Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka Techniczna (IGM)**

PRACA DYPLOMOWA
INŻYNIERSKA

Projekt i implementacja aplikacji mobilnej do edycji zdjęć

Stanisław Kurzyp

Opiekun pracy
dr. inż. Tomasz Walkowiak

Słowa kluczowe: przekształcanie obrazu, aplikacja mobilna, akceleracja sprzętowa

WROCŁAW 2024

STRESZCZENIE

Celem pracy było napisanie aplikacji mobilnej umożliwiającej użytkownikowi edycję zdjęć. Została zaprojektowana i zrealizowana aplikacja mobilna do edycji zdjęć z wykorzystaniem shader'ów WebGL do operacji na macierzach zdjęcia. Aplikacja wykonana została w technologii React Native przy wsparciu framework'a Expo i wykorzystaniu bibliotek komponentów. Zaimplementowane rozwiązanie oferuje możliwość wizualnej zmiany wyglądu dostarczonego zdjęcia zgodnie z preferencjami jej użytkownika, poprzez złożenie kolejnych przekształceń obrazu.

ABSTRACT

The goal of the project was to develop a mobile application that allows users to edit photos. A mobile photo editing application was designed and implemented using WebGL shaders for operations on image matrices. The application was developed with React Native technology, supported by the Expo framework and component libraries. The implemented solution provides the ability to visually alter the appearance of a given photo according to the user's preferences by applying a sequence of image transformations.

SPIS TREŚCI

1. WPROWADZENIE

Edycja zdjęć oraz manipulacja ich zawartością są w dzisiejszych czasach codziennością. W dobie rozwoju modeli uczenia maszynowego oraz sztucznej inteligencji edycja zdjęć staje się coraz prostsza i wymaga znacznie mniej wiedzy. Jednym poleceniem wydanym dla modelu językowego możemy dodawać lub usuwać obiekty z obrazu, zmieniać kolory i styl zdjęć, a nawet generować je od podstaw. Wiele z tych operacji wymaga zaawansowanych algorytmów wykrywania oraz rozpoznawania obiektów. Jednak w klasycznej edycji zdjęć wszystkie operacje, które skutkują zmianą wyglądu bez zmiany jego zawartości, takie jak dostosowanie saturacji, zmiana jasności czy poprawa balansu bieli, są stosunkowo prostymi działaniami realizowanymi na każdym pikselu lub grupie sąsiadujących pikseli. W niniejszej pracy opisana została realizacja aplikacji mobilnej na platformę Android, która oferuje możliwość klasycznej edycji zdjęć.

W badaniach [28] stwierdzono, że mobilne aplikacje do edycji zdjęć są regularnie używane przez 4 na 10 użytkowników smartfonów. Ponadto w raporcie [17] zakładany jest wzrost wartości rynku z 322 mln dolarów do 447,5 mln dolarów do roku 2032. Warto również zwrócić uwagę na udział aplikacji mobilnych w całym rynku aplikacji do edycji zdjęć. Według badań [29], 59,30% edycji zdjęć przeprowadzane jest na urządzeniach mobilnych działających na systemie iOS lub Android. Powyższe statystyki jasno świadczą o dynamicznym rozwoju tego sektora, co może się wiązać z zapotrzebowaniem na nowe rozwiązania. Za główny powód wzrostu popularności mobilnych aplikacji do edycji zdjęć uznawane są dwa czynniki. Pierwszym z nich jest wzrost popularności mediów społecznościowych oraz istotności wizualnego przekazu medialnego. Drugi to natomiast naturalne następstwo rozpowszechnienia fotografii mobilnej i jej dominacja w dziedzinie fotografii sięgająca według różnych źródeł nawet ponad 90% udziałów w rynku.

Rynek mobilnych aplikacji do edycji zdjęć zdominowany jest przez duże firmy. Najczęściej oferują one swój produkt w modelu, który zakłada udostępnienie części funkcjonalności bezpłatnie, a resztę funkcji zawiera pełna wersja aplikacji, często wymagająca wykupienia kosztownej subskrypcji. Przykładem może być Adobe oraz ich aplikacja "Adobe Photoshop Lightroom" na urządzenia przenośne. Na rynku brak kompletnych, a przy tym bezpłatnych rozwiązań. Ponadto wsparcie dla deweloperów w postaci bibliotek open-source skupionych na edycji zdjęć jest również ograniczone. Skutkuje to trudnościami realizacji nowych, niekomercyjnych, projektów w tej dziedzinie.

1.1. CEL PRACY

Celem pracy było zaprojektowanie i zaimplementowanie rozwiązania mobilnego, które oferuje możliwość edycji zdjęć na platformie Android. Aplikacja umożliwia użytkownikowi wybór zdjęcia z galerii telefonu, wykadrowanie go zgodnie z preferencjami, a następnie dokonanie jego edycji poprzez zmianę wartości zaimplementowanych filtrów w celu poprawy walorów estetycznych. Ważnym aspektem pracy był dobór odpowiednich technologii, który zapewnił płynność działania aplikacji oraz optymalizował proces przekształcania zdjęć.

W ramach projektu trzeba było zapoznać się z przodującymi rozwiązaniami w branży mobilnej edycji zdjęć oraz algorytmami przekształcania obrazu. Praca porusza również temat akceleracji sprzętowej działania programu przy wykorzystaniu karty graficznej urządzenia.

Część wizualna aplikacji została zaplanowana ze starannością o zapewnienie intuicyjności w działaniu oraz przejrzystości. Między innymi wykorzystane zostały gotowe komponenty z biblioteki React Native Paper [24] w celu ujednolicenia wyglądu i zgodności z dobrymi standardami tworzenia interfejsu użytkownika.

Stworzone rozwiązanie opiera się o technologię React Native wspieraną przez framework Expo. React Native oferuje możliwość pisania aplikacji mobilnych w języku Java Script w sposób zbliżony do pisania aplikacji webowych, jednocześnie zapewniając dostęp do natywnych funkcjonalności platformy, na której aplikacja jest uruchamiana.

Ze względu na specyfikę realizowanej aplikacji oraz użytą technologię — React Native — należało zaplanować rozwiązanie dla wysoce wymagających operacji obliczeniowych związanych z przekształcaniem macierzy zdjęcia przez kolejne filtry. W tym celu wykorzystano WebGL — technologię stosowaną w aplikacjach webowych jako połączenie z kartą graficzną urządzenia, aby przyspieszyć operacje graficzne.

1.2. ZAKRES PRACY

Praca zawiera informacje dotyczące przebiegu procesu projektowania aplikacji oraz przegląd zagadnień merytorycznych związanych z zastosowanymi technologiami i rozwiązaniami. W rozdziale
2 przeprowadzona została analiza dostępnych na rynku rozwiązań. Rozdział 3 traktuje o wymaganiach oraz założeniach, jakie realizuje system. Zawiera również opis interakcji użytkownika z systemem i diagram przypadków użycia. Rozdział 4 koncentruje się na zastosowanych technologiach wraz ze szczegółowym wytłumaczeniem, do czego zostały wykorzystane. Wskazane zostały również ich wady i zalety w kontekście tego projektu oraz uzasadnienie dla ich wyboru. Następnie w rozdziale 5 przeanalizowano temat ekosystemu React Native, jego nowej architektury oraz wykorzystywanego frameworka Expo. Kolejno w rozdziale 6 poruszono temat akceleracji sprzętowej obliczeń na urządzeniach mobilnych oraz istotę tego zagadnienia w kontekście edycji zdjęć. W rozdziale 7 zaprezentowane są

przykłady implementacji istotnych części aplikacji wraz z fragmentami kodu źródłowego. Rozdział 8 przedstawia zaimplementowane przekształcenia obrazu, fragmenty kodu za nie odpowiedzialne oraz efekty ich zastosowania. Następnie w rozdziale 9 zawarto podręcznik użytkowania, w ramach którego przedstawione zostały widoki aplikacji wytłumaczenie jak korzystać z aplikacji. W rozdziale 10 podjęto się oceny zaproponowanego rozwiązania oraz dokonano weryfikacji wymagań i założeń zdefiniowanych na początku pracy. Przedstawiono również perspektywy na rozwój aplikacji.

2. PRZEGLĄD DOSTĘPNYCH ROZWIĄZAŃ

Branża edycji fotografii cyfrowej skupiona jest na dwóch głównych platformach: komputerów stacjonarnych i laptopów oraz urządzeń mobilnych takich jak smartfony lub tablety. W zależności od platformy oferta dostępnego oprogramowania różni się, więc przegląd dokonany został osobno dla poszczególnej platformy.

2.1. KOMPUTERY STACJONARNE I LAPTOPY

Adobe Lightroom

Adobe Lightroom[1] to najpopularniejsze oprogramowanie do edycji zdjęć wśród fotografów [10]. Doceniane jest ze względu na swoją wszechstronność oraz szeroką gamę dostępnych funkcji. Oferuje wszystkie potrzebne fotografom funkcje, które od wersji 13.3 rozszerzone zostały poprzez wykorzystanie sztucznej inteligencji *Generative AI*. Oferuje ona możliwość zaznaczenia obszaru na zdjęciu, a następnie poprzez napisanie polecenia w oknie czatu, zdefiniowanie co model sztucznej inteligencji ma w tym obszarze wygenerować. Pozwala to przyspieszyć wiele manualnych etapów edycji zdjęcia takich jak retusz niechcianych obiektów.

Oprócz *Generative AI*, Adobe Lightroom wyróżnia się również skoordynowanym ekosystemem Adobe pozwalającym na udostępnianie plików z jednej aplikacji Adobe do drugiej oraz wykorzystanie chmury *Creative Cloud*. *Creative Cloud* oferuje możliwość przechowywanie określonej ilości danych poza dyskiem komputera. Ponadto dostępna jest webowa wersja Lightroom Web, która nie wymaga instalacji. Korzystanie z Adobe Lightroom wymaga jednak wykupienia subskrypcji, a darmowe korzystanie możliwe jest tylko w ramach okresu próbnego.

2.1.1. Gimp

Gimp[2] to bezpłatny otwartoźródłowy program do edycji grafiki rastrowej. Jego możliwości bardziej zbliżone są do Adobe Photoshop niż Adobe Lightroom i skierowany jest w większym stopniu do przetwarzania lub tworzenia grafik niż do edycji zdjęć. Pomimo tego oferuje on podstawowe operacje edycji zdjęć takie jak zmiana jasności, kontrastu, czy też wyostrzanie. Jednakże warto zaznaczyć, że nie jest on bezpośrednią konkurencją

[1] `https://lightroom.adobe.com/`
[2] `https://www.gimp.org/`

dla programów do edycji zdjęć, a wspominany został ze względu na to, że wyróżnia się otwartym dostępem do kodu źródłowego oraz brakiem opłat za użytkowanie.

2.1.2. Canva

Canva[3] to darmowe narzędzie do tworzenia grafik, prezentacji oraz innych materiałów wizualnych. Według badań [10], Canva rozwija się bardzo dynamicznie, przyciągając co raz więcej nowych użytkowników. Oprócz podstawowych możliwości edycji zdjęć oferuje ona szeroką gamę innych funkcjonalności, co czyni z niej uniwersalne narzędzie do prostych zadań graficznych. Ponadto dostarcza przestrzeń w chmurze do przechowywania projektów oraz wsparcie sztucznej inteligencji. Canva skierowana jest do użytkowników tworzących treści na platformy społecznościowe. Według badań [14] w Stanach Zjednoczonych Canva jest najczęściej wybieranym edytorem zdjęć, prześcigając tym samym między innymi Lightroom'a. Warto zaznaczyć, że posiada znacząco mniejsze możliwości w dziedzinie edycji zdjęć i nie jest odpowiednia do ich profesjonalnej obróbki.

2.2. URZĄDZENIA MOBILNE

2.2.1. Lightroom Mobile

Lightroom Mobile[4] to mobilna wersja aplikacji Lightroom autorstwa Adobe. W wersji płatnej oferuje zbliżone możliwości do wersji na platformy stacjonarne. Wersja bezpłatna udostępnia ograniczony zakres narzędzi. W celu uzyskania pełnego dostępu należy wykupić comiesięczną subskrypcję.

2.2.2. Snapseed

Snapseed[5] to darmowa aplikacja autorstwa firmy Google, która oferuje szeroką gamę narzędzi do edycji zdjęć. Nie posiada ona płatnej wersji i jest dostępna zarówno na platformę Android jak i iOS. Skierowana jest do średnio-zaawansowanych fotografów i brakuje w niej wielu przydatnych funkcjonalności dostępnych w płatnych wersjach innych programów, np. Adobe Lightroom Mobile.

2.3. PODSUMOWANIE

Branża fotografii mobilnej dynamicznie się rozwija, lecz w oparciu o jej strukturę [14] można stwierdzić, że zdominowana jest przez projekty dużych korporacji. Ponadto produkty oferowane przez liderów rynku są najczęściej płatne lub bardzo ograniczone w darmowej wersji. Brakuje solidnych rozwiązań otwartoźródłowych mogących mierzyć się z płatną

[3] `https://www.canva.com/`
[4] `https://www.adobe.com/pl/products/photoshop-lightroom/mobile.html`
[5] `https://play.google.com/store/apps/details?id=com.niksoftware.snapseed`

konkurencją. Ponadto React Native jest dynamicznie rozwijającą się technologią, która wraz z pojawieniem się nowej architektury opisanej w rozdziale 5.1.4 ma potencjał przyspieszenie swojego rozwoju. Brak jednak nowych aplikacji oraz wspieranych bibliotek dla w React Native, które pozwalają na wydajną edycję zdjęć na urządzeniach mobilnych. Aplikacja to pokazuję, że ta technologia jest wystarczająco wydajna i elastyczna w implementacji, by umożliwić ich tworzenie.

