МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Секція інформаційно-комунікаційних технологій

ВИПУСКНА РОБОТА

на тему:

«Мобільний додаток каталог-помічник для вивчення програмування та систематизації набутих знань для операційних систем IOS та Android»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Шутилєва О.В.**

Студент гр. ІН-71 **Гирявенко Д. Р.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедри Довбиш А.С.

"_____" _____2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-71 спеціальності «122 – Комп'ютерні науки» денної форми навчання Гирявенка Дмитрія Романовича

Тема: «Мобільний додаток каталог-помічник для вивчення програмування та систематизації набутих знань для операційних систем IOS та Android»

Затверджена наказом по СумДУ

№_____ від _____2021 р.

Зміст пояснювальної записки: 1) огляд та аналіз існуючих аналогів; 2) постановка завдання й формулювання завдань дослідження; 3)проектування інформаційної системи та бази даних 3) огляд технологій для реалізації завдання; 4) розробка мобільного додатку; 5) аналіз результату.

Дата видачі завдання "_____"_____2020 р.

Керівник випускної роботи _____ Шутилєва О.В.

Завдання прийняв до виконання _____ Гирявенко Д. Р.

РЕФЕРАТ

Записка: 60 стор., 26 рис., 1 додаток, 14 джерел.

Об’єкт дослідження – мобільні додатки для вивчення мов програмування.

Мета роботи – розробка мобільного каталогу-помічник для вивчення програмування та систематизації набутих знань для операційних систем IOS та Android.

Методи дослідження – метод порівняння, узагальнення та моделювання.

Результати – розроблено мобільного каталогу-помічник для вивчення програмування та систематизації набутих знань. Додаток створено на базі мови програмування JavaScript з використанням фреймворку React Native та технологіями Redux, Firebase API.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ВНЗ – Вищий навчальний заклад

ОС – Операційна система

БД – База даних

ТЗ – Технічне завдання

ГІ – Графічний інтерфейс

UX – досвід користувача

RN – React Native

ІС – Інформаційна система

API – Application Programming Interface

MVVM – Model-View-View-Model

DFD – Data Flow Diagrams

SADT – Structured analysis and design technique

ІСР – Інтегроване середовище розробки

ВСТУП

Програмування – один з найцінніших навичок для кар'єрного зростання, саморозвитку. Дослідження проведене Stack Overflow у 2019 році показало, що понад 85% [1] розробників вивчають нові мови програмування, фреймворки та інструменти самостійно (без проходження офіційних курсів).

Однак, самоосвіта не завжди є найефективнішим способом отримати нові знання. Головна проблема це відсутність системного підходу. Важко зрозуміти, що вивчати в першу чергу, а що в другу, і в якому порядку. У додаток, тяжко знайти дійсно корисну інформацію в зручній формі та в одному місці. Всі ці проблеми негативно пливають на мотивацію та зацікавленість. Багато людей кидають навчання. Саме з такими помилками зіштовхнувся і я під час вивчення комп'ютерних наук (програмування). Рішенням даної проблеми може слугувати єдиний інформаційний довідник про сучасні мови програмування з можливістю систематизувати набуті знання.

Революція мобільних технологій, розповсюдження інтернету та зростання популярності планшетів вже позначаються на наших щоденних звичках, зокрема і в освіті. Мобільне навчання в даний час дуже поширене і відоме як mLearning. Воно являє собою спосіб отримати доступ до різноманітного онлайн-контенту за допомогою мобільного телефону.

Мобільних додатків для вивчення мов програмування стає все більше. Однак єдиної саме інформаційної платформи, яка б вирішували основну проблему самоосвіти, а саме: структурності та системного підходу до навчання немає.

Актуальність мобільного додатку полягає в створенні першої інформаційної платформи по вивченню мов програмування, яка допоможе отримати в одному місці актуальну, систематизовану та корисну інформацію про бажану мову програмування, та систематизувати набуті знання.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Огляд останніх досліджень і публікацій

У сучасному світі програмування є не такою вже і простою справою. Щоб стати програмістом, потрібно бути не тільки розумним, але ще терплячим і настирливим. Навчання в даній сфері завжди супроводжується головою болем, червоними від недосипання очима і відчуженим поглядом.

Велика кількість студентів кидають навчання вже через кілька тижнів. І основною причиною цього є неправильно обраний напрям навчання, методика або навіть підручник з програмування.

Довгий час Вузи навіть і не намагалися перелаштувати свою систему навчання під потреби сучасного ринку. Лише деякі навчальні заклади, відчувши віяння нового часу почали перекроювати своє навчання під світові стандарти.

Спеціалізовані комерційні курси і навчальні заклади з'явилися не так давно. Спочатку якість викладання і даються в них знань через відсутністю досвіду, перебували на вкрай низькому рівні. Не вистачало грамотних фахівців, здатних навчити новачків не тільки теоретичних знань, але і практичним навичкам програмування. А це в професії програміста є найбільш важливим аспектом. Тому більшість із сьогоднішніх гуру російської IT-індустрії починали своє навчання програмуванню з нуля самостійно. Хоча в наш час кількість професіоналів, що займаються викладанням, помітно зросла. З'являються якісні курси і навіть приватні університети. Але тенденція самоосвіти в даній сфері зберігається і донині.

У своїй статті [2] “6 проблем, з якими ви зіткнетеся, вивчаючи програмування самостійно” професор Гарвардського університету Xiang Zhou описав 6 основних проблем під час вивчення. Авто вважає відсутність мотивації, пошук оптимальної кількості часу для програмування є основними проблемами.

Автор статті, представник комп'ютерної академії "ШАГ" [3] вважає що саме "мапа", або ж план вивчення є найважливішим елементом успішного вивчення мов програмування.

1.2. Аналіз програмних продуктів-аналогів

Існує безліч програм які дозволяють користувачам вивчати та практикувати мови програмування. Також існує велика кількість web курсів для вивчення бажаної мови програмування. Однак інформаційних порталів які агрегують ресурси для вивчення програмування невелика кількість.

Після вивчення ринку технологій і пошуку схожого програмного забезпечення для вивчення мов програмування були розглянуті наступні програми: Datacamp:Learn Data Science, SoloLearn: Learn to Code, Mimo: научись программировать, Grassshopper: Learcn to Code.

Все програмне забезпечення було інстальовано з магазинів додатків App Store для операційної системи IOS, та Google Play Store для операційної системи Android.

Datacamp: Learn Data Science

Користувач має можливість обрати курс. Метод навчання оснований на покроковому виконанні завдань. За кожне виконання завдання користувач отримує «очки» досвіду. За ці «очки» можна придбати підказки до завдань. Всі завдання добре структуровані та поділені на логічні блоки. Кожне завдання побудоване наступним чином: спочатку користувач має можливість ознайомитися з теоретичною частиною завдання, наступним кроком закріпити набуту інформацію на практиці. В додатку відсутня українська та російська мови. До деяких завдань присутні відео пояснення. Додаток має досить зручний UX та красивий користувацький інтерфейс. Присутні push-повідомлення.

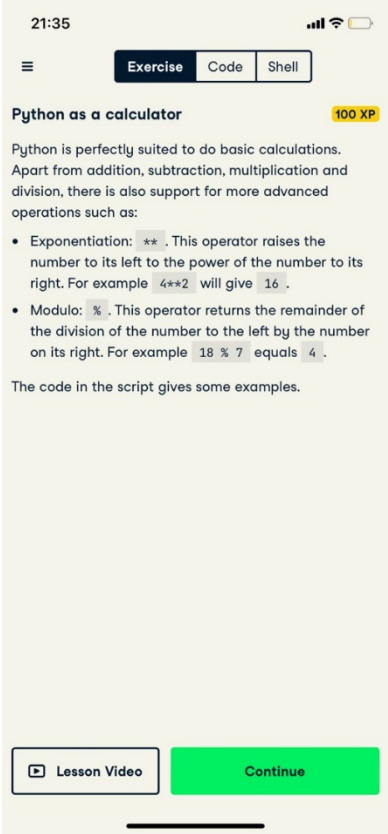


Рисунок 1 – Інтерфейс додатку Datacamp: Learn Data Science

SoloLearn: Learn to Code

Присутнє опитування яке дає більш точну інформацію про рівень підготовки користувача. Програма генерує персональний курс. Важливою перевагою доданку є можливість переглянути відео пояснення. Присутня функція під назвою “Співтовариство”, де користувачі мають можливість обговорювати різного роду питання зв’язані з обраною мовою програмування. Присутній вбудований інтерпретатор. В доданок, цікавим функціоналом є змагання з програмування, користувач обирає опонента та вирішує задачі з програмування на швидкість. Хто швидше виконає всі завдання – переміг. Даний функціонал додає мотивації вивчати мови більш досконало та в ігровій формі. Присутні push-повідомлення.

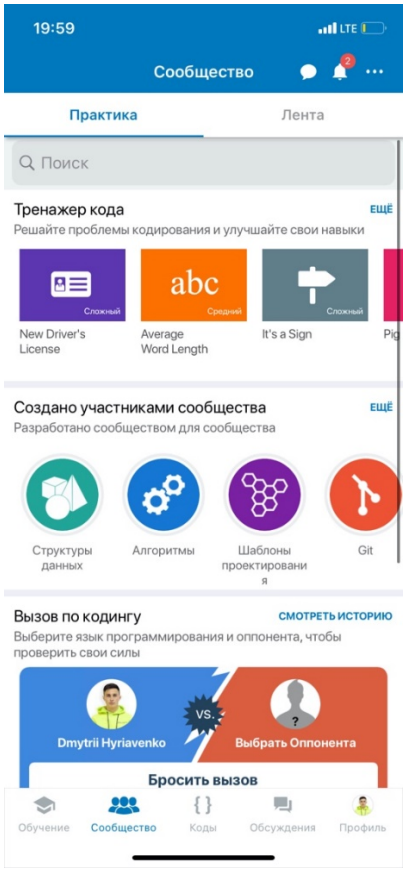


Рисунок 2 – Інтерфейс додатку SoloLearn: Learn to Code

Mimo: научись программировать

Присутнє опитування яке дає більш точну інформацію про рівень підготовки користувача. Програма генерує персональний курс. Завдання поділені на уроки. Кожне завдання є тестом з декількома варіантами. Навчання є геймифіковане. Присутня система життів. При помилці в завданні, система забирає життя. Поповнити запас можливо через деякий час автоматично або за монети, які можна отримати за успішно виконані завдання. Для користувача існує можливість обрати годину щоденного нагадування, це дає більшу організований процес навчання. Присутня система внутрішніх подарунків за успішно виконані завдання. Присутня таблиця лідерів поділених на різні ліги(в залежності від кількості «очків» досвіду). Присутній вбудований інтерпретатор наступних мов програмування: SQL, Python, HTML. Додаток має досить зручний UX та красивий користувацький інтерфейс. Присутні push-повідомлення.

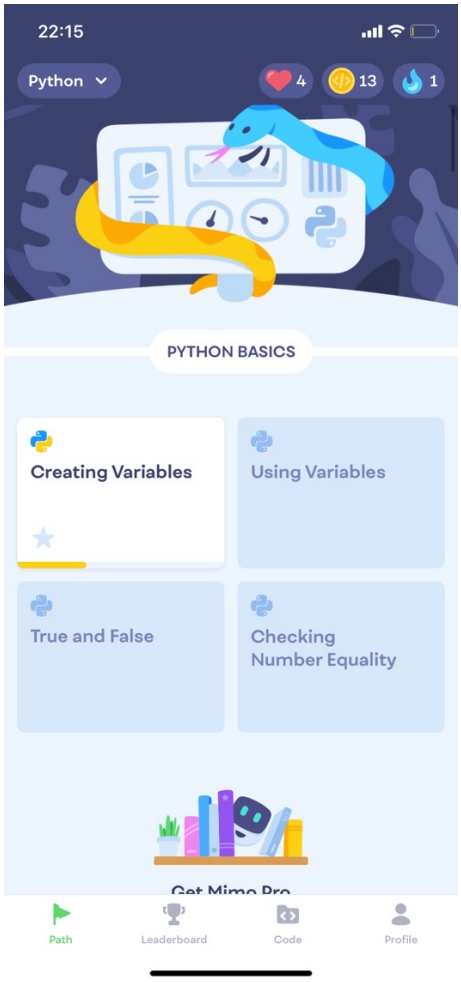


Рисунок 3 – Інтерфейс додатку Mimo: научись программировать

Grasshopper: Learn Data Science

Присутнє опитування яке дає більш точну інформацію про рівень підготовки користувача. Програма генерує персональний курс. Після закінчення курсу, користувач отримує віртуальний сертифікат. Кожне завдання є тестом з декількома варіантами. Всі завдання структуровані на логічні блоки. Проходження завдань відбувається за чітко встановленим планом. Надзвичайно простий у користуванні та привабливий інтерфейс. Присутні push-повідомлення.

