestora, este suficientă o intervenție într-un singur loc

VII.3.2 Filters

- deși utilizate doar în ecranul de editare a unei fotografii, filtrele au un conținut ridicat și s-a preferat stocarea lor într-un fișier separat și importat ulterior în ecran
- aici sunt declarate atât filtrele de tipul proprietate, cât și efectele, fiecare având și o denumire și factorul de multiplicare în raport cu bara de intensitate (aceasta având valori între 0 și 1)

VII.3.3 Style

- pentru a păstra uniformitatea interfeței, s-a configurat o temă a aplicației, formată din culori (primary, secondary etc), constante de padding și de spațiere între componente, raza generală a unui border, dimensiunile fonturilor etc.
- acestea au fost utilizate în fiecare obiect de stilizare a componentelor și, de asemenea, în cadrul customizării temei pentru React Navigation

Având toate acestea în vedere, aplicația a fost dezvoltată într-o manieră unitară ce permite dezvoltarea continuă.

O primă eficientizare a aplicației ar putea reprezinta posibilitatea de stocare a fotografiilor pe Cloud, în locul memoriei cache. De asemenea, s-ar putea crea o comunitate online în cadrul căreia să fie postate fotografii editate de utilizatori, care să includă servicii de like, share etc.

O modalitate de a obține valoare monetară în urma aplicației este posibilitatea de cumpărare a unor filtre speciale. Aplicația poate include, la crearea contului, un trial de un număr dat de zile în

cadrul căruia utilizatorul să aibă acces la toate filtrele de care aplicația dispune, incluzând cele contra cost. După expirarea acestui termen, vor fi disponibile doar filtrele gratuite și va exista posibilitatea cumpărării celorlalte, individual sau sub formă de pachet.

VIII. Concluzii

Există mai multe nivele de abstractizare atunci când vine vorba de biblioteca React Native. Este nevoie de înțelegerea conceptului de Single Page Application care, pe scurt, reprezintă rescrierea dinamică a conținutului paginii în momentul în care se dorește modificarea conținutului afișat. Pornind de la acesta, React este o librărie ce se ocupă de randarea conținutului unei pagini în mod dinamic, eficientizând modalitatea de a construi interfețe pentru utilizator.

React Native reprezintă o modalitate optimă pentru dezvoltarea aplicațiilor mobile, într-o manieră cât mai puțin costisitoare ce păstrează performanța unei aplicații native. Modul de dezvoltare al unei aplicații utilizând React Native se aseamănă dezvoltării de pagini web prin intermediul React, fapt ce eficientizează procesul de învățare prin acoperirea a două platforme. De asemenea, pentru a ușura procesele de dezvoltare, build și deploy, poate fi utilizată platforma Expo, construită special pentru dezvoltarea aplicațiilor în React Native. Dezavantajul ei îl reprezintă limitarea în dezvoltare, prin urmare, fiind necesară o analiză amplă a funcționalităților dorite înainte de alegerea acestei opțiuni.

În cadrul React Native, datorită podului dintre domeniile Javascript si limbajul nativ, componentele scrise în Javascript devin, după compilare, elemente native de interfață a căror performanță va fi dependentă doar de sistemul de operare al telefonului.

Prelucrarea imaginilor presupune modificarea matricei de pixeli a unei fotografii prin intermediul aplicării unor filtre. Un filtru este definit printr-o matrice a cărei valori descriu modul în care se modifică fiecare canal al unui pixel, luând în considerare valorile inițiale ale pixelului respectiv.

Pentru a realiza filtrul particularizat Sepia, s-au păstrat proporțiile canalelor de culoare pe fiecare pixel și s-a dezvoltat matricea nouă de transformare în funcție de culoarea dată ca valoare de intrare și de maparea 1 la 127.

Funcționalitatea de creare de filtre reprezintă elementul distinctiv față de aplicațiile existente pe piață. Aceasta poate fi utilizată în cadrul procesului de înțelegere al impactului matricei de transformare asupra fotografiilor și atribuirii de valori diferite parametrilor acesteia. De asemenea, un utilizator experimentat poate crea filtre cât mai diverse și unice, pe care le poate accesa de pe dispozitivul propriu oricând.

IX. Bibliografie

[1] Andro Babu, *Evolution from Web Sites to Web Apps — PWA,* https://medium.com, publicat la data de 26 martie 2018, accesat ultima dată pe data de 15 februarie 2020, la URL: https://medium.com/beginners-guide-to-mobile-web-development/evolution-from-web-sites-to-web-apps-pwa-6aa25aeecd2b

[2] Bonnie Eisenman, *Learning React Native*, https://www.oreilly.com, accesat ultima dată pe data de 15 martie 2020, la URL: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html

[3] Constantin Vertan, *Prelucrarea şi analiza imaginilor,* 1999

[4] John R. Jensen, Steven R. Schill, *Contrast Enhancement*, http://knightlab.org, accesat ultima dată pe data de 20 martie 2020, la URL: http://knightlab.org/rscc/legacy/RSCC_Contrast_Enhancement.pdf

[5] Mosh Hamedani. *React Tutorial - Learn React - React Crash Course*, https://www.youtube.com/channel/UCWv7vMbMWH4-V0ZXdmDpPBA, publicat la data de 26 ianuarie 2018, accesat ultima dată pe data de 15 februarie 2020, la URL: https://www.youtube.com/watch?v=Ke90Tje7VS0 ,

[6] Neoteric, *Single-page application vs. multiple-page application,* https://medium.com, publicat la data de 2 decembrie 2016, accesat ultima dată pe data de 25 ianuarie 2020, la URL: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58

[7] React Native Documentation, https://reactnative.dev/docs/getting-started, ultima dată pe data de 27 ianuarie 2020

[8] Tal Kol, *Performance Limitations of React Native and How to Overcome Them,* https://medium.com, publicat la data de 19 ianuarie 2016, accesat ultima dată pe data de 17 martie 2020, la URL: https://medium.com/@talkol/performance-limitations-of-react-native-and-how-to-overcome-them-947630d7f440

[9] Tal Kol, *Introduction to React Native Performance,* https://speakerdeck.com, publicat la data de 20 iulie 2016, accesat ultima dată pe data de 17 martie 2020, la URL: https://speakerdeck.com/talkol/introduction-to-react-native-performance

[10] University of Tartu, *Digital Image Processing*, https://sisu.ut.ee/imageprocessing/book/1, accesat 3 martie 2020

[11] Vitaly K, *Native vs. Hybrid App Development: Which Approach to Choose?*, https://www.cleveroad.com, publicat la data de 5 octombrie 2019, accesat ultima dată pe data de 13 martie 2020, la URL: https://www.cleveroad.com/blog/native-or-hybrid-app-development-what-to-choose

[12] Wikipedia, *History of mobile phones*, https://www.wikipedia.org/ , accesat ultima dată pe data de 20 mai 2020, la URL: https://en.wikipedia.org/wiki/History_of_mobile_phones

[13] Wikipedia, *React (web framework)*, https://www.wikipedia.org/ , accesat ultima dată pe data de 27 ianuarie 2020, la URL: https://en.wikipedia.org/wiki/React_(web_framework)

[14] Wikipedia, Negative *(photography)*, https://www.wikipedia.org/ , accesat ultima dată pe data de 7 iunie 2020, la URL: https://en.wikipedia.org/wiki/Negative_(photography)