3. WYMAGANIA

W poniższym rozdziale przedstawione zostały wymagania funkcjonalne oraz niefunkcjonalne dla aplikacji mobilnej do edycji zdjęć. Ponadto zawiera on również diagram przypadków użycia.

3.1. WYMAGANIA FUNKCJONALNE

3.1.1. Wybór zdjęcia

Po uruchomieni aplikacji użytkownik ma możliwość wyboru zdjęcia. Użytkownik znajdujący się na ekranie startowym aplikacji może kliknąć w przycisk *Choose photo* by zapoczątkować wybór zdjęcia. Spowoduje to otwarcie natywnego modalu, którego wygląd zależny jest od systemu operacyjnego urządzenia. Umożliwia on wybór zdjęcia do edycji z galerii telefonu lub podpiętego dostawcy rozwiązań chmurowych do przechowywania plików, na przykład domyślnie Google Drive dla Android'a. Wybór zdjęcia zakończony sukcesem prowadzi do otworzenia kolejnego modalu z pierwszym krokiem edycji. Wybór zdjęcia niezakończony sukcesem prowadzi do wyświetlenia alertu o niepowodzeniu i wycofaniu do widoku startowego.

3.1.2. Przycięcie oraz obrót zdjęcia

Po wyborze zdjęcia nastąpi uruchomienie kolejnego modalu, który oferuje możliwość przycięcia zdjęcia, obrócenia go oraz przekształcenia w lustrzane odbicie. Wycofanie się z tego widoku prowadzi do wyświetlenia alertu o niepowodzeniu i powrotu do widoku startowego.

3.1.3. Edycja zdjęcia

Kolejny widok oferuje możliwość edycji zdjęcia poprzez regulowanie wartości poszczególnych filtrów. Filtry zawarte są w dwóch zakładkach menu, a ich wartości zmieniane są za pomocą suwaków. Wycofanie się z tego widoku spowoduje powrót do ekranu startowego. Dokładny opis dostępnych filtrów zawarty jest w rozdziale 8.

3.1.4. Zapis zdjęcia

Po zakończonej edycji użytkownik może zapisać efekty swojej pracy w pamięci lokalnej urządzenia poprzez kliknięcie przycisku *Save*. O efekcie operacji zostanie poinformowany

poprzez wyświetlenie się modalu z informacją. Zapisane zdjęcie widoczne będzie w galerii zdjęć urządzenia. Użytkownik może kontynuować edycję i ponownie zapisać zdjęcie.

3.2. WYMAGANIA NIEFUNKCJONALNE

3.2.1. Wydajność

Kluczowym wymaganiem jest zapewnienie odpowiedniej wydajności tworzonej aplikacji. Proces edycji zdjęcia wymaga wielu obliczeń podczas przekształcania macierzy obrazu. Każda zmiana wartości suwaka zmienia wartość filtru, któremu on odpowiada. Skutkuje to ponownym przekształceniem obrazu i wykonaniem wielu obliczeń. Aplikacja powinna zapewnić płynność działania i wykonywać przekształcenia natychmiastowo dla ludzkiego oka. Za płynne działanie uznawane jest takie działanie, gdzie efekt zmian wartości filtra widoczny jest w czasie krótszym niż 40 milisekund, czyli około 24 klatek na sekundę. Jest to wartość, w jakiej ludzie oko uznaje obraz za płynny, a 24 klatki na sekundę stosowane są w kinematografii [3]. Testy przeprowadzone zostaną na urządzeniu Google Pixel 7 Pro z systemem operacyjnym Android 15.

3.2.2. Intuicyjność

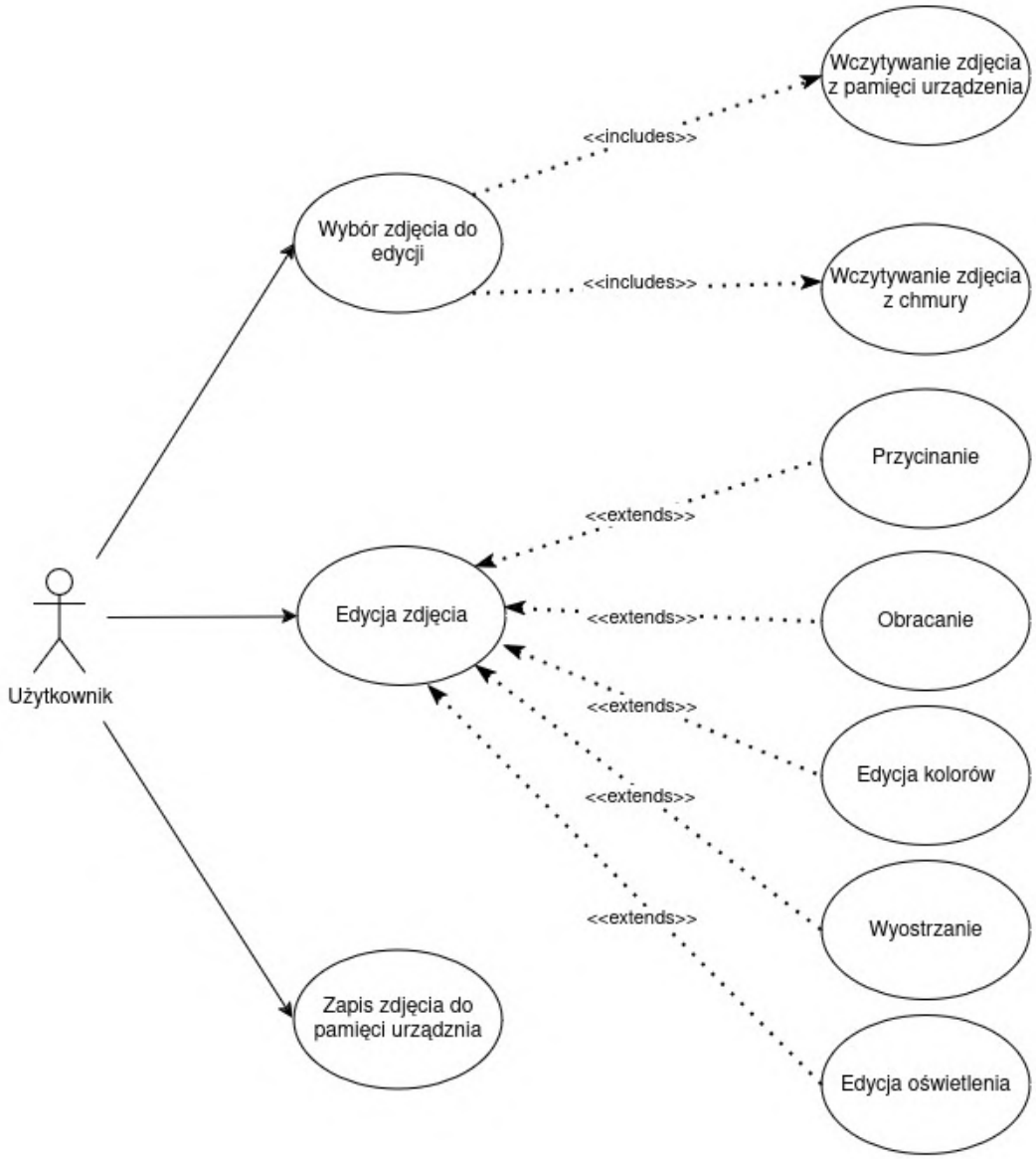
Interface użytkownika powinien być prosty i intuicyjny. Dostępne akcje dla użytkownika powinny być jednoznaczne oraz zrozumiałe dla każdego użytkownika.

3.2.3. Rozszerzalność

Aplikacja powinna być tak zaprojektowana, by umożliwić jej dalszy rozwój. Jej architektura powinna umożliwiać dodanie kolejnych funkcjonalności, a w szczególności kolejnych filtrów i przekształceń obrazu.

3.3. DIAGRAM PRZYPADKÓW UŻYCIA

Rysunek 3.1 przedstawia diagram przypadków użycia dla aplikacji mobilnej do edycji zdjęć będącej przedmiotem tej pracy. Zaznaczono na nim wszystkie funkcje, które może wykonać użytkownik. Pierwszą funkcją, którą musi wykonać użytkownik, jest wczytanie zdjęcia. Może on wybrać czy chce wczytać je z pamięci urządzenia, czy z chmury połączonej z urządzeniem mobilnym, na którym korzysta z aplikacji. Po wczytaniu wybranego zdjęcia może dokonać jego edycji, korzystając z wymienionych operacji. *Edycja kolorów* oraz *Edycja oświetlenia* to szerokie grupy różnych dostępnych operacji wpływających kolejno na kolorystykę, oraz jasność zdjęcia. Ponadto użytkownik może dokonać zapisu zdjęcia do pamięci wewnętrznej urządzenia.



Rys. 3.1. Diagram przypadków użycia

4. PRZEGLĄD TECHNOLOGII I NARZĘDZI ZASTOSOWANYCH W PROJEKCIE

4.1. REACT NATIVE

React Native [16] to opensource'owy framework służący do tworzenia międzyplatformowych aplikacji. Wspiera on rozwój aplikacji na platformy Android oraz iOS. Ponadto możliwe jest również tworzenie aplikacji na urządzenia
z systemem AndroidTV, tvOS oraz tworzenie rozwiązań webowych. Ten projekt skupia się na zastosowaniu React Native do stworzenia aplikacji mobilnej na platformę Android,
a dokładny opis funkcjonowania oraz architektury React Native znajduje się w rozdziale 5.1. Framework ten został wybrany do realizacji aplikacji mobilnej ze względu na podobieństwo rozwoju aplikacji do rozwoju aplikacji webowych, dużą popularność - ponad milion pobrań co tydzień [7], a także dobre wsparcie społeczności i dokumentacji.

4.2. EXPO

Expo to opensource'owy framework wspomagający proces tworzenia i rozwoju aplikacji w technologii React Native. Jest on oficjalnie rekomendowany przez twórców React Native w dokumentacji [4]. Do najistotniejszych zalet Expo należy usprawnienie testowania aplikacji poprzez wykorzystanie Expo Go, szeroki wybór bibliotek oraz usprawnienie procesu kompilacji i publikacji produkcyjnej wersji aplikacji. Dokładny przegląd framework'a Expo znajduje się w rozdziale 5.2.

4.3. REACT NATIVE PAPER

React Native Paper to opensource'owa biblioteka komponentów dla aplikacji React Native. Dostarcza ona gotowe implementacje przydatnych komponentów takich jak przyciski, pola tekstowe, paski nawigacji lub modale. Oferowane komponenty są zgodne z *Material Design*, czyli stylem graficznym stworzonym przez Google, który według dokumentacji [6] łączy w sobie zasady dobrego projektowania z innowacyjnym wyglądem. Według twórców React Natvie Paper [24] pozwala na zmniejszenie nakładu pracy przy tworzeniu interfejsu graficznego i zapewnienia spójności wyglądu komponentów w aplikacji.

4.4. EXPOGL

Zgodnie z dokumentacją [2], ExpoGL to biblioteka, która umożliwia wykorzystanie OpenGL ES do renderowania grafiki 2D i 3D w aplikacjach mobilnych opartych na React Native. Główna funkcjonalność Expo GL polega na udostępnieniu komponentu GLView, który działa jako cel renderowania dla OpenGL ES, tworząc kontekst graficzny i wyświetlając go na ekranie urządzenia. Pozwala to na bezpośrednią pracę z grafiką 3D i 2D, umożliwiając tworzenie bardziej zaawansowanych efektów wizualnych w aplikacjach mobilnych. GLView tworzy kontekst graficzny podczas montowania widoku, a jego bufor rysowania jest prezentowany na ekranie aplikacji w każdej klatce animacji.

Dokładny opis zastosowania tej biblioteki i innych, które pośrednio są przez nią wykorzystywane, umieszczony jest w osobnym rozdziale 6;

4.5. ANDROID STUDIO

Android Studio [12] to oficjalne środowisko IDE służące do tworzenia aplikacji na platformę Android. W projekcie został wykorzystany Menadżer Urządzeń dostarczany w ramach Android Studio, który pozwala na tworzenie wirtualnych urządzeń z w pełni funkcjonalnym systemem operacyjnym Android. Dzięki temu narzędziu możliwe jest testowanie aplikacji React Native bez wykorzystania fizycznego urządzenia.

4.6. FIGMA

Figma [11] to narzędzie pozwalające tworzyć interaktywne prototypy widoków aplikacji oraz stron internetowych. W projekcie narzędzie Figma zostało wykorzystane do zaprojektowania interfejs graficzenego aplikacji.