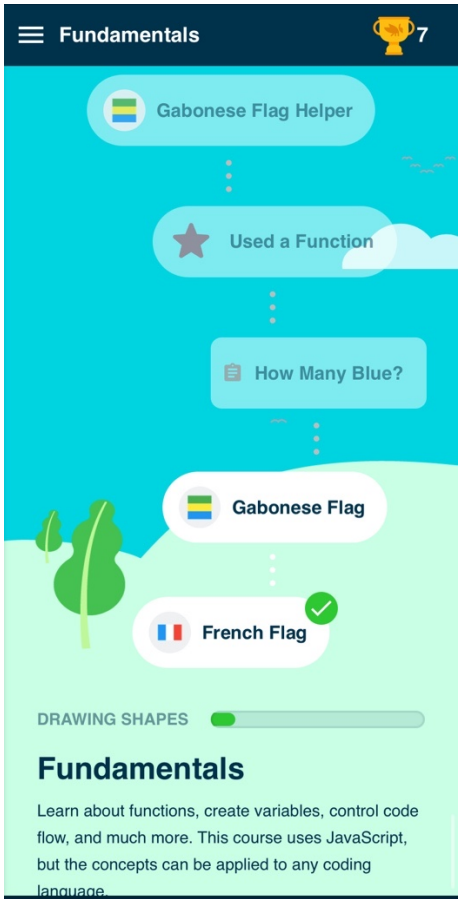


Рисунок 4 – Інтерфейс додатку Grasshopper: Learn Data Science

Було проведено огляд існуючих програмних продуктів, що можуть бути використані для вирішення поставлених задач. Всі додатки мають надзвичайно простий в користуванні та привабливий інтерфейс. За функціональністю всі варіанти дуже близькі, з невеликою різницею в підходах.

Однак, проаналізовані доданки спрямовані на суто практичне вивчення мови програмування, однак обраний за тему дипломної роботи додаток має на меті не дати саме практичний досвід вивчення, а надати місце де користувач зможе в зручній формі в одному мисці знайти необхідні матеріали різного типу, а саме статті, книги, відео, курси для вивчення різних мов програмування.

1.3. Постановка задачі

На основі аналізу предметної області, визначивши основну проблему самоосвіти у людей, котрі вивчають мови програмування, проаналізувавши програмні продукти-аналоги було наступні завдання:

1. Провести проектування інформаційної системи додатку, а саме:
   - Визначити область застосування та найменування додатку;
   - Визначити функціональні та нефункціональні вимоги до доданку;
   - Побудувати функціональну модель;
   - Побудувати діаграму потоків даних;
   - Побудувати діаграму варіантів використання доданку;
   - Побудувати діаграму станів акторів.
2. Провести моделювання бази даних, а саме розробити структуру JSON дерева.
3. Обрати інструменти розробки та обґрунтувати вибір;
4. Програмно реалізувати:
   - Клієнтську частину;
   - Серверну частину;
   - Інтегрувати клієнтську та серверну частини.
5. Протестувати додаток
6. Підготувати додаток до публікації в магазини додатків.

2. МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

2.1. Проектування інформаційної системи

2.1.1. Найменування і область застосування

Найменування програмного продукту – мобільний додаток «Dreamio». Даний продукт призначений для того, щоб відображати актуальну інформацію про новини в обраній мові програмування, список навчальних відео, статей, курсів, книг. Даний додаток надає користувачу детальний план вивчення обраної мови програмування.

Користувачами інформаційного ресурсу можуть бути люди, які тільки почали вивчати мови програмування хочуть отримати детальну та структуровану інформацію про мови програмування, так і програмісти вже з досвідом, які бажають дізнатися “фішки” обраної мови або вивчити нові.

2.1.2. Нефункціональні (технічні) вимоги

У результаті аналізу функціональних вимог і огляду аналогічних проектів були сформульовані наступні нефункціональні вимоги: – додаток має бути написано на мові JavaScript під платформу Android та IOS з використанням фреймворку React Native. Версія для планшету, у рамках дипломної роботи, відсутня.

2.1.3. Функціональні вимоги

Додаток для вивчення мов програмування повинен відповідати таким функціональним вимогам: додаток повинен дозволяти користувачеві переглядати з підключення до інтернету необхідну інформацію, а саме: список мов програмування, список статей, відео, курсів та книг. Додаток повинен надавати користувачеві можливість змінювати свої персональні дані, а саме ім'я.

Мобільний додаток повинен надавати користувачу детальний план вивчення обраної мови програмування. Додаток повинен надавати можливість користувачу обрати кінцеву дату вивчення мови програмування.
Додаток повинен надавати можливість користувачу змінити обрану мову програмування.

2.1.4. Побудова функціональної моделі

Розробка інформаційної системи неможлива без її ретельного проектування: вплив цього кроку на наступні етапи життєвого циклу інформаційної системи, в основі якої лежить створювана база даних є досить суттєвим. Перш за все була створена модель SADT.

SADT (Structured analysis and design technique) являє собою сукупність методів, правил і процедур, призначених для побудови функціональної моделі об'єкта будь-якої предметної області. Дана функціональна модель SADT відображає функціональну структуру об'єкта, тобто вироблені їм дії і зв'язку між цими діями. Результатом застосування методології SADT є модель, яка складається з діаграм, фрагментів текстів і глосарію, що мають посилання(зв’язок) один на одного. Модель може мати декілька рівнів деталізації. [4]

Функціональний блок графічно зображується у вигляді прямокутника і уособлює собою деяку конкретну функцію в рамках даної системи. За вимогами стандарту назва кожного функціонального блоку має бути сформульовано в глобальному сенсі.

Кожна з чотирьох сторін функціонального блоку має своє певне значення (роль), при цьому:

- «Управління»(Верхня сторона) - механізм управління операціями процесу.
- «Вхід» (Ліва сторона) - всі сутності, що надходить в процес;
- «Вихід» (Права сторона) – результат процесу;

- «Механізм» (Нижня сторона)- механізм, який використовується для виконання процесу.

Кожен функціональний блок в рамках єдиної даної системи повинен мати свій унікальний ідентифікаційний номер.

На основі аналізу теоретичних відомостей про SADT було реалізовано діаграму роботи інформаційної системи “Мобільний додаток каталог-помічник для вивчення програмування та систематизації набутих знань”(рис.5).

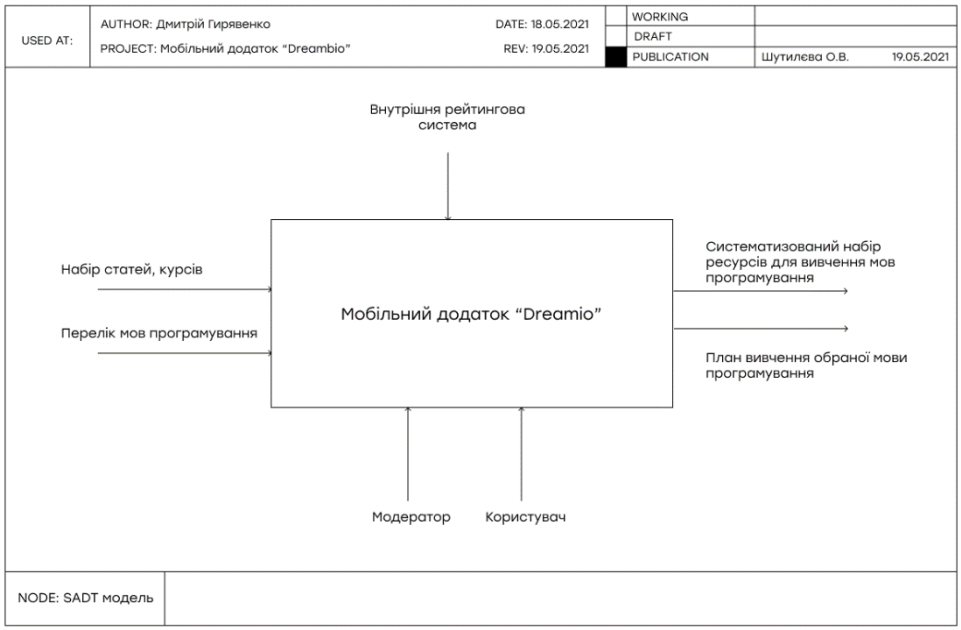


Рисунок 5 – SADT модель

2.1.5. Побудова діаграми потоків даних(DFD діаграма)

DFD моделювання дозволяє представити бізнес-процеси у вигляді формальних процедур описуваних стандартними засобами. Класична DFD показує зовнішні по відношенню до системи джерела і стоки (Адресати) даних, ідентифікує логічні функції (процеси) і групи елементів даних, що зв'язують одну функцію з іншого (потоки), а також ідентифікує сховища (накопичувачі) даних, до яких здійснюється доступ. Структури потоків даних і визначення їх компонент зберігаються і аналізуються в словнику даних. Кожна логічна функція (процес) може бути деталізована за допомогою DFD нижнього рівня.

коли подальша деталізація перестає бути корисною, переходять до вираження логіки функції за допомогою специфікації процесу (міні-специфікації).

На основі аналізу теоретичних відомостей про SADT було реалізовано діаграму роботи інформаційної системи “Мобільний додаток каталог-помічник для вивчення програмування та систематизації набутих знань”(рис.6).

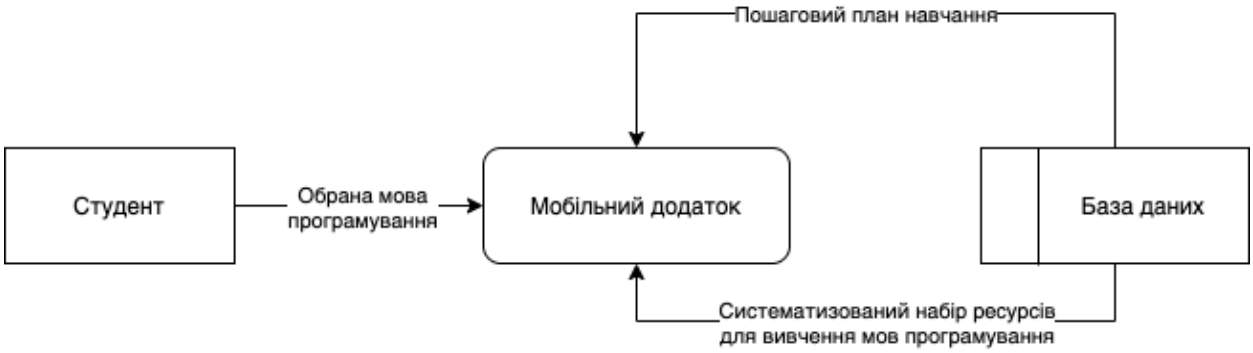


Рисунок 6 – DFD діаграма

2.1.6. Варіанти використання додатку

Діаграма варіантів використання (сценаріїв поведінки, прецедентів) призначена для визначення функціональних вимог до системи та керування всіма процесами розробки інформаційної системи . Проектування та тестування ІС виконуються на даної основі діаграми. Вона дозволяє зрозуміти, як результати, яких хоче досягти користувач цієї системи може вплинути на архітектуру системи та як повинні поводитися її компоненти для реалізації функціональних можливостей, які хоче користувач.

Діаграма використання при визначенні вимог може відповідати на питання "Що користувач очікує від інформаційної системи?", "Що повинна робити система для конкретного користувача?".

Цей підхід дозволяє шукати функції, які потрібні багатьом користувачам, і видаляти, які не можуть допомогти користувачам виконувати важливі завдання.

Дана діаграма складається з акторів, варіантів використання і відносин між ними. В додаток, при побудові використовуються елементи нотації: механізми розширення, примітки і і т.д.

Суть даної діаграми складається в наступному [5]: проектована система представляється у вигляді безлічі акторів, що взаємодіють з системою за допомогою так званих варіантів використання. При цьому актором називається будь-який об'єкт, суб'єкт або система, що взаємодіє з моделюється системою ззовні. У свою чергу варіант використання - це специфікація сервісів (функцій), які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, що здійснюються системою при взаємодії з актором. При цьому в моделі ніяк не відбивається те, яким чином буде реалізовано цей набір дій.

Діаграма варіантів використання мобільного додатка представлена на рис. 7. З додатком взаємодіє два актори - зареєстрований і незареєстрований користувачі.