4.7. TYPESCRIPT

TypeScript [21] to opracowane przez Microsoft, rozszerzenie dla języka JavaScript. Wprowadza ono statyczne typowanie oraz dostarcza możliwość definiowania typów dla zmiennych, funkcji, obiektów oraz klas co pozwala na wykrywanie błędów już na etapie pisania kodu, a nie dopiero podczas jego kompilacji lub wykonania. TypeScript oferuje pełne wsparcie dla obiektowego podejścia programowania, w którego skład wchodzą klasy, dziedziczenie, interfejs'y oraz abstrakcje w zgodzie ze standardami ECMAScript. TypeScript posiada kompilator, kompilujący kod TypeScript do JavaScript, który wykonywany może być przez przeglądarki lub inny silnik JavaScript jak w przypadku React Native.

5. EKOSYSTEM REACT NATIVE

5.1. REACT NATIVE

Zgodnie z [20] React Native to opensource'owy framework Java Script autorstwa firmy Facebook. Dzięki swojemu podejściu do tworzenia aplikacji umożliwia pisanie programów działających zarówno na Androidzie, jak i iOS bez konieczności poznawania natywnych języków tych platform oraz tworzenia dwóch oddzielnych projektów dla tej samej aplikacji. Całość oparta jest o język Java Script, a rozwój oprogramowania zbliżony jest do tworzenia aplikacji webowej z wykorzystaniem biblioteki React.js. Ułatwia to znacząco rozpoczęcie prac nad projektem oraz umożliwia ponowne używanie komponentów ze stron internetowych.

Korzyści płynące z wykorzystywania technologii React Natvie do tworzenia aplikacji mobilnych sprawiły, że jest on wykorzystywany przez największych gigantów technologicznych takich jak Tesla, Airbnb, Facebook czy też Shopify. Według [18], ideą działania React Native jest połączenie dwóch elementów - kodu pisanego w JavaScript oraz natywnego kodu platformy, na którą pisana jest aplikacja. Natywny kod dla Android'a realizowany jest w językach Java lub Kotlin, natomiast dla iOS są to języki Swift lub Objective-C. Warto zaznaczyć, że możliwe jest pisanie w pełni funkcjonalnych aplikacji bez pisania kodu natywnego. Większość potrzebnych modułów natywnych dostarczanych jest w bibliotekach, a pisanie własnych modułów jest opcją, a nie wymogiem dla twórców.

5.1.1. Jak działa React Native

Projekt w React Native zawiera zarówno kod pisany w JavaScript jak i w języku Java lub Kotlin dla platformy Android oraz Swift, lub Objective-C dla iOS. Ze względu na wykorzystanie w końcowej aplikacji połączenia dwóch języków, JavaScript i wybranego języka natywnego dla danej platformy, wymagane jest odpowiednie środowisko uruchomieniowe. Natywny język jest obsługiwany przez sam system operacyjny danego urządzenia. JavaScript wymaga natomiast własnego silnika — wirtualnej maszyny — na której zostanie wykonany. Krytycznym fragmentem tej architektury jest komunikacja pomiędzy tymi dwoma środowiskami. Na komunikację składa się przesyłanie danych ze strony natywnej do strony JavaScript i w druga stronę oraz wywoływanie funkcji wraz z przekazaniem ich parametrów pomiędzy obydwoma stronami. Fragmenty natywnego kodu realizujące funkcje programu nazywane są modułami natywnymi. W ramach natywnych modułów

realizowany jest dostęp do wbudowanych funkcjonalności systemu operacyjnego, takich jak obsługa gestów oraz dostęp do modułów zarządzanych przez system operacyjny, na przykład kamera lub sensory urządzania.

5.1.2. Wersja React Native

Do wersji 0.75 React Native aplikacje domyślnie tworzone były w *starej architekturze*. Wraz z wersją 0.76, której wydanie miało miejsce 25 października 2024 roku, wszystkie aplikacje realizowane są w nowej architekturze. Zmiana architektury, która zostanie szczegółowo opisana w dalszej części tego rozdziału, wprowadziła znaczące usprawnienia i przyspieszyła działanie aplikacji budowanych w tej technologii [30]. Mimo że całkowite przejście na nową architekturę nastąpiło od wersji 0.76, to już od wersji 0.68 możliwe jest eksperymentalne realizowanie aplikacji w tej architekturze. Aplikacja opisywana w tej pracy zrealizowana została na wersji 0.75 frameworka React Native i wykorzystuje eksperymentalną *nową architekturę*. Przejście ze starej na nową architekturę dla wersji 0.75 wymaga zmiany jednej flagi w konfiguracji projektu oraz rekompilacji części natywnej projektu. W projekcie wykorzystywany jest framework Expo, którego działanie oraz wpływ na wybór architektury opisane są w kolejnym rozdziale.

5.1.3. Stara architektura

W celu zrozumienia zmian oraz ogólnego sposobu działania framework'a, krótko przedstawiono jego *stara architektura*.

Aplikacja React Native składa się z dwóch głównych części:

1. **Kod JavaScript**
2. **Kod Natywny**

Kod wykonywany jest w trzech wątkach:

1. **Wątek JavaScript** - wykonuje kod JavaScript na wybranym silniku JavaScript
2. **Wątek Natywny/UI** - wykonuje kod natywny oraz obsługuje zdarzenia interface'u użytkownika takie jak renderowanie obrazu lub obsługa gestów.
3. **Wątek Shadow** - oblicza pozycje elementów do wyświetlenia na ekranie

Cała komunikacja pomiędzy JavaScript a wątkiem natywnym realizowana jest przez komponent Mostu (*ang. Bridge*). Komponent ten serializuje wszystkie dane przekazywane z JavaScript do części natywnej. Przesyłane dane są w formacie JSON i muszą zostać deserializowane po drugiej stronie. Warto zaznaczyć, że most jest jednowątkowy i asynchroniczny. Jego wydajność jest ograniczona i pomimo zadowalających wyników niższa niż w przypadku aplikacji całkowicie natywnych.

5.1.4. Nowa architektura

Nowa architektura niweluje potrzebę wykorzystywania mostu, zastępując go JSI (JavaScript Interface), czyli nowym interface'em dla JavaScript napisanym w C++, by zapewnić bezpośredni kontakt do obiektów i funkcji JavaScript. Wykorzystanie C++ powoduje, że ten interface działa na natywnym poziomie i może być wykorzystywany przez wiele urządzeń, co otwiera możliwości rozwoju React Native na inne branże, a nie tylko aplikacji mobilnych. JSI wykorzystywany jest w następujących komponentach *nowej architektury*:

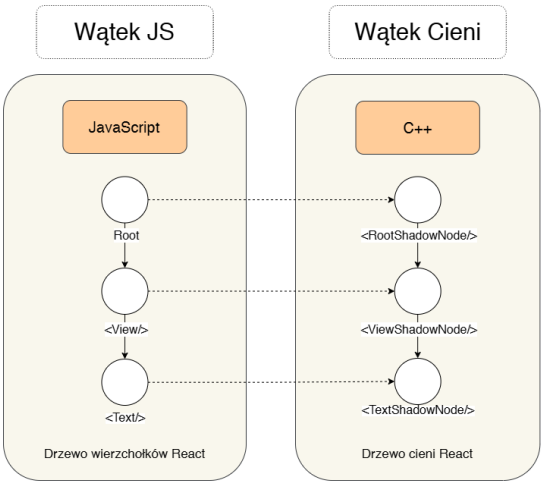
— **Hermes Engine** - jest to silnik JavaScript, który wykonuje kod JavaScript na urządzeniach mobilnych. Pomimo że React Native umożliwia korzystanie z innych silników, Hermes Engine jest wskazany, ze wzgledu na to, że został zoptymalizowany do pracy z React Native i pozwala na zmniejszenie wielkości aplikacji, jej zużycia pamięci oraz czasu włączania.
— **Turbo Module** - jest to nowy system zarządzania natywnymi modułami w aplikacji. Zastąpił on system działający na podstawie most i serializację danych do plików JSON w ramach *starej architektury*. Turbo Module wykorzystuje JSI oraz "Natywny Kod" wygenerowany przez Codegen do zoptymalizowania czasu wczytywania aplikacji poprzez wczytywanie modułów dopiero gdy są faktycznie potrzebne, a nie przy starcie aplikacji. Podejście to nazywane jest leniwym ładowaniem (*ang. lazy loading*) i umożliwione jest dzięki JSI pozwalającemu na przechowywanie referencji do modułów natywnych bezpośrednio w kodzie JavaScript
— **Fabric** - natywny silnik wyświetlania komponentów na ekranie urządzenia. W nowej architekturze udostępnia on swoje funkcje dla Java Script i strony natywnej bez wykorzystania mostu, dzięki JSI.

Warto zaznaczyć, że Turbo Modules wraz z JSI pozwala na pisanie modułów natywnych w języku C++ zarówno na platformie iOS jak i Android. Zgodnie z wypowiedzią twórcy biblioteki *react-native-mmkv* [30] rozwiązanie to pozwoliło zaimplementować bibliotekę w całości jako natywny moduł pisany w C++. Twórca biblioteki *Reanimated* wskazuje również, że zastosowanie modułów natywnych w C++ pozwala na pisanie kodu, który realizowany może być zarówno na platformie Android jak i iOS, co ułatwia i usprawnia rozwój oraz jest jedną z głównych idei framework'a React Native.

Proces wyświetlania aplikacji

Wyświetlanie aplikacji, zwane dalej renderowaniem aplikacji, realizowane jest przez silnik Fabric. Proces ten składa się z 3 faz:

1. faza renderowania
2. faza zatwierdzania
3. faza montowania



Rys. 5.1. Schemat fazy renderowania

Faza renderowania

W tej fazie React wykonuje kod w celu utworzenia drzewa elementów React. Elementy React to proste obiekty JavaScript opisujące wyświetlające się na ekranie komponenty. Na podstawie drzewa elementów React powstaje drzewo cieni React (*ang. react shadow tree*) w C++. Fabric tworzy drzewo cieni z węzłów React (*react nodes*) reprezentujących komponenty, które mają być następnie wyświetlone na ekranie.

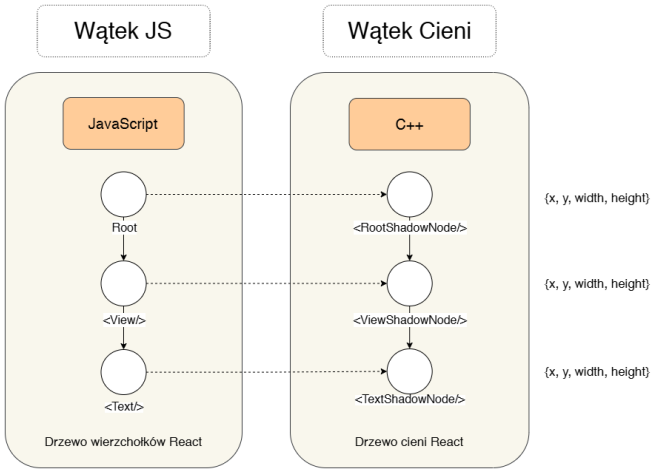
```
const App = () => {
  return (
    <View>
      <text>Hello World</Text>
    </View>
  );
};
```

Listing 5.1: Przykładowy komponent React

Stworzone w fazie renderowania, przedstawionej na rysunku 5.1, drzewo cieni odzwierciedla relacje rodzic-dziecko istniejące pomiędzy węzłami React. Gdy proces renderowania się kończy, wywołuje on proces zatwierdzania.