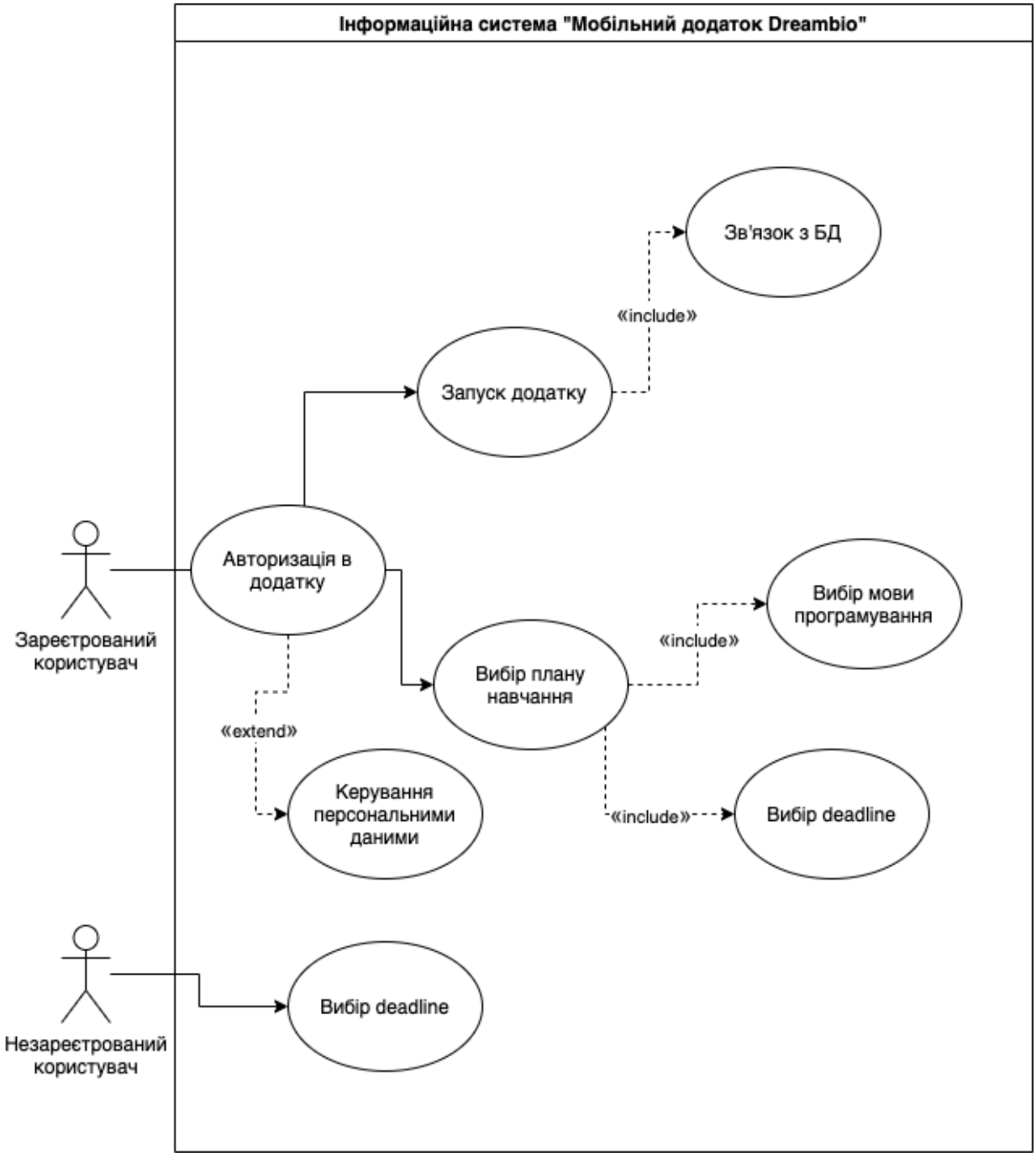


Рисунок 7 – Варіанти використання додатку

Зареєстрований користувач може авторизуватися в додатку через адресу електронної пошти. Без авторизації подальше використання додатка неможливо. Зареєстрований користувач має можливість відновити пароль. Зареєстрований користувач може переглянути список мов програмування та обрати одну для вивчення. Зареєстрований користувач має можливість переглядати статті, відео, курси та книги по обраній мові програмування. Зареєстрований користувач має

можливість переглядати та взаємодіяти з детальним планом вивчення обраної мови програмування. Зареєстрований користувач може визначати кінцеву дату вивчення мови програмування.

Зареєстрований користувач може керувати персональними даними. Незареєстрований користувач може зареєструватися в додатку.

2.1.7. Побудова діаграми станів

Діаграми станів призначені для моделювання різних станів, в яких може перебувати об'єкт (актор). Діаграми станів відображають поведінку об'єкта. На діаграмі є два спеціальних стани: початковий і кінцевий. Діаграма діяльності корисна для опису алгоритму дій, але вона не дає уявлення про поведінку певного об'єкта в рамках окремого випадку використання або системи в цілому, що необхідно при об'єктно-орієнтованому програмуванні.

На сьогоднішній день при проектуванні складної Системи прийнято ділити її на частини, кожну з яких потім розглядати окремо. Таким чином, при об'єктної декомпозиції Система розбивається на об'єкти або компоненти, які взаємодіють один з одним, обмінюючись повідомленнями. Повідомлення описують або представляють собою деякі події. Отримання об'єктом повідомлення активізує його і спонукає виконувати покладені на його програмним кодом дії. При цьому підході Система стає керованою, тому розробникам часто важливо знати, як повинен реагувати той чи інший об'єкт на певні події. Ініціаторами подій можуть бути як об'єкти самої Системи, так і її зовнішнє оточення.

Початковий стан виділяється чорною крапкою: воно відповідає стану об'єкта в момент його створення. Кінцевий стан позначається чорною точкою. Діаграма станів для варіантів використання, пов'язаних з актором «Зареєстрований користувач», показана на рисунку 8.

Рисунок 8 – Варіанти використання додатку для “Зареєстрований користувач” ролі

Діаграма станів для варіантів використання, пов'язаних з актором «Незареєстрований користувач», показана на рисунку 9.

Рисунок 9 – Варіанти використання додатку для “Незареєстрований користувач” ролі

2.1.8. Прототипування

Прототипування – створення моделі або макету який симулює, що представляє фінальний варіант системи. На відміну від технічних завдань, він дає можливість випробувати розробку [6]. На ньому схематично зображуються основні елементи продукту і їх відгук на дії користувача. Одна з основних цінностей прототипування - його продуктивність. У процесі створення

прототипу з'являються сотні, якщо не тисячі ідей [6]. Для прототипування продуктів існує безліч інструментів, однак основним інструментом для створення прототипів було обрано Figma. Даний інструмент не є сервісом суто для прототипування, Figma – комплексний додаток який призначений і для дизайну інтерфейсу, тому при вибору даного варіанту можливо використовувати один єдиний сервіс для прототипування так дизайну. Це досить зручно. Figma працює без офлайн-версії, це крос-платформове рішення , що якісно виділяє його серед найближчих конкурентів. А головне, Figma можна використовувати для прототипування великих і складних проектів і дизайн-систем, що не завжди можливо на інших платформах. Інтерфейс легкий у використанні, зручний та інтуїтивний. Даний інструмент не потребує перенесення прототипу з іншої програми, тому процес дизайну займає набагато менше часу в порівнянні з іншими системами(напр. InVision). В додаток є можливість одночасної онлайн роботи відразу декількох дизайнерів, розробників. Сервіс є безкоштовний для умов розробки прототипу для дипломної роботи. Однак для роботи обов’язково потрібне інтернет-підключення. У додатку Figma було реалізовано прототип додатку(рисунок 10).

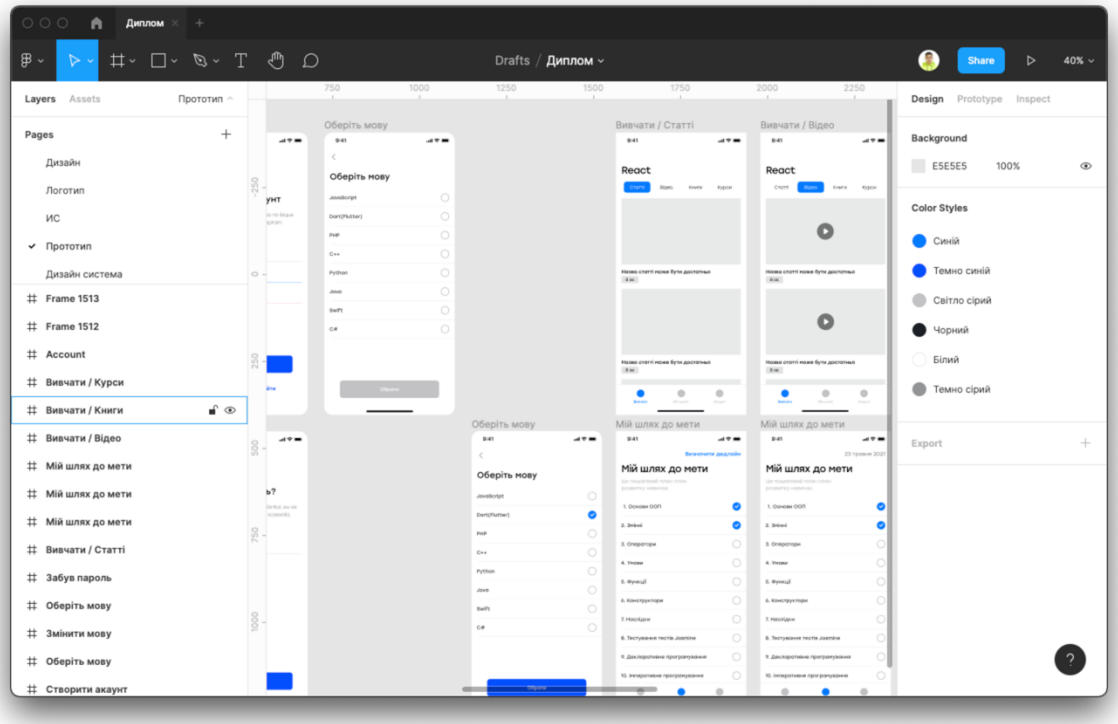


Рисунок 10 – Прототип додатку

Наступним кроком було реалізувати інтерактивний прототип на основі вже існуючого.

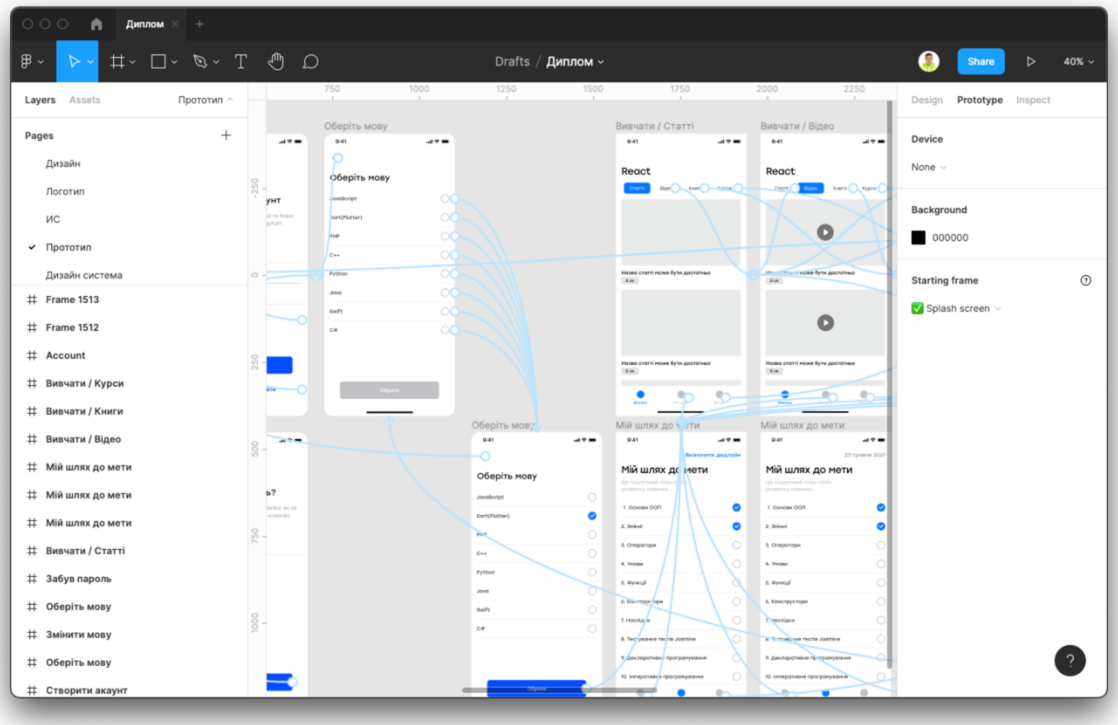


Рисунок 11 – Інтерактивний прототип

Перевагами такого прототипу є зручність пояснення навігації та основних функцій додатку для розробників та замовників.

2.1.9. Розробка інтерфейсу додатку

Розробка інтерфейсу розпочалась з дослідження сучасних тенденцій в дизайні інтерфейсів. Для аналізу було оброблено ряд інтернет-ресурсів:

- www.dribbble.com
- www.behance.net
- www.uplabs.com

Було обрано декілька стильових рішень, на які можна опиратися під час розробки користувацького інтерфейсу.

Основним інструментом для створення прототипів було обрано Figma. Це пз дозволить використовувати і для дизайну інтерфейсу додатку. Figma є найпопулярнішим серед інструментів для дизайну(за даними дослідження) [7], оскільки має велику кількість переваг порівняно з іншими інструменти і практично позбавленні недоліків. Тому саме Figma є найкращим варіантом.