Faza zatwierdzania

W fazie zatwierdzania, przedstawionej na rysunku 5.2, wykorzystywany jest między-platformowy silnik układu Yoga, który dokonuje wyliczeń położenia oraz wymiarów dla poszczególnych węzłów cieni React, czyli składowych drzewa cieni. Następnie, po przetworzeniu całego drzewa cieni, zostaje ono wyznaczone jako kolejne do zamontowania

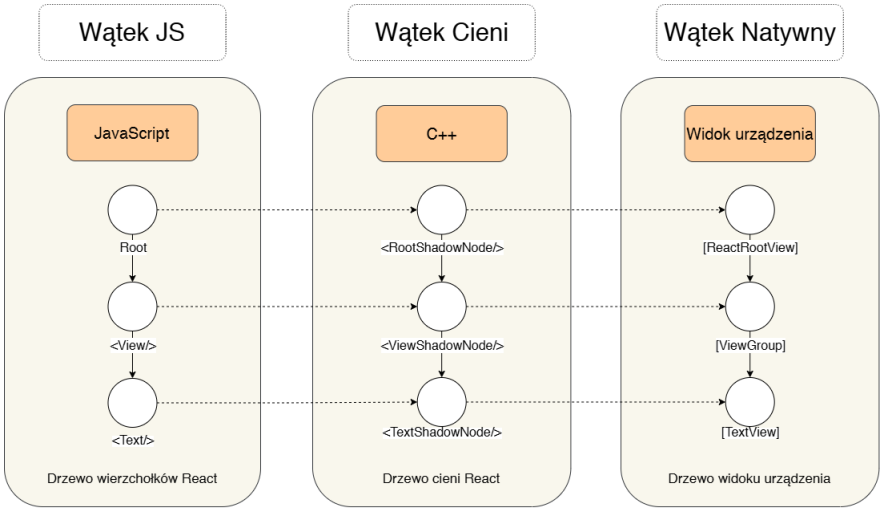


Rys. 5.2. Schemat fazy zatwierdzania

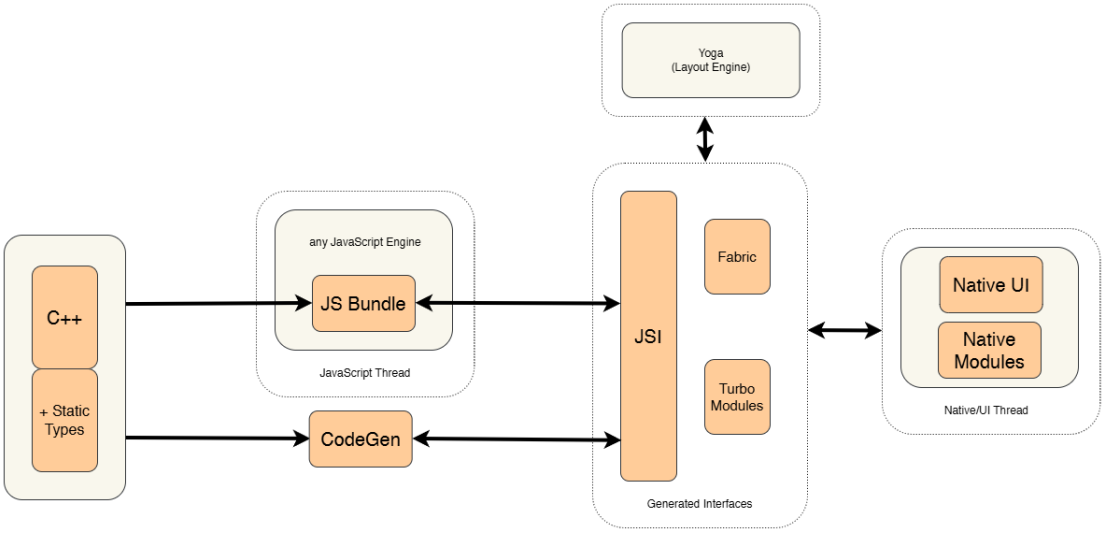
(*ang. mount*). Oznacza to, że przedstawia ono kolejny stan interface'u graficznego aplikacji do wyświetlenia na wyświetlaczu urządzenia.

Faza montowania

Jest to faza, w której drzewo cieni React zawierające obliczenia wykonane przez silnik Yoga zostaje przekształcone w drzewo widoków hosta, czyli urządzenia, na którym uruchomiona została aplikacja React Native. Oznacza to, że zostają przygotowane informacje o tym, jakie piksele wyświetlacza opisują poszczególne komponenty i jak wyświetlić na nim obecny stan aplikacji. Przedstawiona została ona na rysunku 5.3.



Rys. 5.3. Schemat fazy montowania



Rys. 5.4. Przepływ danych w nowej architekturze React Native. W oparciu o [27].

Przepływ danych w nowej architekturze

Poniższa sekcja opisuje przepływ informacji oraz logikę nowej architektury React Native.

1. Działanie aplikacji zaczyna się od wczytania kodu bajtowego, który zawiera kod JavaScript aplikacji. Kod ten przetworzony zostaje przez silnik Hermes.
2. Następnie kod JavaScript komunikuje się z kodem natywnym przy wykorzystaniu JavaScript Interface (JSI).
3. Natywny kod zawiera natywne moduły, które tworzone są z wykorzystaniem Turbo Module, czyli nowego modułu do zarządzania natywnymi modułami, który zapewnia by były one dostępne poprzez JSI z poziomu kodu Java Script.
4. Następnie interface użytkownika zostaje zdefiniowany przez kod JavaScript z wykorzystaniem komponentów React. Komponenty te zostają przetworzone w procesie renderowania przez silnik Fabric.
5. Fabric korzysta z biblioteki Yoga w celu obliczenia pozycji oraz wymiarów poszczególnych komponentów na wyświetlaczu urządzenia.
6. W celu wyświetlenia komponentów na ekranie Fabric korzysta z komponentu Fabric Renderer by narysować interface użytkownika na ekranie z wykorzystaniem natywnego API graficznego. W przypadku iOS jest to Core Animation oraz SurfaceView dla Android'a. Fabric Renderer jest odpowiedzialny również za przetwarzanie animacji oraz gestów.

5.2. EXPO

Zgodnie z dokumentacją [15] to wszechstronne narzędzie i framework do tworzenia aplikacji mobilnych, który wspiera programistów na każdym etapie procesu deweloperskiego. Zintegrowane środowisko deweloperskie oferuje wbudowane narzędzia oraz automatyczną konfigurację kodu natywnego. Expo automatyzuje wiele kluczowych procesów związanych z tworzeniem oraz testowaniem aplikacji. Dostarcza ono narzędzie linii poleceń ułatwiające startowaniem aplikacji oraz serwera testowego.

5.2.1. Rozwój i testowanie aplikacji

Aplikację testować można na trzy sposoby

1. Testowanie na symulatorze Android Studio
2. Testowanie z wykorzystaniem aplikacji Expo Go
3. Testowanie bezpośrednio na fizycznym urządzeniu

Pisanie aplikacji React Native z wykorzystaniem Expo oferuje wiele możliwości jej testowania. Wartym uwagi jest sposób z wykorzystaniem aplikacji Expo Go. Budując aplikację React Native, korzystając z narzędzia linii poleceń Expo, tworzy się lokalny serwer deweloperski. Następnie z poziomu fizycznego urządzenia, poprzez aplikację Expo Go, należy połączyć się z tym serwerem. Expo Go pobiera z niego wszystkie wymagane pliki, następnie w swoim środowisku deweloperskim uruchamia aplikację React Native. Podejście to wspiera gorące odświeżanie (*ang. hot reloading*) co ułatwia wprowadzanie i testowanie zmian, a w szczególności prace nad interface'em graficznym aplikacji.

Cały proces uruchamiania aplikacji w ramach Expo Go jest znacząco krótszy niż proces budowania aplikacji do pliku .apk dla platformy Android. W przypadku tej aplikacji budowanie pliku .apk trwa 20 razy dłużej i wymaga ponownego budowania, by wprowadzić każdą zmianę. Expo GO posiada jednak ograniczone możliwości obsługi bibliotek. Wspiera wszystkie oficjalne biblioteki Expo, ale problemy mogą wystąpić przy bibliotekach innych twórców. Ponadto projekty zawierające natywne moduły niezaimplementowane
w ramach Expo Go również nie będą działać poprawnie. Tu warto zaznaczyć, że wszystkie wykorzystywane biblioteki w ramach tego projektu są wspierane przez Expo Go. Do zalet Expo Go należy również to, że daje ono możliwość rozwoju aplikacji na platformę iOS korzystając z komputera z systemem Windows.

Stosując klasyczne podejście, by zbudować aplikację na platformę iOS, wymagany jest system macOS. W przypadku Expo Go można jednakże zbudować aplikację i umieścić ja na serwerze deweloperskim, a następnie wystarczy połączyć się z tym serwerem z urządzenia z systemem iOS poprzez aplikację Expo Go. Budowanie aplikacji może zostać wykonane zarówno na komputerze z systemem Windows jak i macOS oraz nie jest wymagane do tego środowisko xCode i konto dewelopera Apple. Kolejny istotny fakt na temat rozwoju aplikacji

React Native to różnica pomiędzy *Expo Go build*, a *development build*. Według twórców Expo Go [23] dobrze sprawdza się do szybkiego prototypownia aplikacji. Jak zostało wcześniej wspomniane, ma ono jednak pewne ograniczenia. W celu przetestowania aplikacji w wersji docelowej to znaczy w skompilowanej do wersji rozwojowej (*ang. development build*) można wykorzystać narzędzie Expo Application Services (EAS) oferowane przez Expo. Korzystając z narzędzia linii poleceń EAS można zalogować się do konta Expo, a następnie zainicjalizować projekt Expo. Po utworzeniu projektu możliwe jest kompilowanie aplikacji do wersji rozwojowej z wykorzystaniem serwerów Expo. Oznacza to, że ponownie niewymagane jest posiadanie xCode lub Android Studio na komputerze, na którym pisana jest aplikacja. Wersja rozwojowa aplikacji różni się od wersji uruchamianej w środowisku Expo tym, że posiada swoje własne środowisko uruchomieniowe JavaScript oraz jest skompilowana pod konkretną platformę — Android lub iOS. Ponadto wspiera ona własne moduły natywne oraz jest polecanym rozwiązaniem przy tworzeniu komercyjnych aplikacji. Warto zaznaczyć również, że wersja rozwojowa wspiera gorące odświeżanie,
o ile wprowadzone w projekcie zmiany nie zmieniają natywnego kodu aplikacji.

5.2.2. Release aplikacji

Release to proces przygotowania i udostępnienia aplikacji w wersji skierowanej do jej klienta końcowego. Expo Application Services przydatne jest również podczas kompilowania aplikacji do wersji produkcyjnej (*ang. production build*). Zgodnie z dokumentacją [1], EAS oferuje wsparcie w budowaniu aplikacji zarówno do postaci .apk dla Android jak i predefiniowane kompilacje do dystrybucji w App Store lub Play Store, czyli sklepów aplikacji dla platform iOS oraz Android. Budowanie aplikacji odbywa się na serwerach Expo i jest objęte ograniczeniami w zależności od wybranego planu usługi. W wersji bezpłatnej, która została wykorzystana podczas tworzenia tego projektu, można wykonać 30 kompilacji na miesiąc do wersji produkcyjnej aplikacji.

6. AKCELERACJA SPRZĘTOWA I WEBGL

Aplikacja wykorzystuje procesor graficzny urządzenia, zwany dalej GPU (*ang. Graphics Processing Unit*) lub kartą graficzną, w celu akceleracji obliczeń graficznych. W tym rozdziale omówiono zalety wykorzystania GPU w aplikacjach mobilnych do edycji zdjęć, przybliżono rolę biblioteki `expo-gl`[2] oraz wyjaśniono sposób działania technologii WebGL[25] i OpenGL ES[19].

6.1. ZASTOSOWANIE GPU W APLIKACJI MOBILNEJ

GPU to jednostka sprzętowa zoptymalizowana do wykonywania obliczeń równoległych, co czyni ją szczególnie efektywną w przetwarzaniu grafiki i obrazów. W aplikacjach mobilnych do edycji zdjęć wykorzystanie GPU pozwala na:

— **Przetwarzanie równoległe** — operacje, takie jak zmiana kontrastu, jasności, nasycenia czy nakładanie filtrów, są realizowane na wielu pikselach jednocześnie.
— **Lepszą wydajność** — GPU jest wydajniejsze niż CPU w obróbce obrazów, zwłaszcza o wysokiej rozdzielczości, co zmniejsza czas przetwarzania i zwiększa responsywność aplikacji.
— **Efektywność energetyczną** — wykonywanie operacji graficznych na GPU zużywa mniej energii w porównaniu do CPU, co jest istotne w urządzeniach mobilnych.
— **Zaawansowane efekty** — dzięki GPU możliwe jest implementowanie złożonych efektów, takich jak rozmycie Gaussa, mapowanie HDR czy dynamiczne zmiany barw zdjęcia.

6.2. INTEGRACJA GPU Z REACT NATIVE I EXPO

Aplikacja została zbudowana w technologii React Native przy użyciu platformy Expo. Kluczowym elementem umożliwiającym wykorzystanie GPU jest biblioteka `expo-gl`, która tworzy kontekst graficzny WebGL, który jest API JavaScript wykorzystującym technologię OpenGL ES. Dzięki temu aplikacja uzyskuje dostęp do funkcji karty graficznej.

6.2.1. Technologia WebGL i OpenGL ES

WebGL (*ang. Web Graphics Library*) to API umożliwiające renderowanie grafiki 3D i 2D w przeglądarkach internetowych oraz aplikacjach mobilnych. WebGL jest implementacją standardu OpenGL ES 2.0 w środowisku JavaScript, co pozwala na:

— Renderowanie grafiki bezpośrednio na GPU,
— Pisanie niestandardowych shaderów w języku GLSL (*ang. OpenGL Shading Language*),
— Tworzenie dynamicznych efektów i transformacji w czasie rzeczywistym.

OpenGL ES (*ang. OpenGL for Embedded Systems*) to standard graficzny zaprojektowany z myślą o urządzeniach mobilnych. Jego lekka i zoptymalizowana architektura umożliwia wydajne wykorzystanie GPU w ograniczonym środowisku sprzętowym.

6.2.2. Funkcjonalność expo-gl

Biblioteka `expo-gl` [2] umożliwia tworzenie kontekstu WebGL w aplikacji React Native. Główne zalety korzystania z tej biblioteki to:

— **Integracja z Expo** — `expo-gl` współpracuje ze środowiskiem Expo oraz umożliwia wykorzystanie Expo Go do testowania aplikacji.
— **Wsparcie dla shaderów** — programowanie efektów wizualnych odbywa się za pomocą shaderów napisanych w GLSL.
— **Obsługa tekstur** — możliwe jest renderowanie obrazów na teksturach GPU, co pozwala na wydajne modyfikacje zdjęć [13].