Наступним кроком є розробка дизайн систему для оптимізації подальшої роботи. В дизайн систему ввійшли наступні елементи інтерфейсу:

Перевагою такого підходу є оптимізація часу внесення змін у вже готовий дизайн продукту при необхідності. Створивши дизайн елемент один раз, його можна використовувати в різних місцях. В додаток, змінивши зовнішній вигляд один раз, заміниться всюди де використовується автоматично. Також для дизайн-система корисна для структуризації.

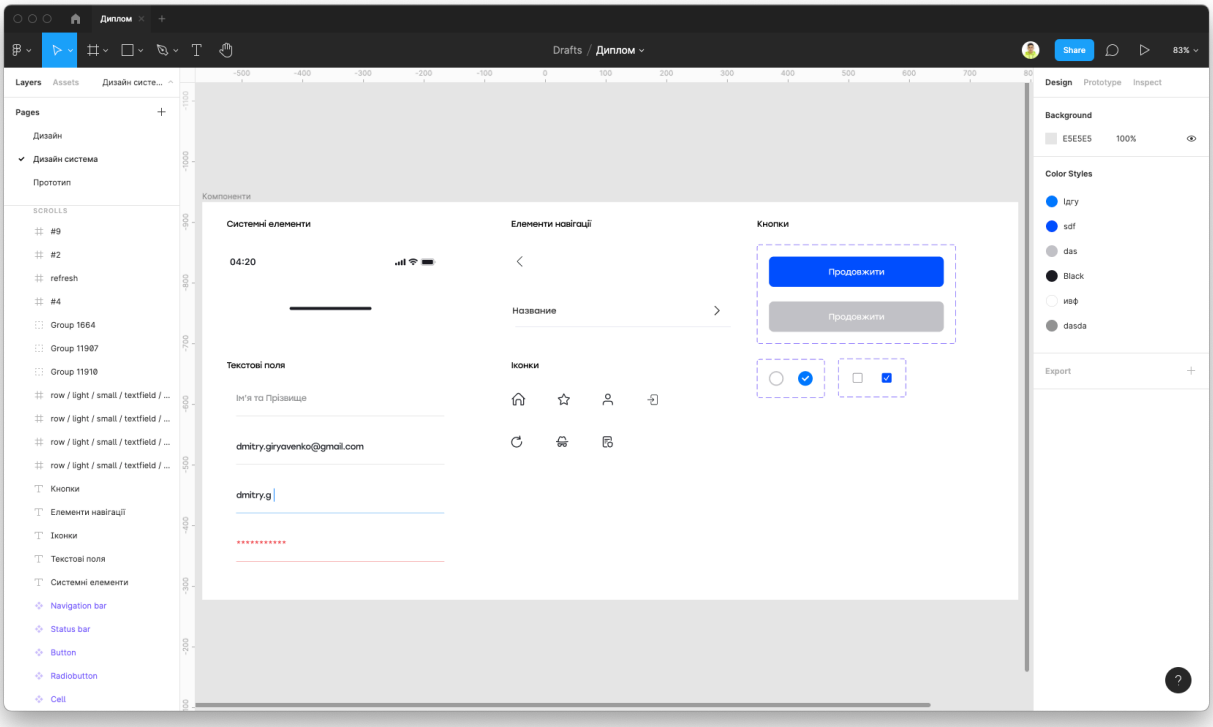


Рисунок 12 – Дизайн-система додатку

Наступним кроком є розробка. Самого інтерфейсу користувача. Перш за все було розроблено Цей та всі інші на основі прототипу та технічного завдання.

На поточному етапі частина тексту використовувалася без смислового навантаження. Дизайн користувацького інтерфейсу було розпочато з створення дизайну авторизації.

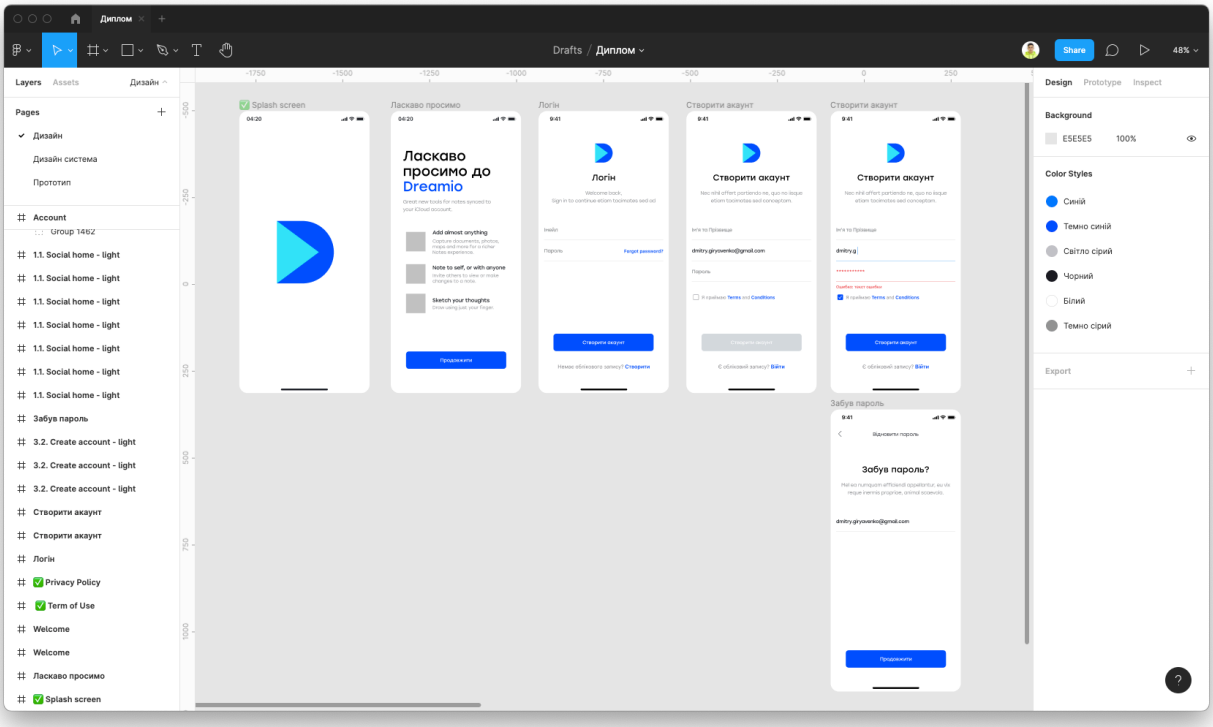


Рисунок 13 – Дизайн екранів авторизації

Наступним кроком є реалізація основного потоку роботи додатку.

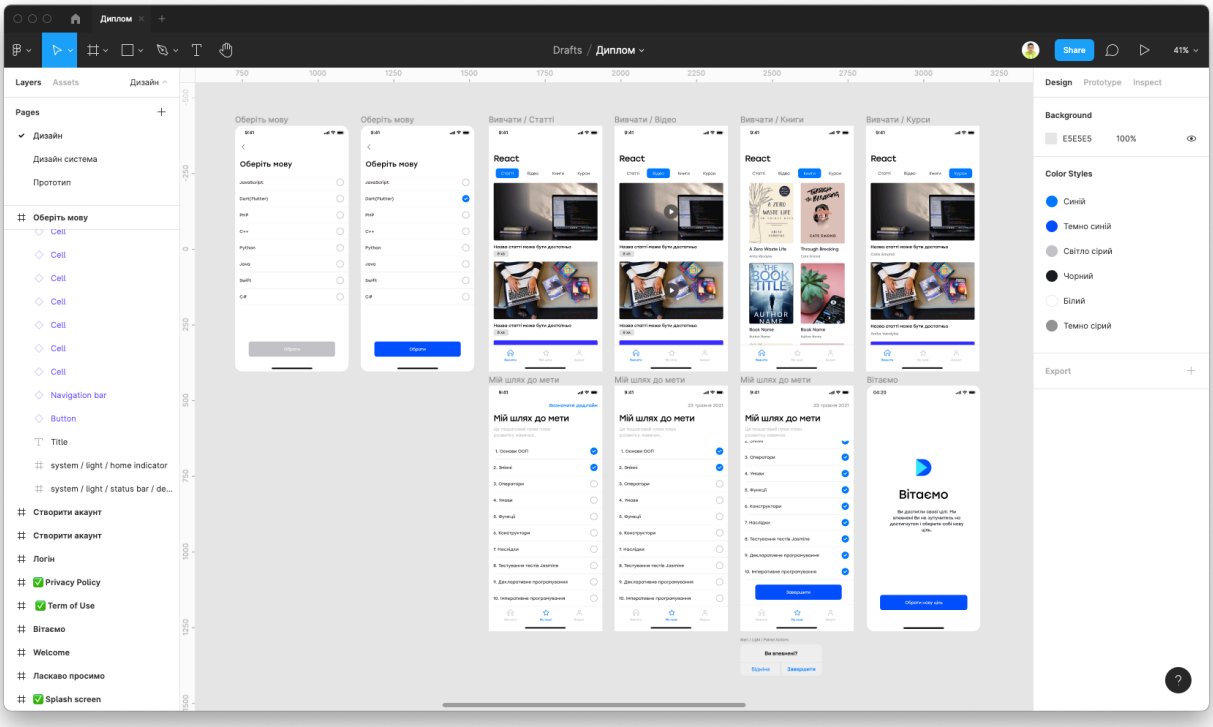


Рисунок 14 – Дизайн основного потоку екранів

Останнім кроком є реалізувати налаштування .

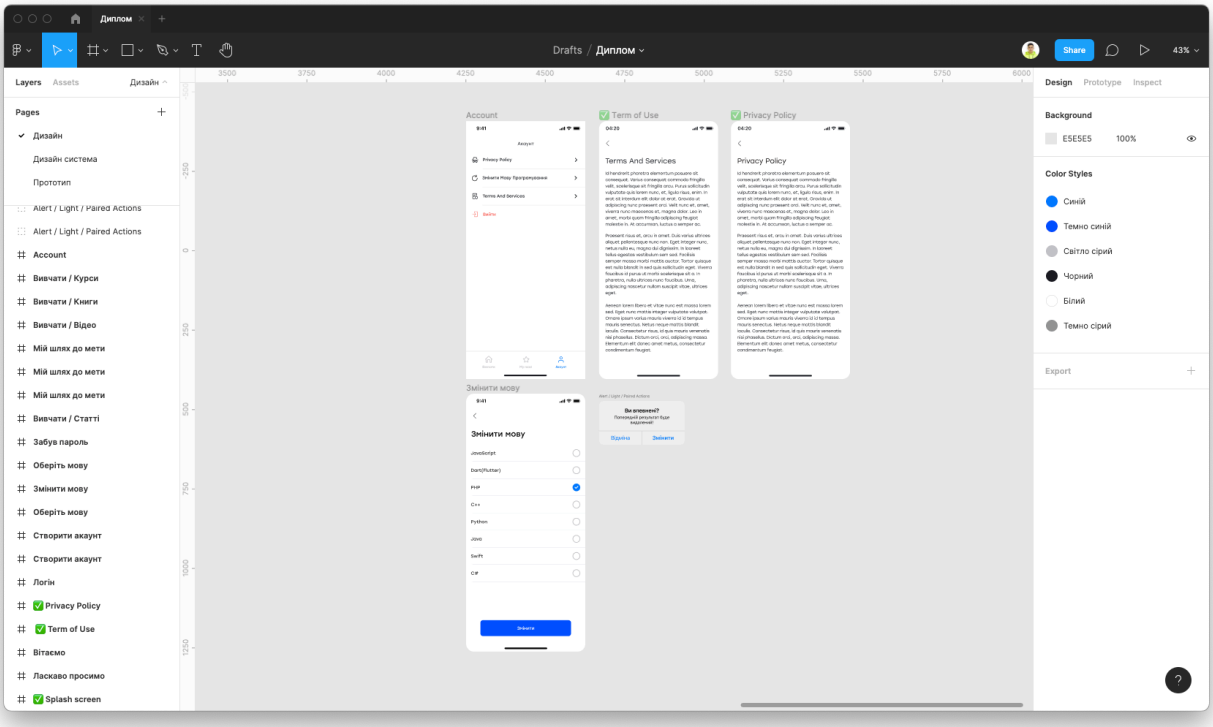


Рисунок 15 – Дизайн екранів налаштування

У додаток було розроблено систему для підтвердження важливих дій користувача за допомогою “alerts view”.

2.1.10. Розробка логотипу додатку

Для розробки логотипу додатку використовувався також інструмент Figma. Спочатку було обрано концепцію та стиль бажаного логотипу. Далі від рук було нарисовано кілька десятків варіантів. Наступним кроком було перенести чернетку в Figma. На основі кольорової палітри додатку було розфарбовано логотип. Фінальний варіант відображений на р 16.

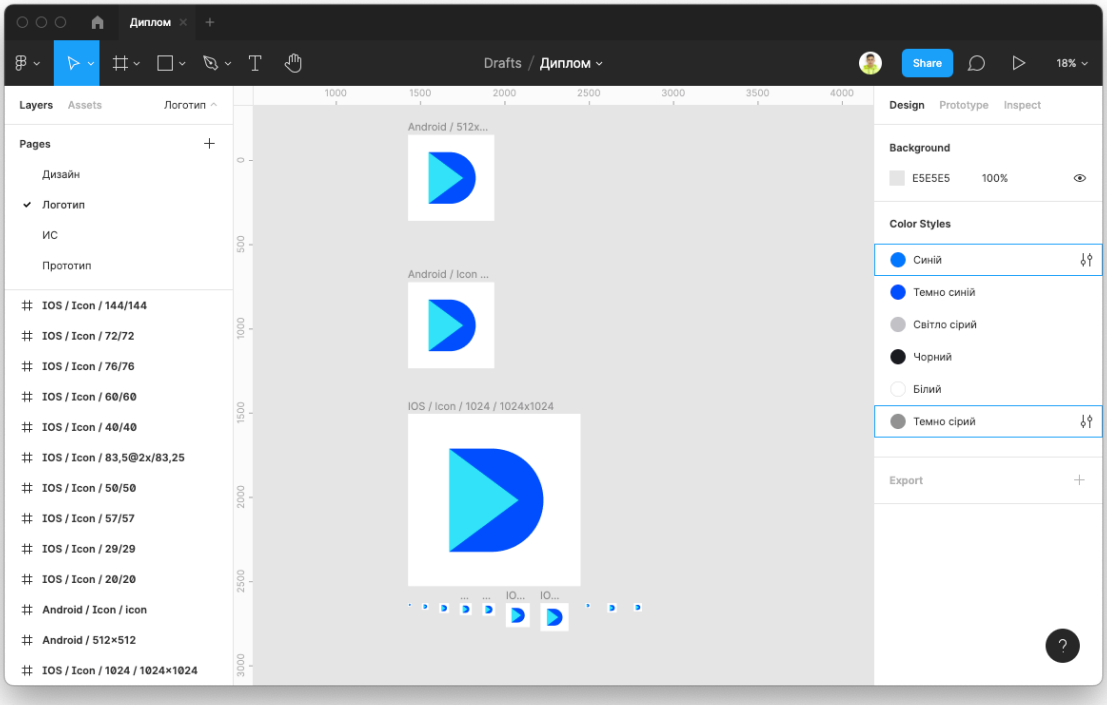


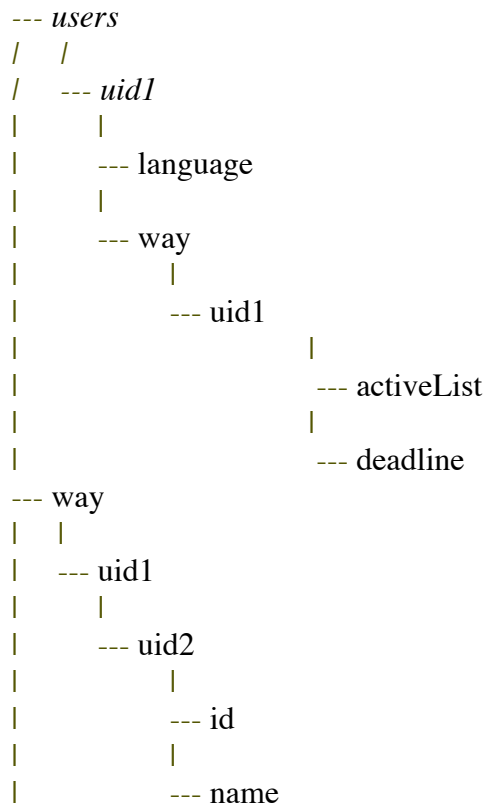
Рисунок 16 – Логотип додатку

2.2. Моделювання моделі бази даних

Для розроблення серверної частини додатку було обрано технологію Firebase. Firebase – ієрархічна база даних тому визначити ERD діаграму (як це роблять для реляційних баз даних) неможливо, тому найкращим варіантом є показати структуру JSON дерева:

```
Firebase-root
  |
  --- languages
  |    |
  |    --- uid1
  |         |
  |         --- id
  |         |
  |         --- name
  |
  --- materials
  |    |
  |    --- articles
  |         |
  |         --- uid1
  |              |
```

```
|              --- img
|              |
|              --- language
|              |
|              --- minutes
|              |
|              --- name
|              |
|              --- url
|
|    --- books
|          |
|          --- uid1
|              |
|              --- img
|              |
|              --- language
|              |
|              --- author
|              |
|              --- name
|              |
|              --- url
|
|    --- courses
|          |
|          --- uid1
|              |
|              --- img
|              |
|              --- language
|              |
|              --- author
|              |
|              --- name
|              |
|              --- url
|
|    --- videos
|          |
|          --- uid1
|              |
|              --- img
|              |
|              --- language
|              |
|              --- minutes
|              |
|              --- name
|              |
|              --- url
|
```



Загалом було спроектовано 4 сутності:

- languages – список мов програмування;
- materials – ресурси для вивчення мов програмування;
- users – список користувачів;
- way – відповідає за шлях користувача під час вивчення мов програмування.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ

3.1. Вибір архітектури клієнтської частини

Для реалізації даної технологічної платформи необхідно було визначитися з ключовим архітектурним рішенням – вибір патерну проектування, який би передбачав високий рівень абстракції, так як це диктується головною особливістю даного проекту – кросплатформове рішення для управління структурою і контентом мобільних клієнтських додатків без необхідності повторного складання. Основні патерни проектування мобільних додатків зазвичай диктуються окремими технологічними платформами.

Основною архітектурою було обрано Flux – це архітектура, відповідальна за створення шару даних в JavaScript додатках і розробку серверної сторони в веб-додатках. Flux доповнює складові компоненти виду View в React, використовуючи односпрямований потік даних.

Flux це більше ніж шаблон, більше ніж фреймворк і має 4 головних компонента:

- Дія (Action) – помічники, які передають дані в Dispatcher;
- Диспетчер (Dispatcher) – отримує ці дії і передає корисне навантаження зареєстрованим “callback”;
- Сховище (Stores) – діють як контейнери для стану програми та логіки. Реальна робота додатка відбувається в Stores. Stores, зареєстровані для прослуховування дій Dispatcher, будуть відповідно і оновлювати View.
- Views (React компонент) – React компоненти захоплюють стан з Stores, а потім передають дочірнім компонентвс. [8]

Це не схоже наприклад на звичний MVC, які часто можна зустріти в інших фреймворків. Контролер в даному підході, який відповідає за графічні компоненти (Views).

Flux слідує концепції односпрямованого потоку даних, що робить його простим для пошуку помилок. Дані проходять через прямий потік вашої програми. React і Flux – на даний момент два найпопулярніших фреймворки, які використовують принцип односпрямованого потоку даних. [9]

У той час як React використовує віртуальний DOM для відображення змін, Flux робить це трохи інакше. У Flux, взаємодія з призначеним для користувача інтерфейсом викликає ряд дій, які можуть змінити дані програми.

Flux має відкритий вихідний код, і це скоріше шаблон проектування, а не фреймворк.

Flux допомагає зробити код більш передбачуваним, в порівнянні наприклад з MVC фреймворками. Розробники можуть створювати додатки, не турбуючись про складні взаємодії між джерелами даних. Flux вигідно відрізняється більш організованим потоком даних - односпрямованим. У Flux всі зміни проходять через один напрямок, через Dispatcher даних. Store не може бути змінено саме по собі, і той же самий принцип працює для інших Actions. Зміни, які необхідно внести, повинні пройти через Dispatcher, через Actions.

Рисунок 17 – Схема роботи Flux архітектури

Основна ідея стоїть за використанням Flux-архітектури – спростити структуру програми. Це полегшує підтримку і розуміння, коли додаток стає

більш складним. Оскільки немає ніякої двозначності у взаємини між різними компонентами, робота стає активною.

Крім того, Flux є послідовним і повторюваним, що робить роботу з ним дуже логічною при створенні Action. Також простіше зробити, щоб Store знав, як обробляти дії, зберігати дані і активувати зміна події.

3.2. Опис інструментів розробки

Під час проектування інформаційної системи було вивчено та проаналізовано предметну область та вимоги замовника. Після ретельного аналізу було вирішено розроблювати програмний продукт за допомогою кросплатформових рішень.

Для реалізації клієнтської частини було обрано технологію React Native, основна мова програмування - JavaScript.

React – це JS-бібліотека для створення користувацьких інтерфейсів, зазвичай для веб-додатків. Вона розроблена в Facebook і поширюється під ліцензією open source з 2013 року. React широко поширена, і на відміну від більш великих MVC-фреймворків вирішує щодо вузьке завдання: рендеринг інтерфейсу.

React дозволяє легко створювати інтерфейси, розділяючи кожну сторінку на невеликі фрагменти і компоненти. Він дуже зручний для створення веб-додатків і не вимагає великого порогу входження. У React Native код пишеться на JavaScript, і здійснюється за допомогою JavaScriptCore. Так само можна використовувати нативні модулі платформи, наприклад камеру або Bluetooth. Для цього пишеться код, який реалізує функціональність на мові, який призначається для розробки під конкретну платформу (Java / Swift / Objective C) і повідомляється з середовищем JavaScript за допомогою bridge.

React Native є найбільш популярним фреймворком для написання кросплатформових додатків. Як показую дослідження [9] проведене всесвітньо відомою дослідницькою агенцією Statista, 42% користувачів використовують

React Native як основний фреймворк для написання крос-платформових додатків.

Популярність React має ряд причин. Вона компактна і має високу продуктивність, особливо при роботі з швидкоплинними даними. Завдяки компонентної структурі, React заохочує до написання модульного, повторно використаного коду.

Для серверної частини було обрано технологію Firebase. Firebase - це хмарна база даних, яка дозволяє користувачам зберігати і отримувати збережену інформацію, а також має зручні засоби і методи взаємодії з нею.

Firebase зберігає текстові дані в JSON форматі і надає зручні методи для читання, оновлення та видалення даних. Також, Firebase має інструменти для реєстрації та авторизації користувачів, зберігати сесії для авторизованих користувачів, зберігати медіа-файли до яких з легкістю можна отримати доступ завдяки Cloud Storage.

Firebase не є повністю безкоштовним інструментом. Частина функціоналу залишається недоступною для тих користувачів, які користуються безкоштовною версією. Але основний функціонал, такий як: реєстрація, авторизація та зберігання тексту доступні в безкоштовному тарифі. Для цілей додатку, безкоштовний план повністю задовольняє всі потреби.

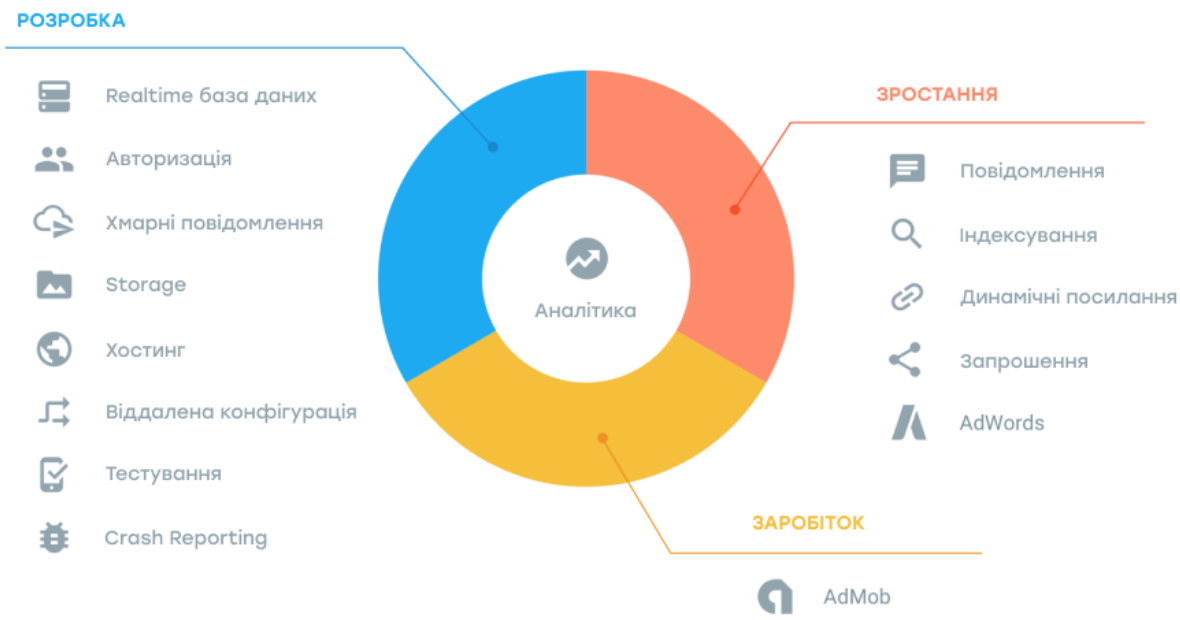


Рисунок 18 – Функціональні можливості Firebase

Firebase – це, по своїй суті, набір інструментів, які розробники можуть використовувати, створюючи додатки в залежності від своєї цілей та потреб.