6.3. PRZYKŁADY ZASTOSOWAŃ GPU W EDYTORZE ZDJĘĆ

W aplikacjach do edycji zdjęć, GPU umożliwia implementację zaawansowanych funkcji, takich jak:

1. **Filtry graficzne w czasie rzeczywistym**: nakładanie efektów, takich jak zmiana tonacji kolorów
2. **Transformacje geometryczne**: obracanie, skalowanie lub deformowanie obrazów z wykorzystaniem macierzy transformacji GPU.
3. **Efekty warstwowe**: dodawanie i edytowanie wielu warstw, takich jak tekst, naklejki czy maski, bez zauważalnego opóźnienia.
4. **Zaawansowane przetwarzanie obrazu**: implementacja algorytmów, takich jak rozmycie Gaussa czy filtrowanie przestrzenne, które wymagają intensywnych obliczeń.

6.4. PODSUMOWANIE

Wykorzystanie GPU w aplikacjach mobilnych do edycji zdjęć pozwala na znaczne zwiększenie wydajności oraz poprawę jakości przetwarzania grafiki. Dzięki integracji z biblioteką `expo-gl` i wykorzystaniu technologii WebGL oraz OpenGL ES, możliwe jest implementowanie zaawansowanych funkcji graficznych w środowisku React Native. Taka architektura zapewnia użytkownikom płynne działanie aplikacji i możliwość edycji zdjęć o wysokiej rozdzielczości w czasie rzeczywistym.

7. IMPLEMENTACJA

Projekt wykonany został z wykorzystaniem framework'a Expo, który rozszerza formę zwykłego projektu React Native zwanego *bare workflow*. Do zarządzania widokami w projekcie wykorzystana została bibliotekareact-navigation, która udostępnia wsparcie dla natywnego stosu (*ang. native stack*), czyli historii wyświetlanych widoków. Dzięki temu możliwe jest wykorzystanie wbudowanych, natywnych gestów cofania systemu operacyjnego danego urządzenia mobilnego. Na listingu 7 przedstawiony jest kod głównego komponentu aplikacji React Native wraz ze strukturą widoków.

Widoki otoczone są kontekstami. W React Native kontekst (*ang. context*) to mechanizm umożliwiający przekazywanie danych pomiędzy komponentami bez konieczności przekazywania ich przez *propsy*, czyli parametry komponentu, na każdym poziomie drzewa komponentów. Jest szczególnie przydatny w przypadku danych globalnych. W listingu 7 wyszczególnić można kontekst <PaperProvider/>, który dostarcza funkcjonalności i stylizację zgodną z biblioteką Material Design poprzez integrację z biblioteką 'react-native-paper'. Dzięki temu aplikacja może korzystać z komponentów takich jak przyciski, pola tekstowe czy karty, które są zgodne z wytycznymi projektowymi Material Design. Ponadto zastosowany został między innymi <EditingProvider/>, którego zadaniem jest umożliwienie dostępu różnym komponentom do aktualnego stanu zdjęcia.

7.1. KOMPONENT <EDITINGSCREEN/>

Widok edycji zdjęć zrealizowany został w sposób przedstawiony na 7.1. W górnej części ekranu znajduje się komponent <GLImage/> zawierający kontekst expo-gl. W dolnej części ekranu znajduje się pasek z trzema przyciskami odpowiedzialnymi za wyświetlanie menu z suwakami do edycji poszczególnych filtrów.

Do stylowania komponentów wykorzystany został StyleSheet, czyli wbudowany w React Native mechanizm definiowania stylów komponentów, który umożliwia efektywne i przejrzyste zarządzanie wyglądem aplikacji. StyleSheet pozwala na definiowanie stylów w formie obiektów JavaScript, co przyspiesza działanie aplikacji dzięki optymalizacjom natywnym.

```
const Stack = createNativeStackNavigator<RootStackParamList>();

export default function App() {
    return (
        <RootSiblingParent>
            <PaperProvider>
                <NavigationContainer>
                    <EditingProvider>
                        <Stack.Navigator
                            initialRouteName="Home"
                            screenOptions={{
                                headerStyle: {
                                    backgroundColor: '#333131',
                                },
                                headerTintColor: '#fff',
                                headerTitleStyle: {
                                    fontWeight: 'bold',
                                },
                            }}>
                            <Stack.Screen
                                name="Home"
                                component={HomeScreen}
                                options={{
                                    title: "react-native-image-editor",
                                }}
                            />
                            <Stack.Screen
                                name="Edit"
                                component={EditingScreen}
                            />
                        </Stack.Navigator>
                    </EditingProvider>
                </NavigationContainer>
            </PaperProvider>
        </RootSiblingParent>
    );
}
```

Listing 7.1: Kod przedstawiający główną strukturę aplikacji.

```
const EditingScreen: React.FC<EditingScreenProps> = ({ route }) => {
    const [isColorSelected, setIsColorSelected] = useState(false);
    const [isLightSelected, setIsLightSelected] = useState(false);
    const [isDetailSelected, setIsDetailSelected] = useState(false);

    return (
        <>
            <View style={styles.container}>
                <GLImage/>
            </View>

            {isColorSelected && <ColorTab />}
            {isDetailSelected && <DetailTab />}
            {isLightSelected && <LightTab />}

            <BottomMenu
                isColorSelected={isColorSelected}
                setIsColorSelected={setIsColorSelected}
                isLightSelected={isLightSelected}
                setIsLightSelected={setIsLightSelected}
                isDetailSelected={isDetailSelected}
                setIsDetailSelected={setIsDetailSelected}
            />
        </>
    );
};

const styles = StyleSheet.create({
    container: {
        padding: '3%',
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'flex-start',
        backgroundColor: '#171722',
        width: '100%',
        height: '70%',
    },
});
```

Listing 7.2: Kod przedstawiający strukturę komponentu `EditingScreen`.

7.2. KOMPONENT `<GLIMAGE/>`

Komponent `<GLImage/>`, przedstawiony na listingu 7.2, enkapsuluje w sobie logikę odpowiedzialną za edycję oraz zapis zdjęcia. Jego główne zadanie to stworzenie kontekstu

dla WebGL, wczytanie shadera ze zdjęciem oraz przeprowadzenie przekształceń na każdym jego pikselu zgodnie z wartościami filtrów uzyskanymi z kontekstu EditingContext.

```
    return (
        <View style={styles.container} ref={GLWrapperViewRef} onLayout={(e)
        ↪  => setDimensions(e)}>

            <GLView style={[styles.glView, {width: '100%', height:
            ↪  WebGLViewPixelHeight}]}
                    onContextCreate={onContextCreate}/>

            <Button
                icon={"content-save"}
                mode={"contained-tonal"}
                testID={"detail"}
                onPress={handleSave}
                style={{margin: 20}}
            >
                Save
            </Button>
        </View>
    );
```

Listing 7.3: Kod przedstawiający strukturę komponentu GLImage.

7.3. STRUKTURA KOMPONENTU

Komponent GLImage, przedstawiony na listingu 7.4 składa się z następujących kluczowych elementów:

1. <GLView/>: Komponent, który obsługuje tworzenie kontekstu WebGL.
2. Funkcje do tworzenia shaderów i programu WebGL.
3. Funkcja onContextCreate: odpowiada za inicjalizację WebGL i ustawienia renderowania.
4. Mechanizm useEffect: aktualizuje uniformy shaderów przy zmianie stanu filtrów.
5. Obsługa zapisu obrazu do galerii z wykorzystaniem MediaLibrary.

7.4. OPIS KLUCZOWYCH FUNKCJI

7.4.1. Funkcja createShader

Funkcja createShader, przedstawiona na listingu 7.5, odpowiada za kompilację shadera w WebGL. Dokonuje tego w następujących krokach:

```
return (
    <View style={styles.container} ref={GLWrapperViewRef} onLayout={(e) =>
    ↪  setDimensions(e)}>

        <GLView style={[styles.glView, {width: '100%', height:
        ↪  WebGLViewPixelHeight}]}
                onContextCreate={onContextCreate}/>

        <Button
            icon={"content-save"}
            mode={"contained-tonal"}
            testID={"detail"}
            onPress={handleSave}
            style={{margin: 20}}
        >
            Save
        </Button>
    </View>
);
```

Listing 7.4: Kod struktury komponentu `GLImage`.

1. Tworzy shader za pomocą `gl.createShader(type)`.
2. Ładuje kod źródłowy shadera (`gl.shaderSource`).
3. Kompiluje shader (`gl.compileShader`).
4. Sprawdza poprawność kompilacji.

7.4.2. Funkcja `linkProgram`

Funkcja `linkProgram`, przedstawiona na listingu 7.6 , łączy vertex shader i fragment shader w program WebGL w następujących krokach

1. Tworzy nowy program (`gl.createProgram`).
2. Dołącza shadery (`gl.attachShader`).
3. Łączy program (`gl.linkProgram`).
4. Sprawdza poprawność procesu, a W przypadku błędu loguje komunikat diagnostyczny.

7.4.3. `onContextCreate`

Funkcja `onContextCreate`, przedstawiona na listingu 7.7, realizuje proces inicjalizacji WebGL.

1. Ustawienie kontekstu: przypisuje kontekst WebGL do zmiennej referencyjnej `glRef`.
2. Pobieranie obrazu: za pomocą `Asset.fromURI` ładuje obraz z podanego URI.

```
const createShader = (gl: ExpoWebGLRenderingContext, type: any, source:
↪  string) => {
    const shader: WebGLShader = gl.createShader(type);
    gl.shaderSource(shader, source);
    gl.compileShader(shader);
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        console.error('Shader compile failed with:',
        ↪  gl.getShaderInfoLog(shader));
        gl.deleteShader(shader);
        return null;
    }
    return shader;
};
```

Listing 7.5: Funkcja `createShader`.

```
const linkProgram = (gl: ExpoWebGLRenderingContext, vertexShader:
↪  WebGLShader, fragmentShader: WebGLShader) => {
    const program: WebGLProgram = gl.createProgram();
    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);
    gl.linkProgram(program);
    if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {
        console.error('Program link failed with:',
        ↪  gl.getProgramInfoLog(program));
        return null;
    }
    return program;
};
```

Listing 7.6: Funkcja `linkProgram`.

3. Kompilacja shaderów: wykorzystuje funkcje `createShader` i `linkProgram`.
4. Ustawienie pozycji wierzchołków oraz tekstury.
5. Tworzenie tekstury i przypisanie jej do GPU.
6. Przypisanie uniformów shaderów do zmiennych kontrolujących filtry.
7. Renderowanie obrazu za pomocą `gl.drawArrays`.

```
const onContextCreate = async (gl: ExpoWebGLRenderingContext) => {
    glRef.current = gl;

    const asset = await Asset.fromURI(imageURI);
    await asset.downloadAsync();

    const vertexShader = createShader(gl, gl.VERTEX_SHADER,
    ↪  vertexShaderSource);
    const fragmentShader = createShader(gl, gl.FRAGMENT_SHADER,
    ↪  fragmentShaderSource);
    const program = linkProgram(gl, vertexShader, fragmentShader);
    gl.useProgram(program);

    const positionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
    const positions = new Float32Array([-1, -1, 1, -1, -1, 1, 1, 1]);
    gl.bufferData(gl.ARRAY_BUFFER, positions, gl.STATIC_DRAW);

    const texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, {
    ↪  localUri: asset.localUri });

    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    gl.endFrameEXP();
};
```

Listing 7.7: Funkcja `onContextCreate`.

7.5. AKTUALIZACJA FILTRÓW

Wartości filtrów są dynamicznie przekazywane do shaderów za pomocą uniformów. Mechanizm `useEffect` zapewnia, że zmiany w stanie kontekstu `EditingContext` automatycznie aktualizują rendering obrazu. Kod odpowiedzialny za jego realizację przedstawiony jest na listingu 7.8.

```
useEffect(() => {
    if (glRef.current) {
        const gl = glRef.current;
        gl.useProgram(gl.getParameter(gl.CURRENT_PROGRAM));
        gl.uniform1f(brightnessLocationRef.current, state.brightness);
        gl.clear(gl.COLOR_BUFFER_BIT);
        gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
        gl.endFrameEXP();
    }
}, [state]);
```

Listing 7.8: Kod aktualizujący uniformy shaderów.

7.6. TESTY WYDAJNOŚCIOWE APLIKACJI

Przeprowadzone zostały testy wydajnościowe aplikacji w celu weryfikacji wymagania płynnego działania przedstawionego w 3.2. Za definicję płynnego działania została uznana wartość 24 klatek na sekundę, czyli standardowa liczba klatek w dzisiejszej kinematografii. W celu weryfikacji płynności zmierzony został czas, w jakim aplikacja przetwarza zdjęcie, nakładając na nie wszystkie zaimplementowane filtry. W efekcie pomiarów otrzymano średnią wartość 13,2 milisekundy. W przypadku filmu z 24 klatkami na sekundę każda klatka jest widoczna na ekranie przez około 41,6 milisekundy. Oznacza to, że zaimplementowane rozwiązanie gwarantuje płynne działanie, a czas wymagany na przetworzenie zdjęcia interpretować można jako równowartość około 76 klatek na sekundę. Powyższa sytuacja ma swoje zastosowanie gdy użytkownik aplikacji przesuwa suwak filtra i zmienia jego wartość, a tym samym następuje wielokrotne przetwarzanie zdjęcia.

8. PRZEKSZTAŁCENIA OBRAZU

Aplikacja mobilna do edycji zdjęć stworzona w ramach tego projektu oferuje trzy główne grupy przekształceń obrazu, zwanych potem filtrami.

1. Przekształcenia oświetlenia - filtry światła
2. Przekształcenia koloru - filtry koloru
3. Przekształcenia szczegółu - filtry szczegółu

8.1. DZIAŁANIE FILTRÓW

Zastosowane filtry wpływają na wartości pikseli obrazu na dwa główne sposoby: poprzez operacje na pojedynczych pikselach lub operacje z uwzględnieniem otoczenia.

8.1.1. Operacje na pojedynczych pikselach

Każdy piksel może być przetwarzany niezależnie, a jego nowa wartość jest wyznaczana jedynie na podstawie jego oryginalnej wartości. Na podstawie [26], przekształcenie piksela można opisać za pomocą funkcji transformującej 8.1.

$$s = T(z), \tag{8.1}$$

gdzie z to intensywność (wartość) piksela w oryginalnym obrazie, T to funkcja transformująca, a s jest nową wartością piksela na obrazie wynikowym.

8.1.2. Operacje z uwzględnieniem otoczenia

Każdy piksel może być przetwarzany na podstawie wartości pikseli znajdujących się w jego sąsiedztwie. Nowa wartość piksela jest funkcją zarówno jego oryginalnej wartości, jak i wartości sąsiednich pikseli w określonym obszarze wokół niego.

Zgodnie z [26], niech $\mathbf{S_{xy}}$ oznacza zbiór współrzędnych pikseli tworzących sąsiedztwo piksela $(\mathbf{x}, \mathbf{y})$. Przetwarzanie z uwzględnieniem otoczenia generuje odpowiednią wartość piksela w obrazie wynikowym g na podstawie operacji przeprowadzonej na pikselach o współrzędnych należących do zbioru $\mathbf{S_{xy}}$.

Załóżmy, że operacją, którą chcemy przeprowadzić, jest obliczenie średniej wartości pikseli w prostokątnym otoczeniu o wymiarach $m \times n$, wyśrodkowanym na danym pikselu o współrzędnych (x, y). Wartości pikseli w tym otoczeniu mają współrzędne należące do

zbioru $\mathbf{S_{xy}}$. Wynik tej operacji, czyli nową wartość piksela $g(x, y)$ w obrazie wynikowym, można wyrazić wzorem 8.2.

$$g(x, y) = \frac{1}{mn} \sum_{(r,c) \in \mathbf{S_{xy}}} f(r, c), \tag{8.2}$$

gdzie $f(r, c)$ to wartość piksela obrazu wejściowego o współrzędnych (r, c), natomiast m i n to liczba pikseli w poziomie i pionie w wybranym otoczeniu.

Przykładowo, w przypadku filtra o rozmiarze 3×3, otoczenie każdego piksela złożone jest z 9 pikseli. Obliczając uśrednioną wartość intensywności na tym obszarze, przypisujemy wynik do piksela $g(x, y)$ w obrazie wynikowym. Obraz g jest tworzony przez przesuwanie otoczenia (o wymiarach $m \times n$) po każdym pikselu obrazu f i powtarzanie operacji średniej dla każdej nowej lokalizacji.

Według [26], podejście to jest często stosowane w filtrach wygładzających (np. filtr dolnoprzepustowy), które redukują szumy w obrazie poprzez uśrednianie wartości sąsiadujących pikseli.

8.1.3. Otoczenie

Otoczenie piksela $(\mathbf{x}, \mathbf{y})$ to zbiór pikseli znajdujących się w jego bezpośredniej okolicy, które mogą wpływać na jego wartość po przetworzeniu. Wyróżnia się następujące typy otoczenia:

1. **Otoczenie 4-elementowe (sąsiedztwo 4-krotne)** – składa się z pikseli o współrzędnych przedstawionych w równaniu 8.3

$$S_{xy\mathbf{4}} = \{(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1)\}. \tag{8.3}$$

2. **Otoczenie 8-elementowe (sąsiedztwo 8-krotne)** – składa się z pikseli w najbliższym otoczeniu piksela $(\mathbf{x}, \mathbf{y})$, zarówno w pionie, poziomie, jak i po przekątnych. Opisuje to równanie 8.4.

$$S_{xy\mathbf{8}} = \left\{ \begin{matrix} (x - 1, y + 1), (x, y + 1), (x + 1, y + 1), \\ (x + 1, y), (x + 1, y - 1), (x, y - 1), \\ (x - 1, y - 1), (x - 1, y) \end{matrix} \right\}. \tag{8.4}$$

Zgodnie z [22], sąsiedztwa pikseli odgrywają kluczową rolę w procesach przetwarzania obrazu. Są one szczególnie istotne przy stosowaniu takich filtrów jak wyostrzanie, rozmycie czy medianowe, które wykorzystują wartości pikseli w określonym otoczeniu, aby osiągnąć pożądany efekt na przetworzonym obrazie.

8.2. FILTRY ŚWIATŁA

Filtry światła to takie przekształcenia obrazu, których zastosowanie wpływa na sposób, w jaki światło interpretowane jest na zdjęciu. Dzięki nim można zmieniać tonację, wpływać na barwę kolorów oraz regulować kontrast zdjęcia. Ta grupa skupia się głównie na poprawie dynamiki i wyrazistości zdjęcia poprzez wpływ na intensywność światła.

8.2.1. Filtr jasności

Filtr jasności (*ang. brightness*) jest jedną z podstawowych operacji przetwarzania obrazu, która polega na rozjaśnieniu lub przyciemnieniu obrazu poprzez dodanie stałej wartości do każdej składowej kolorów RGB pikseli. Rozjaśnianie obrazu odbywa się poprzez zwiększenie wartości intensywności poszczególnych składowych koloru (czerwonego, zielonego i niebieskiego) w każdym pikselu, co skutkuje zwiększeniem ogólnej jasności obrazu. Natomiast zmniejszenie wartości powoduje przyciemnienie obrazu.

Dla każdego piksela $f(x, y)$ o składowych R, G, B, nową wartość piksela $g(x, y)$ po przetworzeniu przez filtr jasności można wyrazić wzorem 8.5.

$$g(x, y) = f(x, y) + \textit{value}, \tag{8.5}$$

gdzie *value* to stała liczba dodawana do każdej składowej koloru (R, G, B), a $f(x, y)$ i $g(x, y)$ oznaczają odpowiednio wartości intensywności RGB piksela przed i po zastosowaniu filtra.

```
vec4 brightness(vec4 color, float value) {
  // Zwiększenie wartości składowych RGB o wartość 'value'
  vec3 rgb = color.rgb + value;
  // Zwracamy nowy kolor z zachowaniem oryginalnej wartości alfa
  ↪  (przezroczystości)
  return vec4(rgb, color.a);
```

Listing 8.1: Przykład implementacji filtra jasności w shaderze GLSL

Kod zawarty w 8.1 implementuje filtr jasności w shaderze GLSL. Funkcja `brightness` przyjmuje jako argumenty oryginalny kolor piksela `color` (typu `vec4`) oraz wartość `value`, która określa, o ile rozjaśnić (wartość dodatnia) lub przyciemnić (wartość ujemna) obraz.

Wewnątrz funkcji:

— `color.rgb` wyodrębnia składowe RGB oryginalnego koloru.
— Dodanie `value` do każdej składowej RGB (`color.rgb + value`) powoduje zwiększenie jasności o określoną wartość.
— Wynikowy kolor jest tworzony jako `vec4(rgb, color.a)`, co oznacza, że do przekształconych składowych RGB dołączana jest oryginalna wartość przezroczystości (alfa) `color.a`.

Dzięki temu rozwiązaniu filtr jasności pozwala na kontrolowane zwiększanie lub zmniejszanie jasności obrazu poprzez prostą operację dodawania do każdej składowej koloru.

Zdjęcia 8.2 oraz 8.3 przedstawia efekt zastosowania filtra jasności na obrazie 8.1.

8.3. FILTRY KOLORU

Filtry koloru to takie przekształcenia obrazu, których zastosowanie wpływa na tonację i nasycenie barw, umożliwiając zmianę nastroju zdjęcia i nadanie mu określonego klimatu. Regulowanie parametrów filtrów pozwala na wyszczególnienie i uwypuklenie wybranych cech edytowanego zdjęcia.

8.3.1. Filtr saturacji

Filtr saturacji (*ang. saturation*) umożliwia modyfikację nasycenia kolorów w obrazie, co wpływa na intensywność i żywotność barw. Zwiększenie saturacji prowadzi do bardziej wyrazistych i intensywnych kolorów, podczas gdy jej zmniejszenie sprawia, że obraz staje się bardziej stonowany, a w skrajnym przypadku czarno-biały (odcienie szarości).

Zmiana saturacji piksela $f(x, y)$ o składowych R, G, B, prowadząca do nowej wartości $g(x, y)$, uwzględnia wyznaczenie odcienia szarości (desaturacji) oraz interpolację pomiędzy tym odcieniem a oryginalnym kolorem w zależności od zadanej wartości saturacji *value*. Wzór ten można zapisać równaniem 8.6

$$g(x, y) = (1,0 + \textit{value}) \cdot f(x, y) + \big(1,0 - (1,0 + \textit{value})\big) \cdot \textit{desaturated}\big(f(x, y)\big), \quad (8.6)$$

gdzie *desaturated*$(f(x, y))$ oznacza wersję piksela $f(x, y)$ o zmniejszonej saturacji, a *value* określa, w jakim stopniu oryginalny kolor ma być zachowany (dla *value* $= -1,0$ uzyskujemy obraz czarno-biały).

Przy obliczeniach odcienia szarości stosuje się wagi zgodne z percepcją ludzkiego oka. Ich wartości określona są w wytycznych dotyczących dostępu do treści internetowych (WCAG) [9]. Luminancję wyznacza wzór 8.7.

$$\textit{luma} = 0,2126 \cdot R + 0,7152 \cdot G + 0,0722 \cdot B. \quad (8.7)$$

Kod zawarty w 8.2 implementuje filtr saturacji w języku GLSL. Funkcje w nim zaimplementowane działają w następujący sposób:

— `applyDesaturated` oblicza odcień szarości poprzez zastosowanie luminancji składowych R, G, B.
— `applySaturation` tworzy nowy kolor jako interpolację między desaturowaną wersją koloru a oryginałem, z uwzględnieniem wartości saturacji *value*.

Rys. 8.1. Oryginalne zdjęcie bez zastosowania filtra.

Rys. 8.2. Zdjęcie po zastosowaniu filtra jasności o dodatniej wartości.

Rys. 8.3. Zdjęcie po zastosowaniu zimnego filtra jasności o ujemnej wartości.

```glsl
// Obliczenie odcienia szarości na podstawie luminancji
vec4 applyDesaturated(vec4 color) {
    vec3 luma = vec3(0.2126, 0.7152, 0.0722);
    vec3 gray = vec3(dot(color.rgb, luma));
    return vec4(gray, color.a);
}

// Zmiana saturacji z wartością value
vec4 applySaturation(vec4 color, float value) {
    vec4 desaturated = applyDesaturated(color);
    return mix(desaturated, color, 1.0 + value);
}
```

Listing 8.2: Implementacja filtra saturacji w shaderze GLSL

Rozwiązanie to pozwala na dowolne dostosowywanie nasycenia obrazu, uzyskując efekty od intensywnych barw po obraz monochromatyczny.

Rysunki 8.5 oraz 8.6 ilustrują wpływ filtra saturacji na przykładzie obrazu 8.4.

8.3.2. Filtr *Hue*

Filtr *Hue* to filtr odcienia koloru. Umożliwia on zmianę podstawowego odcienia kolorów w obrazie. Odcień jest głównym parametrem definiującym kolor.

Model HSV

HSV (Hue, Saturation, Value) to alternatywny model kolorów w stosunku do RGB. W tym modelu kolory o różnych odcieniach (*Hue*) są rozmieszczone promieniście wokół osi neutralnych kolorów (czarny, biały, szary), przy czym:

— Jasność (*Value*) wzrasta od dolnej części osi (czarny) do górnej części (biały).
— Saturacja (*Saturation*) określa intensywność koloru.

Rysunki 8.7 oraz 8.8 przedstawiają różnice między modelem RGB a HSV.

Implementacja filtru zmiany odcienia kolorów

Aby dokonać modyfikacji odcieni kolorów na zdjęciu, należy najpierw przekonwertować kolor z modelu RGB na HSV. W GLSL można to zrobić za pomocą funckji przedstawionej w 8.3.2.

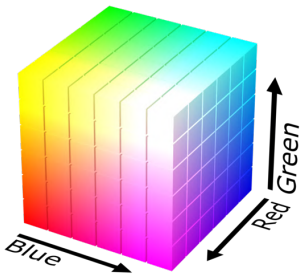
Zmiana odcienia koloru na zdjęciu polega na modyfikacji pierwszej składowej wektora HSV (`HSV[0]`). Implementacja w shaderze GLSL przedstawiona jest w 8.3.2.