Мета Firebase полягає в рішенні трьох основних проблем розробників:

- Швидко створити додаток
- Випустити і забезпечити надійний моніторинг працездатності
- Залучити користувачів,

Розробники, що використовують цю платформу, отримують доступ до сервісів, за допомогою яких вони зможуть розробляти свої продукти, і це дозволяє їм зосередитися безпосередньо на досягненні створенні якісної клієнтської частини.

Деякі з найбільш популярних функцій платформи Google Firebase включають в себе бази даних, автентифікацію, push-повідомлення, аналітику, зберігання файлів і багато іншого.

Оскільки сервіси знаходяться в хмарі, розробники можуть поетапно виконувати масштабування своїх продуктів, не відчуваючи ніяких проблем.

Firebase на даний момент входить в число кращих платформ для розробки додатків, яким довіряють розробники по всьому світу.
Основною середою для розробки було обрано Visual Studio Code

3.3. Обґрунтування вибору інструментів розробки

Дані технології в сукупності дають змогу збудувати якісний та надійний продукт, який захищений від патентних позовів з боку розробників, бо всі ці технології покриті ліцензіями, які виключають таку можливість і надають доступ до вихідних кодів даних проектів.

Для реалізації клієнтської частини було обрано технологію React Native, основна мова програмування – JavaScript. Розробляти додаток за допомогою React Native досить швидше та зручніше для розробників, оскільки це фреймворк з відкритим кодом і дозволяє повторно використовувати кодову базу для розробки додатків.

React Native має ряд характеристик, які вплинули на вибір:

Наявність бібліотек і віджетів. React Native присутній на ринку вже більше 5 років і має в своєму розпорядженні велику кількість бібліотек для різних випадків, починаючи з push-повідомлень, відтворення відео і компонентів матеріалів, Firebase ML Kit і інтеграціями Apple HealthKit і Google Fit. Також існує ретельно підібраний список високоякісних сторонніх компонентів, які можна використовувати в своєму додатку. React Native також сильно виграє від того, що він заснований на React і має можливість використовувати бібліотеки, створені для React, наприклад Redux або Axios, або привносити сучасні функції, такі як React Hooks. Швидкість рендерингу React Native дещо повільніша(якщо порівнювати наприклад з Flutter) оскільки використовує рідні віджети платформи (Native Views) і передає події через JavaScript. Це впливає на продуктивність. Також У React Native є проблема з часом запуску, оскільки RN повинен завантажити віртуальну машину JS, а потім JS код. На Android це стає складним завданням.

Мережеві можливості. React Native працює з базовим протоколом http (s) з поліфілії fetch, вбудованою підтримкою WebSockets і багаті клієнти з axios. React Native також зчитувати json за замовчуванням, тому що це теж JavaScript. Це також означає, що можна створювати свої типи за допомогою Open Api.

Медіаможливості. React Native підтримує відтворення і запис як аудіо, так і відео. Можна, наприклад, показувати відео в різних форматах, відображати елементи керування для пошуку, зациклення і тд. Також можна відображати субтитри SRT і VTT для відео за замовчуванням. Цей функціонал поставляється з такими компонентами, як react-native-video, react-native-video-controls і іншими.

Безпека. React Native зав'язаний на js і додаток на його основі містить js bundle. Але його завжди можна витягти і зрозуміти логіку програми, або змінити її. Це достатньо серйозна проблема.

Для серверної частини було обрано технологію Firebase. Дана технологія має ряд характеристик, які вплинули на вибір:

Безкоштовний початковий план. Для того, щоб почати користуватися Firebase не треба нічого платити, вона дозволяє користувачам входити в систему, використовуючи свій обліковий запис Google. Firebase пропонує безкоштовний початковий план, який називається Spark, він має безліч функцій, яких часто вистачає, щоб почати працювати

Швидкість розробки. Firebase, це відмінний варіант для розробки додатків, який може дозволити заощадити час на розробку і скоротити час виходу програм на ринок.

Machine Learning. Firebase йде разом з комплектом машинного навчання зі зрозумілими і легкими у використанні API для використання на мобільних платформах, наприклад, для розпізнавання тексту, розпізнавання осіб, маркування зображень, скануванні штрих-кодів і т.д.

Генерація трафіку. Firebase спрощує індексацію додатків, дозволяючи потенційним користувачам швидше знаходити додаток у Пошуку Google

Моніторинг помилок. Функція Firebase Crashlytics, це фантастичний інструмент для швидкого пошуку і усунення проблем. Firebase може відстежувати що не критичні, так і фатальні помилки, всі звіти генеруються на основі того, як помилки впливають на роботу користувачів.

Створення резервної копії. Firebase забезпечує оптимальну безпеку і доступність даних за рахунок здійснення регулярного резервного копіювання. Додатки захищені від будь-якої можливості втрати даних за рахунок використання функції автоматичного резервного копіювання, яка є на цій платформі.

Популярність. React Native є найбільш популярним фреймворком для написання крос-платформових додатків. Як показую дослідження [9] проведене всесвітньо відомою дослідницькою агенцією Statista, 42% користувачів використовують React Native як основний фреймворк для написання крос-платформових додатків.

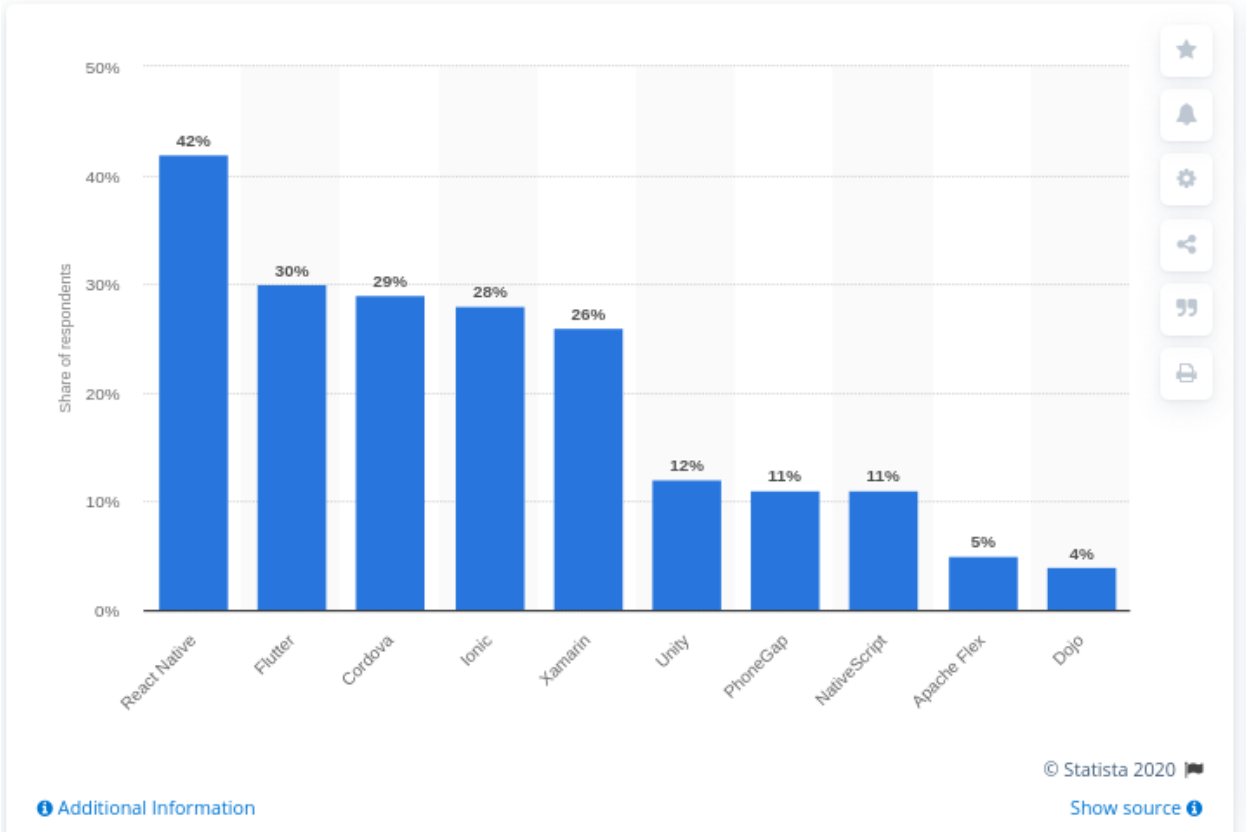


Рисунок 19 – Популярність технологій розробки мобільних додатків

Основним середовищем розробки програмного додатку було обрано Visual Code IDE. Для налаштування параметрів додатку для операційної системи IOS було обрано XCode IDE.

3.4. Програмна реалізація

3.4.1. Клієнтська частини

Першим етапом в створенні клієнтської частини є розгортання проекту в інтегрованому середовищі розробки.