Po modyfikacji odcienia obraz konwertowany jest ponownie do modelu RGB za pomocą funkcji przedstawionej w 8.3.2.

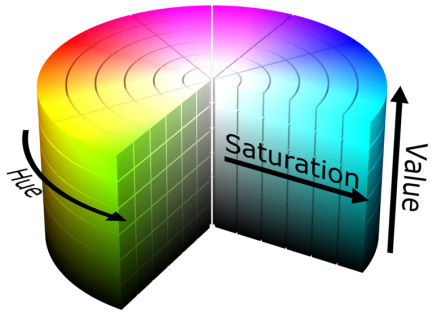
Rys. 8.4. Oryginalne zdjęcie bez zastosowania filtra.

Rys. 8.5. Zdjęcie po zastosowaniu filtra saturacji o dodatniej wartości.

Rys. 8.6. Zdjęcie po zastosowaniu zimnego filtra saturacji o ujemnej wartości.



Rys. 8.7. Reprezentacja przestrzeni barw RGB, Źródło: [8]



Rys. 8.8. Reprezentacja przestrzeni barw HSV. Źródło: [5]

```
vec3 rgb2hsv(vec3 c) {
    vec4 K = vec4(0.0, -1.0 / 3.0, 2.0 / 3.0, -1.0);
    vec4 p = mix(vec4(c.bg, K.wz), vec4(c.gb, K.xy), step(c.b, c.g));
    vec4 q = mix(vec4(p.xyw, c.r), vec4(c.r, p.yzx), step(p.x, c.r));
    float d = q.x - min(q.w, q.y);
    float e = 1.0e-10;
    return vec3(abs(q.z + (q.w - q.y) / (6.0 * d + e)), d / (q.x + e),
    ↪  q.x);
}
```

Listing 8.3: Funkcja konwertująca kolor z modelu RGB na HSV.

```
vec4 applyHue(vec4 color, float value) {
    vec3 hsb = rgb2hsb(color.xyz);
    hsb[0] += (value * 0.5);
    hsb[0] = mod(hsb[0], 1.0);
    vec4 outColor = vec4(hsb2rgb(hsb), 1.0);
    return outColor;
}
```

Listing 8.4: Przesunięcie odcienia w shaderze GLSL.

```
vec3 hsv2rgb(vec3 c) {
    vec4 K = vec4(1.0, 2.0 / 3.0, 1.0 / 3.0, 3.0);
    vec3 p = abs(fract(c.xxx + K.xyz) * 6.0 - K.www);
    return c.z * mix(K.xxx, clamp(p - K.xxx, 0.0, 1.0), c.y);
}
```

Listing 8.5: Funkcja konwertująca kolor z modelu HSV na RGB.

Rys. 8.9. Oryginalne zdjęcie bez zastosowania filtra.

Rys. 8.10. Zdjęcie po zastosowaniu filtra przesunięcia barwowego *hue*.

Efekty filtra odcienia (*Hue*)

8.9 przedstawia oryginalne zdjęcie oraz zastosowanie filtra odcienia 8.10.

8.3.3. Filtr temperatury barwowej

Filtr temperatury barwowej służy dostosowaniu barw kolorów na zdjęciu. Jego działanie symuluje efekt zmiany temperatury światła, która wyrażana jest w kelwinach (K). Temperatury niższe ($< 6500K$) nadają obrazowi cieplejszy odcień (czerwono-pomarańczowy), natomiast wyższe ($> 6500K$) chłodniejszy (niebieski).

Do wyznaczenia wpływu temperatury na kanały RGB zastosowana jest funkcja `kelvinToRGBShift`, która normalizuje temperaturę względem $6500K$, a następnie proporcjonalnie modyfikuje wartości kanałów. Kod realizujący tą funkcję przedstawiony jest na 8.6.

Zastosowanie filtra temperatury

Przesunięcie wartości RGB w obrazie realizowane jest jako przemnożenie oryginalnych wartości składowych RGB przez wyznaczone przesunięcie wynikające ze zmiany temperatury:

```
vec3 kelvinToRGBShift(float kelvinsTemp) {
    float k = kelvinsTemp / 6500.0; // Normalizacja do 6500K
    vec3 rgbShift;

    if (k < 1.0) {
        float ratio = 1.0 - k;
        rgbShift = vec3(1.0 + 0.2 * ratio, 1.0 - 0.1 * ratio, 1.0 - 0.2 *
        ↪  ratio);
    } else if (k == 1.0) {
        rgbShift = vec3(1.0, 1.0, 1.0);
    } else {
        float ratio = k - 1.0;
        rgbShift = vec3(1.0 - 0.2 * ratio, 1.0 - 0.1 * ratio, 1.0 + 0.2 *
        ↪  ratio);
    }

    return rgbShift;
}
```

Listing 8.6: Funkcja konwertująca temperaturę w kelwinach na przesunięcie RGB.

```
vec4 applyTemperature(vec4 color, float value) {
    vec3 rgbShift = kelvinToRGBShift(value);
    return vec4(color.rgb * rgbShift, color.a);
}
```

Listing 8.7: Funkcja GLSL do zastosowania filtra temperatury.

Efekt filtra temperatury

Poniższy przykład prezentuje wpływ zmiany temperatury barwowej na zdjęcie 8.11. Przy niższej obraz 8.13 staje się cieplejszy, natomiast przy wyższej 8.12 nabiera zimniejszego odcienia.

8.4. FILTRY SZCZEGÓŁU

Filtry szczegółu to takie przekształcenia obrazu, których zastosowanie ma wpływ na strukturę i wyrazistość zdjęcia, poprawiając lub zmniejszając ostrość i przejrzystość obrazu. Stosowane są do uwydatniania szczegółów lub wprowadzenia miękkiego, efektu rozmycia.

Rys. 8.11. Oryginalne zdjęcie bez zastosowania filtra.

Rys. 8.12. Zdjęcie po zastosowaniu ciepłego filtra temperatury.

Rys. 8.13. Zdjęcie po zastosowaniu zimnego filtra temperatury.

Rys. 8.14. Oryginalne zdjęcie bez zastosowania filtra.

Rys. 8.15. Zdjęcie po zastosowaniu filtra wyostrzania.

8.4.1. Filtr wyostrzania

Filtr wyostrzania znajduje swoje zastosowanie w przetwarzaniu i edycji obrazów, w celu uwydatnienia szczegółów i krawędzi
w obrazie. Działa on poprzez wykorzystanie splotu (*ang. convolution*) z odpowiednią macierzą (*kernel*), która wzmacnia różnice w jasności między pikselami, zwiększając kontrast na krawędziach.

W 8.8 zawarty jest kod shadera GLSL realizującego filtr wyostrzania wykorzystujący splot macierzy obrazu z kernelem o rozmiarze 3×3.

Efekt działania filtra wyostrzania

Rysunek 8.15 ilustruje wpływ filtra wyostrzania na obraz. Obszary krawędzi stają się bardziej wyraźne, uwydatniając szczegóły obrazu.

```
vec4 applySharpening(sampler2D texture, float sharpen, vec2 v_texcoord,
↪  vec2 resolution) {
    vec2 texelSize = 1.0 / resolution;

    // Definicja sąsiedztwa
    vec2 offset[9];
    offset[0] = vec2(-1.0, 1.0) * texelSize;
    offset[1] = vec2(0.0, 1.0) * texelSize;
    offset[2] = vec2(1.0, 1.0) * texelSize;
    offset[3] = vec2(-1.0, 0.0) * texelSize;
    offset[4] = vec2(0.0, 0.0) * texelSize;  // piksel centralny
    offset[5] = vec2(1.0, 0.0) * texelSize;
    offset[6] = vec2(-1.0, -1.0) * texelSize;
    offset[7] = vec2(0.0, -1.0) * texelSize;
    offset[8] = vec2(1.0, -1.0) * texelSize;

    // Jądro filtra wyostrzania
    float kernel[9];
    kernel[0] = 0.0; kernel[1] = -1.0; kernel[2] = 0.0;
    kernel[3] = -1.0; kernel[4] = 5.0; kernel[5] = -1.0;
    kernel[6] = 0.0; kernel[7] = -1.0; kernel[8] = 0.0;

    vec3 resultColor = vec3(0.0);

    // Obliczenie wartości koloru po splotach z jądrem
    for (int i = 0; i < 9; i++) {
        resultColor += texture2D(texture, v_texcoord + offset[i]).rgb *
        ↪  kernel[i];
    }

    vec3 originalColor = texture2D(texture, v_texcoord).rgb;

    // Interpolacja pomiędzy oryginalnym kolorem a wynikiem splotu
    resultColor = mix(originalColor, resultColor, sharpen);

    return vec4(resultColor, 1.0);
}
```

Listing 8.8: Implementacja filtra wyostrzania w shaderze GLSL.

9. PODRĘCZNIK UŻYTKOWANIA

9.1. URUCHOMIENIA APLIKACJI

Aplikacja oraz kod źródłowy dostępne są w publicznym Repozytorium GitHub. Na wybrane urządzenie z systemem operacyjnym Android należy pobrać plik `.apk` z kodem aplikacji znajdującym się w katalogu `lib`. Następnie należy zainstalować aplikację na urządzeniu poprzez kliknięcie pliku `.apk` i wyrażenie zgody na instalację. Zainstalowana aplikacja znajdzie się w menu aplikacji pod nazwą **react-native-image-editor**. By ją uruchomić, należy kliknąć jej ikonę.

9.2. EKRAN STARTOWY

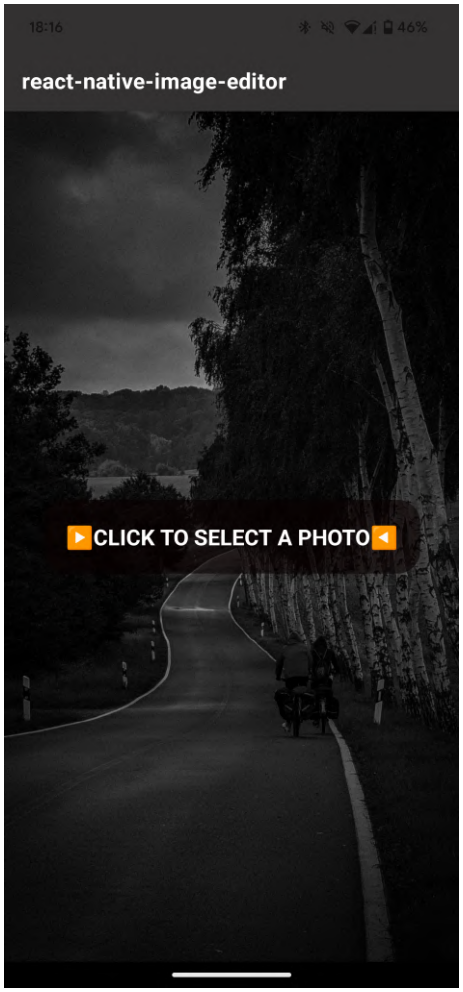
Ekran startowy aplikacji 9.1 przedstawia napis *Click to select a photo*, którego kliknięcie otworzy widget wyboru zdjęcia do edycji. Przedstawiony został on na 9.2. Użytkownik może przeglądać ostanie zapisane zdjęcia na urządzeniu lub przełączyć się na widok Albumów. W prawym, górnym rogu widoczny jest symbol trzech kwadratów jeden na drugim. Kliknięcie w symbol otworzy menu z możliwością przejścia do ustawień połączonej z urządzeniem chmury pamięci, która pozwala na przechowywanie zdjęć. Po połączeniu chmury, w widoku albumów widoczne będą również pliki dostępne w zdalnej lokalizacji. Jeśli użytkownik nie dokona wyboru żadnego zdjęcia, wyświetlone zostanie powiadomienie *You did not select any image* - nie wybrano żadnego zdjęcia.

9.3. WIDOK TRANSFORMACJI ZDJĘCIA

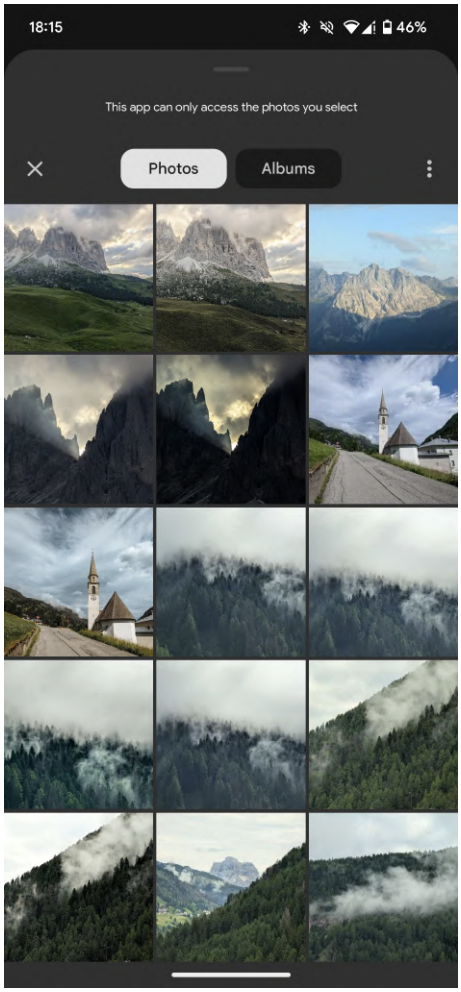
Po wyborze zdjęcia otworzy się widok transformacji zdjęcia, w którym można dokonać przekształceń takich jak:

— przycięcie zdjęcia
— obrót zdjęcia
— Odbicie lustrzane w pionie
— odbicie lustrzane w poziomie

W celu zatwierdzenia przekształceń i przejścia dalej należy kliknąć przycisk *CROP* w prawym górnym rogu. Widok ten przedstawiono na 9.3.

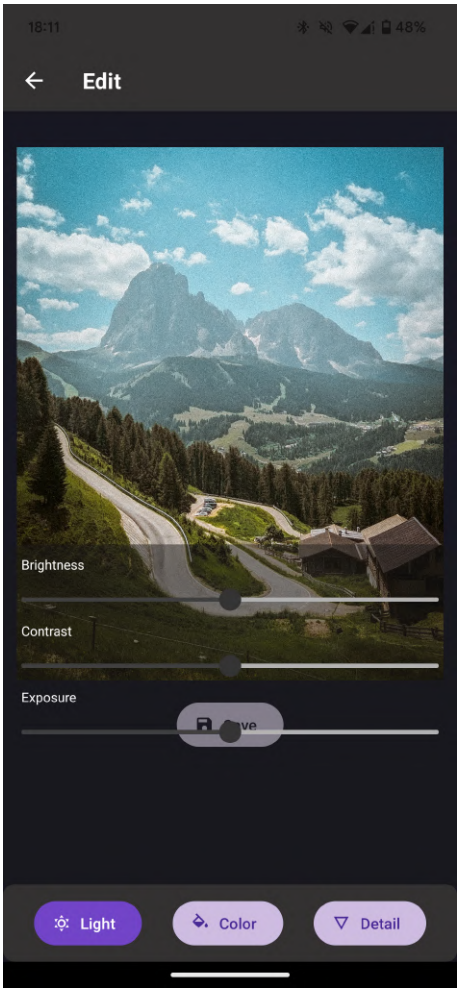


Rys. 9.1. Ekran startowy aplikacji.



Rys. 9.2. Modal wyboru zdjęcia.

Rys. 9.3. Modal przycinania oraz obracania zdjęcia.



Rys. 9.4. Ekran edycji zdjęcia.

9.4. WIDOK EDYCJI ZDJĘCIA

Po zatwierdzeniu przekształceń transformacji zdjęcia, użytkownik trafia do głównego widoku aplikacji 9.4 w którym dokonywane są pozostałe przekształcenia w ramach edycji zdjęcia. W górnej części ekranu wyświetla się edytowane zdjęcie. Pod nim natomiast znajduje się przycisk *Save* służący do zapisu zdjęcia. Na dole ekranu widoczne jest menu zawierające trzy kategorie filtrów. Są to kolejno *Light* - filtry światła, *Color* - filtry koloru oraz *Detail* - filtry szczegółu.

Kliknięci wybranego elementu z dolnego menu spowoduje otwarcie półprzeźroczystego menu z suwakami odpowiedzialnymi za zmianę wartości danego filtru. Jego nazwa podana jest nad suwakiem po lewej jego stronie. Każda zmiana suwaka powoduje automatyczne odświeżenie zdjęcia do nowej, zaktualizowanej o zmienianą wartość wersji. Nad każdym zmienionym filtrem, na wysokości jego nazwy, po prawej stronie pojawi się symbol *restart*. Kliknięcie go spowoduje przywrócenie filtru do startowej - domyślnej - wartości.

W celu zapisania zdjęcia należy kliknąć przycisk *Save* znajdujący się pod nim.
U góry ekranu pojawi się powiadomienie o sukcesie dokonanej operacji, a zapisane zdjęcie dostępne będzie w galerii urządzenie. Użytkownik może kontynuować edycję i zapisać kolejną kopię zdjęcia.

10. PODSUMOWANIE PRACY

10.1. WNIOSKI

Celem projektu było stworzenie mobilnej aplikacji do edycji zdjęć na platformie Android. Użytkownik aplikacji miał mieć możliwość wyboru zdjęcia z galerii urządzenia, a następnie wykonania wybranych operacji przekształceń. Celem przekształceń dokonywanych w aplikacji miała być subiektywna poprawa odbioru wizualnego edytowanego zdjęcia. W ramach przekształceń zaimplementowane miały być cztery główne grupy:

— transformacja zdjęcia - przycięcie, obrót, lustrzane odbicie
— edycja światła - zmiana jasności, luminacji oraz kontrastu
— edycja koloru - saturacja, odcień oraz temperatura
— edycja detalu - filtr wyostrzający

Największym wyzwaniem było zapewnienie płynnego działania aplikacji. Edycja zdjęć jest procesem wymagającym wielu obliczeń. Proces przekształcania zdjęcia polega na zmianie wartości każdego piksela o wyznaczoną wartość. Jest to zadanie dające się zrównoleglić i wykonać na karcie graficznej urządzenia. Zastosowane technologia do tworzenia aplikacji mobilnych, React Native, nie jest jednak typowym rozwiązaniem dla wymagających obliczeniowo problemów. Ze względu na to istnieje niewiele źródeł traktujących o temacie edycji zdjęć w wybranych do realizacji projektu technologiach. Pierwszym krokiem było zatem zgłębienie specyfikacji React Native oraz frameworka Expo w celu znalezienia wydajnego rozwiązania problemu. Zdecydowano się zastosować WebGL poprzez wykorzystanie biblioteki `expo-gl` pozwalające na stworzenie kontekstu WebGL w ramach komponentu React Native. WebGL natomiast zapewnił pomost pomiędzy kodem TypeScript, oraz kartą graficzną urządzenia. Z wykorzystaniem języka shaderów OpenGL ES - GLSL - napisane zostały shadery poszczególnych przekształceń zdjęcia. W celu poprawnej ich implementacji dokonano przeglądu matematycznych podstaw przekształcenia obrazy oraz zmian w przestrzeni barw niezbędnych do ich realizacji. Aplikacja udowadnia, że wybrany stos technologiczny może być skutecznie wykorzystywany do edycji oraz przekształceń zdjęć. Ponadto pozwala on na podejście wykorzystujące natywne elementy systemu operacyjnego takie jak widget wyboru zdjęcia.

Zrealizowana aplikacja spełnia zarówno określone wcześniej wymagania funkcjonalne jak i niefunkcjonalne określone w rozdziale 3. Posiada ona również szerokie możliwości na dalszy rozwój, których przegląd jest przedmiotem kolejnego podrozdziału.

10.2. DALSZY ROZWÓJ APLIKACJI

Aplikacja może być rozwijana w następujących dziedzinach

— Filtry - aplikacja może wspierać większą ilość filtrów przekształcenia obrazu. W szczególności możliwe jest dodanie bardziej zaawansowanych filtrów wyostrzania, zmian tonalnych kolorów oraz innych narzędzi stosowanych w profesjonalnej edycji zdjęć.
— Wsparcie nowych typów plików - obecnie aplikacja wspiera pliki jpg, lecz można ją rozszerzyć o wsparcie plików z przeźroczystością — kanałem Alpha. Ponadto można zaimplementować wsparcie dla typów RAW.
— Sztuczna inteligencja - działanie aplikacji może zostać rozszerzone o wsparcie sztucznej inteligencji w celu ułatwienia i usprawnienia edycji zdjęć
— Profile - można rozwinąć dostępne funkcjonalności o profile, czyli zestawy filtrów o ustalonych parametrach, które użytkownicy mogliby tworzyć i zapisywać w celu ponownego użycia

Ponadto aplikacja, po spełnieniu odpowiednich wymagań, może zostać dystrybuowana na platformie Sklep Play i dostępna dla jej użytkowników do pobrania na urządzenia z systemem Android.

BIBLIOGRAFIA

[1] *Create your first build - learn how to create a build for your app with EAS Build.*, https://docs.expo.dev/build/setup/. Ostatnie wykorzystanie 13.11.2024.

[2] *Expo glview - expo-gl*, https://docs.expo.dev/versions/latest/sdk/gl-view/. Ostatnie wykorzystanie 13.11.2024.

[3] *Frame rate - wikipedia*, https://en.wikipedia.org/wiki/Frame_rate. Ostatnie wykorzystanie 6.12.2024.

[4] *Get started with react native*, https://reactnative.dev/docs/environment-setup. Ostatnie wykorzystanie 13.11.2024.

[5] *Hsv color representation*, https://upload.wikimedia.org/wikipedia/commons/thumb/4/4e/HSV_color_solid_cylinder.png/800px-HSV_color_solid_cylinder.png?20151228061406. Ostatnie wykorzystanie 25.11.2024.

[6] *Material Design — dokumnetacja*, https://m2.material.io/design/introduction#principles. Ostatnie wykorzystanie 10.11.2024.

[7] *React native - npm*, https://www.npmjs.com/package/react-native. Ostatnie wykorzystanie 13.11.2024.

[8] *Rgb color representation*, https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/RGB_color_solid_cube.png/800px-RGB_color_solid_cube.png. Ostatnie wykorzystanie 25.11.2024.

[9] *Web content accessibility guidelines (wcag) 2.1*, https://www.w3.org/TR/WCAG21/#dfn-relative-luminance. Ostatnie wykorzystanie 25.11.2024.

[10] Ahronson, K., *Which editing software is the most popular?*, https://phototeacher.blog/2023/04/03/which-editing-software-is-the-most-popular/. Ostatnie wykorzystanie 25.11.2024.

[11] Andrea, J.C., *Figma Design for Beginners and Seniors: A complete UI/UX* (O'Reilly Media Inc., 2022).

[12] Andrzej, S., *Android Studio. Podstawy tworzenia aplikacji* (Helion, 2015).

[13] Asano, S., Maruyama, T., Yamaguchi, Y., *Performance comparison of fpga, gpu and cpu in image processing*, w: *2009 International Conference on Field Programmable Logic and Applications* (2009), str. 126–131.

[14] Ceci, L., *Leading photo and video editor apps in the united states in july 2024, by downloads*, https://www.statista.com/statistics/1350996/top-photo-editor-apps-us-by-download/. Ostatnie wykorzystanie 25.11.2024.

[15] Contributors, E., *Dokumentacja expo*, https://docs.expo.dev/. Ostatnie wykorzystanie 21.10.2024.

[16] Contributors, R.N., *Dokumentacja react native*, https://reactnative.dev/docs/getting-started. Ostatnie wykorzystanie 21.10.2024.

[17] CredenceResearch, *Photo editor app market by type (entry level, prosumer level, professional level); by platform (macOS, Windows, Android, iOS); by end user (artist/individuals, enterprise users, educational users); by region – growth, share, opportunities competitive analysis, 2024 – 2032 - raport preview*, https://www.credenceresearch.com/report/photo-editor-app-market. Ostatnie wykorzystanie 21.10.2024.

[18] Dabit, N., *React Native in Action* (Manning, 2019).

[19] Dan Ginsburg, Budirijanto Purnomo, D.S., *OpenGL Es 3.0 Programming Guide* (Addison Wesley, 2014).

[20] Eisenman, B., *Learning React Native - building native mobile apps with JavaScript* (O'Reilly Media Inc., 2016).

[21] Goldberg, J., *Learning TypeScript: Enhance Your Web Development Skills Using Type-Safe JavaScript* (O'Reilly Media Inc., 2022).

[22] John C. Russ, F.B.N., *The Image Processing Handbook* (CRC Press, 2017).

[23] Moedano, B., *Expo go vs development builds: Which should you use?*, https://expo.dev/blog/expo-go-vs-development-builds. Ostatnie wykorzystanie 13.11.2024.

[24] React Native Paper Contributors, *Dokumentacja React Native Paper*, https://callstack.github.io/react-native-paper/. Ostatnie wykorzystanie 21.10.2024.

[25] Parisi, T., *Aplikacje 3D. Przewodnik po HTML5, WebGL i CSS3* (Helion, 2013).

[26] Rafael C. Gonzalez, R.E.W., *Digital Image Processing* (Pearson, 2018).

[27] Rahman, A., *React native — ultimate guide on new architecture in depth*, https://github.com/anisurrahman072/React-Native-Advanced-Guide/blob/master/New-Architecture/New-Architecture-in-depth.md#old-architecture-quick-overview. Ostatnie wykorzystanie 5.11.2024.

[28] Richter, F., *Smartphone users have a soft spot for photo editing apps*, https://www.statista.com/chart/28913/share-of-smartphone-users-who-regularly-use-camera-and-photo-editing-apps/. Ostatnie wykorzystanie 21.10.2024.

[29] Saha, S., *Photo editor app market outlook for 2024 to 2034 - raport preview*, https://www.futuremarketinsights.com/reports/photo-editor-app-market. Ostatnie wykorzystanie 21.10.2024.

[30] Team, T.R., *New architecture is here*, https://reactnative.dev/blog/2024/10/23/the-new-architecture-is-here. Ostatnie wykorzystanie 5.11.2024.

SPIS RYSUNKÓW

SPIS LISTINGÓW