Далі було реалізовано верстку екранів мобільного додатку

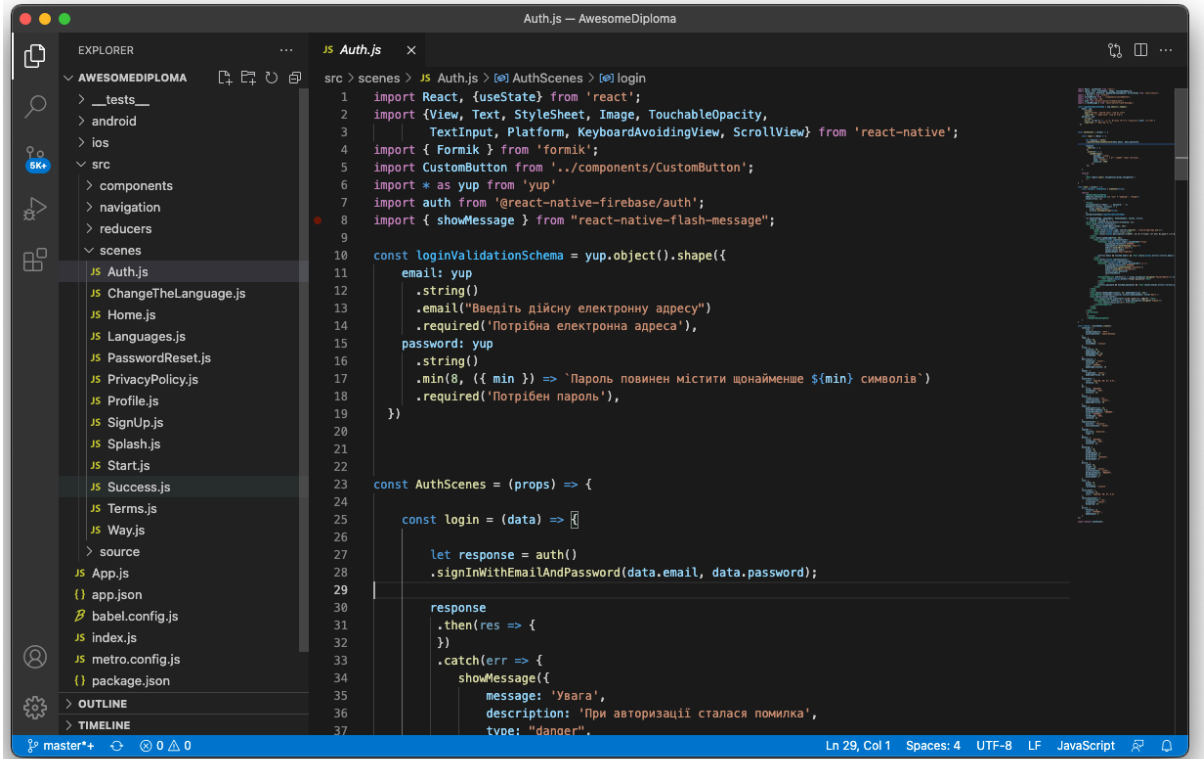


Рисунок 20 – Верстка екранів

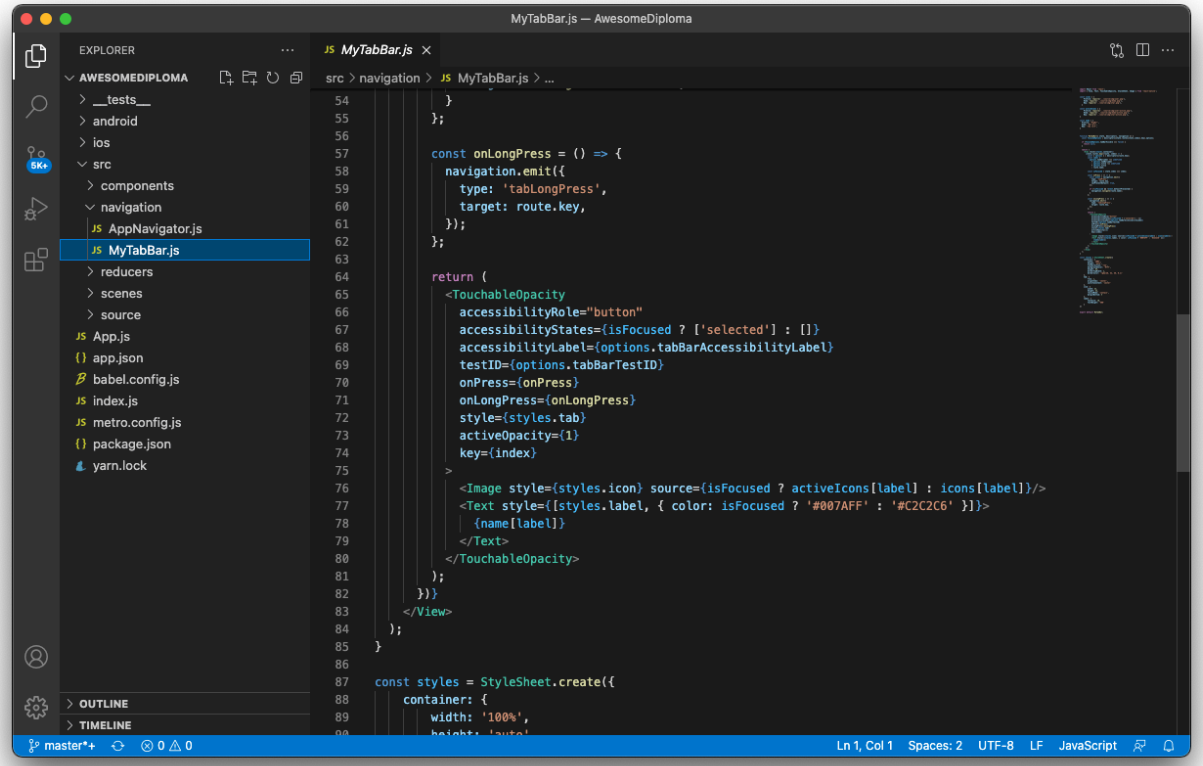


Рисунок 21 – Налаштування навігації додатку

Наступним кроком було виконано налаштування навігації в межах додатку. Це один з найважливіших кроків в реалізації клієнтської частини. Перш за все було налаштовано логіку роботи нижнього бару навігації, та переходи між екранами в яких нижній бар відсутній. Наступним кроком було налаштовано параметри для ОС IOS.

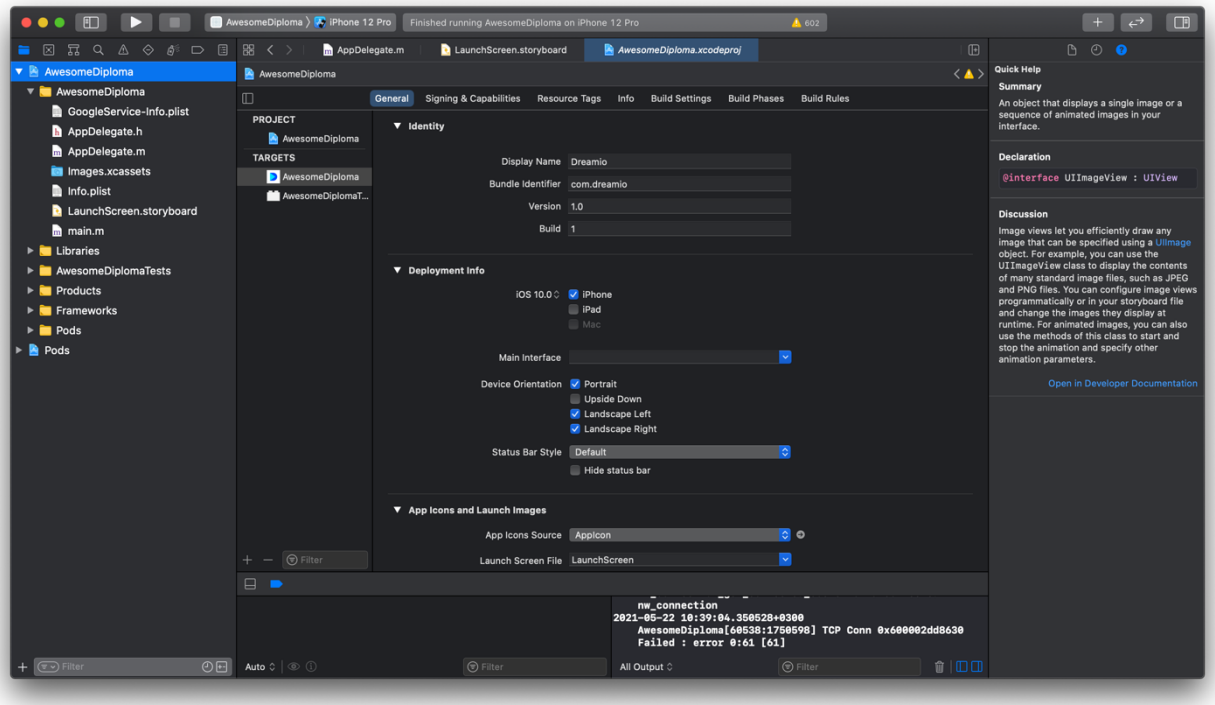


Рисунок 22 – Налаштування для операційної системи IOS

3.4.2. Серверна частина. Інтеграція з клієнтською частиною

Перш за все було створено структуру JSON дерева в Firebase. Також за допомогою Firebase було реалізовано авторизацію в додаток, реєстрацію користувачів та відновлення паролю.

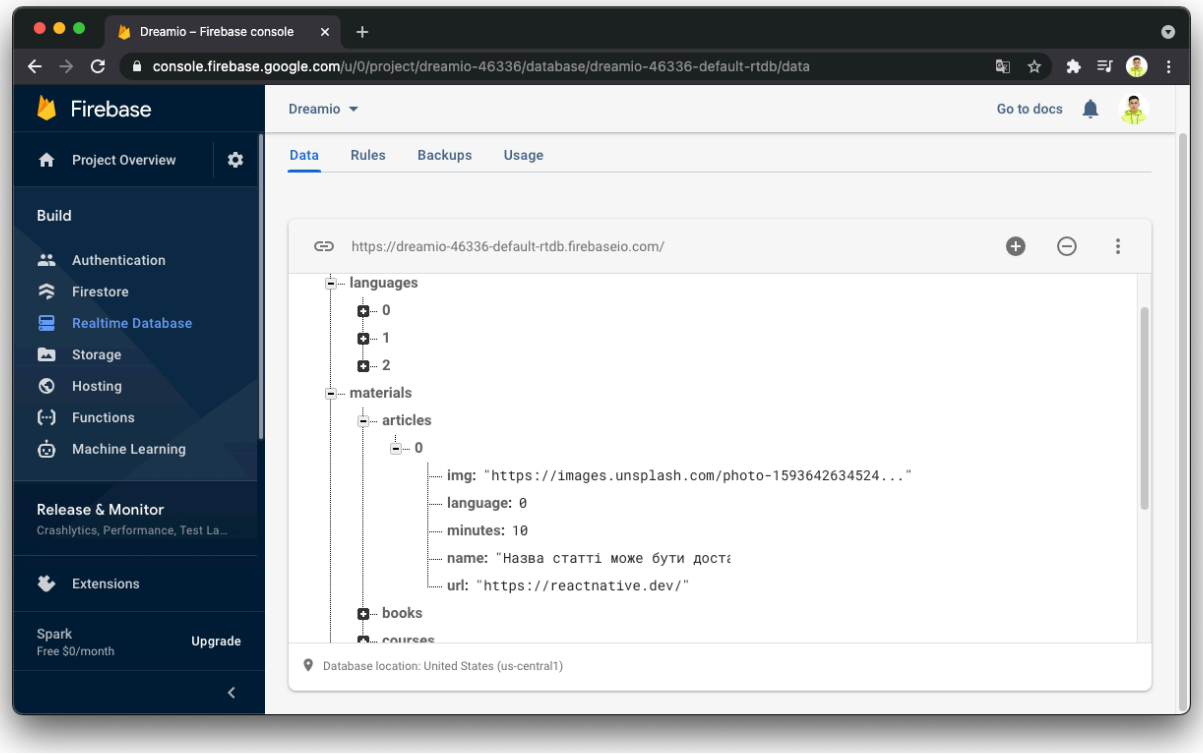


Рисунок 23 – Розробка бази даних

Наступним кроком є інтеграція бази даних Firebase з клієнтською частиною додатку за допомогою XCode IDE.

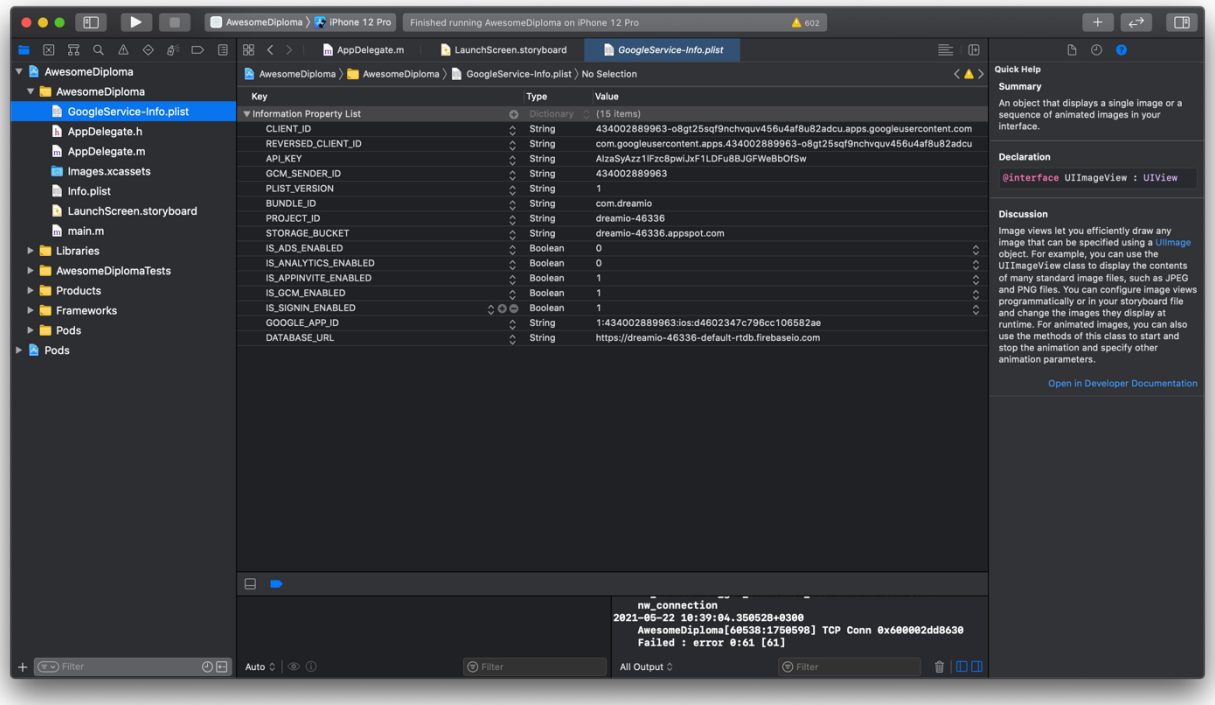


Рисунок 24 – Інтеграція бази даних з клієнтською частиною

3.5. Використання програмного продукту

На етапі проектування та моделювання проекту було проаналізовано та описано цільову аудиторію користувачів мобільного додатку.

Користувачами інформаційного ресурсу можуть бути люди, які тільки почали вивчати мови програмування хочуть отримати детальну та структуровану інформацію про мови програмування, так і програмісти вже з досвідом, які бажають дізнатися “фішки” обраної мови або вивчити нові.

Даний підхід дасть можливість більш точно провести маркетингову компанію мобільного додатку. Перш за все всіх передбачуваних користувачів системи було поділено на сегменти.

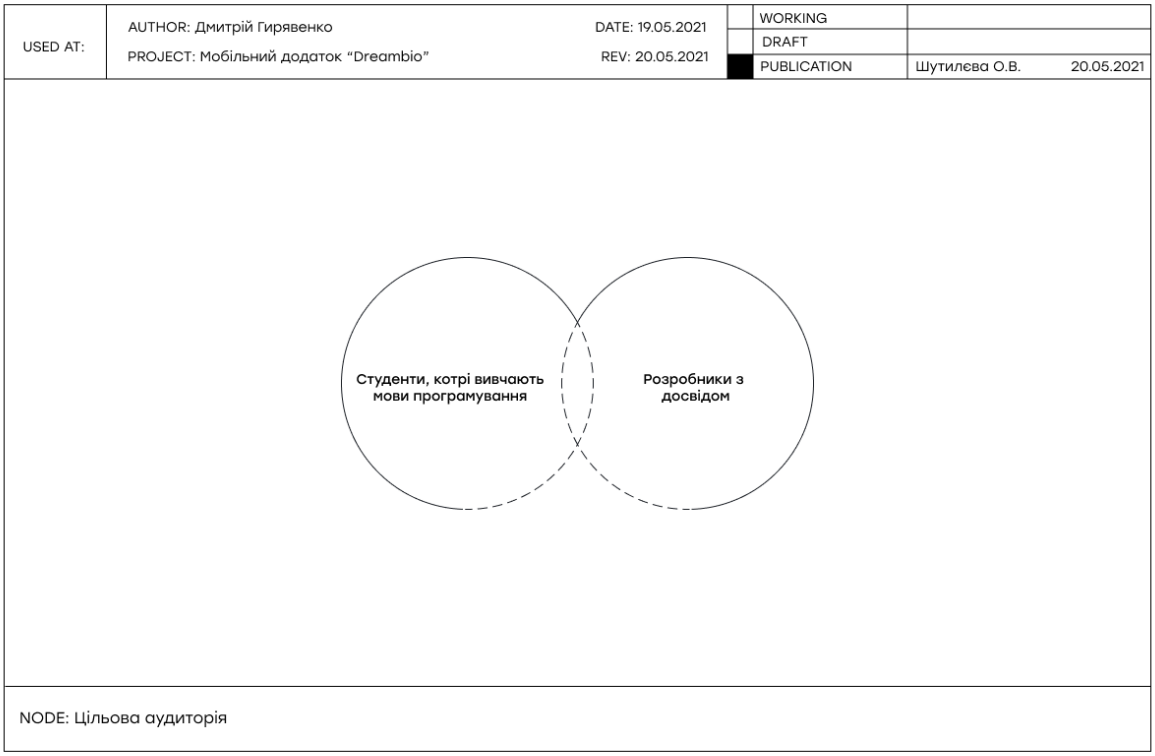


Рисунок 25 – Цільова аудиторія

Безперечно, на поточному етапі життєвого циклу мобільного доданку досить тяжко точно визначити користувачів, тому мобільний доданок буде інтегрований з аналітичними системами, де ми, як розробники, аналізуючи метрики визначити стать, вік, геолокацію користувачів та ще більш точно описувати їх портрети.

ВИСНОВКИ

Під час роботи над випускною роботою перш за все було проведено аналіз предметної області, а також визначено основну проблему самоосвіти у людей, котрі вивчають мови програмування. В додаток, було проаналізовано програмні продукти-аналоги та знайдено точки диференціації для проекту курсової роботи.

Отже, в результаті практики було розроблено мобільний додаток, що задовольняє всім вимогам.

На основі проведеного аналізу предметної області було спроектовано інформаційну систему додатку: визначено функціональні та нефункціональні вимоги до додатку, побудовано функціональну модель, діаграму потоків даних, діаграму використання додатку, діаграму стану акторів. В додаток, було змодельована база даних.

Було досліджено методи і засоби програмної реалізації продукту в результаті чого було обрано створення додатку для мобільної платформи “Android” та “IOS” на основі об’єктно орієнтованої парадигми програмування. Додаток розроблений на мові програмування “Java Script” з використанням крос-платформової технології “React Native”. Це дає змогу підвищити гнучкість та зручність системи в процесі розробки, створити єдину кодову баз ос: “Android” та “IOS”. Розроблена система було випробувана та протестована, вона задовольняє всі програмні вимогами. Під час тестування помилок з рівнем critical не було знайдено.

Загалом, в випускній роботі була повністю вирішена поставлена задача, розглянуто всі супутні завдання та вирішено проблеми, які виникали в ході виконання проекту. Також було отримано нові знання з розробки програмних продуктів для мобільних систем.

СПИСОК ПОСИЛАНЬ

[1] Д. С. і. Д. Етвудом, «Developer Survey Results 2019,» [Онлайновий]. Available: https://insights.stackoverflow.com/survey/2019.

[2] X. Zhou, «The most difficult things about learning to code by yourself — and how to tackle them,» 11 Вересень 2018. [Онлайновий]. Available: https://www.freecodecamp.org/news/the-most-difficult-things-about-learning-to-code-by-yourself-b24ac8c3c23a/.

[3] К. А. ШАГ, «Программисты-самоучки. В чём их проблемы, и почему нужны преподаватели,» 25 Квітень 2021. [Онлайновий]. Available: https://vc.ru/opinions/238396-programmisty-samouchki-v-chem-ih-problemy-i-pochemu-nuzhny-prepodavateli.

[4] К. Л. М. Дэвид А. Марка, Методология структурного анализа и проектирования SADT, 1993.

[5] Л. А.В, «Визуальное моделирование в среде IBM Rational Rose,» 2003.

[6] Т. З. Варфел, Прототипирование. Практическое руководство, Москва: Манн, Иванов и Фербер, 2013.

[7] U. Tools, «2020 Design Tools Survey,» 2020. [Онлайновий]. Available: https://uxtools.co/survey-2020.

[8] M. Kovalyova, «Flux: Архитектура приложений на React.js — всестороннее исследование,» 16 Jun 2018. [Онлайновий]. Available: https://medium.com/@marina.kovalyova/flux-the-react-js-application-architecture-773f515d068d.

[9] Statista, «Cross-platform mobile frameworks used by developers worldwide 2019 and 2020 Published by Shanhong Liu, Jul 2, 2020 React Native is the most popular cross-platform mobile framwork used by global developers, according to a 2020 developer survey. According,» 2020. [Онлайновий]. Available: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours.

[10] B. Eisenman, Learning React Native, O’Reilly Media, Inc, 2017.

[11] Д. УКРАЇНИ, Система розроблення та поставлення продукції на виробництво. Основні терміни та визначення. ДСТУ 3278-95, Київ, 1996 р..

[12] R. U. A. C. CHANDLER, A Project Guide to UX Design: For user experience designers in the field or in the making, United States of America: New Riders, 2009.

[13] S. Diller, N. Shedroff та D. Rhea, Making meaning : how successful businesses deliver meaningful customer experiences, Berkeley, Calif. : New Riders, 2006.

ДОДАТОК А. ЛІСТИНГ КОДУ

Лістинг коду файлу package.json

```
{
  "name": "AwesomeDiploma",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "android": "react-native run-android",
    "ios": "react-native run-ios",
    "start": "react-native start",
    "test": "jest",
    "lint": "eslint ."
  },
  "dependencies": {
    "@react-native-community/datetimepicker": "^3.5.0",
    "@react-native-community/masked-view": "^0.1.11",
    "@react-native-firebase/app": "^12.0.0",
    "@react-native-firebase/auth": "^12.0.0",
    "@react-native-firebase/database": "^12.0.0",
    "@react-navigation/bottom-tabs": "^5.11.10",
    "@react-navigation/drawer": "^5.12.5",
    "@react-navigation/native": "^5.9.4",
    "@react-navigation/stack": "^5.14.4",
    "date-fns": "^2.21.3",
    "formik": "^2.2.7",
    "react": "17.0.1",
    "react-native": "0.64.1",
    "react-native-flash-message": "^0.1.23",
    "react-native-gesture-handler": "^1.10.3",
    "react-native-modal-datetime-picker": "^9.2.3",
    "react-native-reanimated": "^2.1.0",
    "react-native-safe-area-context": "^3.2.0",
    "react-native-screens": "^3.1.1",
    "react-redux": "^7.2.4",
    "redux": "^4.1.0",
    "yup": "^0.32.9"
  },
  "devDependencies": {
    "@babel/core": "^7.14.0",
    "@babel/runtime": "^7.14.0",
    "@react-native-community/eslint-config": "^2.0.0",
    "babel-jest": "^26.6.3",
    "eslint": "^7.26.0",
    "jest": "^26.6.3",
    "metro-react-native-babel-preset": "^0.66.0",
    "react-test-renderer": "17.0.1"
  },
  "jest": {
    "preset": "react-native"
  }
}
```

Лістинг коду файлу AppNavigator.js

```
import React, {useEffect, useState} from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator, TransitionPresets } from '@react-navigation/stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import MyTabBar from './MyTabBar';
import auth from '@react-native-firebase/auth';
import database from '@react-native-firebase/database';
import { useDispatch, useSelector } from "react-redux";
import {updateUser} from '../reducers/User';

import Start from '../scenes/Start';
import Auth from '../scenes/Auth';
import SignUp from '../scenes/SignUp';
import Languages from '../scenes/Languages';

import Home from '../scenes/Home';
import Profile from '../scenes/Profile';
import Way from '../scenes/Way';

import PrivacyPolicy from '../scenes/PrivacyPolicy';
import Terms from '../scenes/Terms';
import Success from '../scenes/Success';
import ChangeTheLanguage from '../scenes/ChangeTheLanguage';
import PasswordReset from '../scenes/PasswordReset';

import Splash from '../scenes/Splash';

const Stack = createStackNavigator();
const Tab = createBottomTabNavigator();

const AuthNavigator = (props) => {
    return (
        <Stack.Navigator
          screenOptions={{
            gestureEnabled: false,
          }}
        >
          <Stack.Screen
            name="Start"
            component={Start}
            options={{
              headerShown: false,
            }}
          />
          <Stack.Screen
            name="Auth"
            component={Auth}
            options={{
              headerShown: false,
            }}
          />
          <Stack.Screen
            name="SignUp"
            component={SignUp}
            options={{
              headerShown: false,
            }}
          />
```

```
            <Stack.Screen
              name="PasswordReset"
              component={PasswordReset}
              options={{
                headerShown: false,
              }}
            />
            <Stack.Screen
              name="Languages"
              component={Languages}
              options={{
                headerShown: false,
              }}
            />
                 <Stack.Screen
                  name="PrivacyPolicy"
                  component={PrivacyPolicy}
                  options={{
                    headerShown: false,
                  }}
                />
                <Stack.Screen
                  name="Terms"
                  component={Terms}
                  options={{
                    headerShown: false,
                  }}
                />
          </Stack.Navigator>
        )
      }

      const BottomTab = () => {
          return (
            <Tab.Navigator tabBar={props => <MyTabBar {...props} />}
                                                  animationEnabled={true}
                                                  initialRouteName="Home">
              <Tab.Screen name="Home" component={Home}/>
              <Tab.Screen name="Way" component={Way}/>
              <Tab.Screen name="Profile" component={Profile} />
            </Tab.Navigator>
          );
        }

        const MainNaw = () => {
          return(
            <Stack.Navigator
            screenOptions={{
              gestureEnabled: false,
            }}>
              <Stack.Screen
                name="BottomTab"
                component={BottomTab}
                options={{
                  headerShown: false,
                }}
              />
              <Stack.Screen
                    name="ChangeTheLanguage"
                    component={ChangeTheLanguage}
                    options={{
                      headerShown: false,
                    }}
                  />
                  <Stack.Screen
                  name="Success"
                  component={Success}
```

```
126:                 options={{
127:                   headerShown: false,
128:                 }}
129:               />
130:                <Stack.Screen
131:                 name="PrivacyPolicy"
132:                 component={PrivacyPolicy}
133:                 options={{
134:                   headerShown: false,
135:                 }}
136:               />
137:               <Stack.Screen
138:                 name="Terms"
139:                 component={Terms}
140:                 options={{
141:                   headerShown: false,
142:                 }}
143:               />
144:           </Stack.Navigator>
145:         )
146:       }
147:
148: const  AppNavigator =  (props) => {
149:
150:   const [authStatus, setAuthStatus] = useState(false);
151:   const [load, setLoad] = useState(true);
152:   const [language, setLanguage] = useState(null);
153:   const dispatch = useDispatch();
154:   const user = useSelector(state => state.user.user);
155:
156:   const  onAuthStateChanged = (user) => {
157:     setAuthStatus(!user ? false : true);
158:     if(user){
159:       let uid = user.uid;
160:       database()
161:       .ref(`users/${uid}`)
162:       .once('value')
163:       .then(snapshot => {
164:         setLoad(false);
165:         let res = snapshot.val();
166:         if(!res){
167:           setLanguage(false);
168:           dispatch(updateUser({uid: user.uid}));
169:         } else {
170:           let language = res.language;
171:           dispatch(updateUser({uid: user.uid, language}));
172:         }
173:
174:       })
175:       .catch(err => {
176:       });
177:     } else {
178:       setLanguage(false);
179:       setLoad(false);
180:     }
181:   }
182:
183:   useEffect(() => {
184:     const subscriber = auth().onAuthStateChanged(onAuthStateChanged);
185:     return subscriber;
186:   }, []);
187:
188:     return (
189:       <NavigationContainer>
190:            {load ? <Splash/> :
191:             authStatus ?
192:             (!language && !user) || (!language && !user.language && user.language !== 0) ?
    <Languages/> : <MainNaw/>
193:              : <AuthNavigator/>
```

```
            }
        </NavigationContainer>
      );
    }

    export default AppNavigator;
```

Лістинг коду файлу Auth.js

```
import React, {useState} from 'react';
import {View, Text, StyleSheet, Image, TouchableOpacity,
        TextInput, Platform, KeyboardAvoidingView, ScrollView} from 'react-native';
import { Formik } from 'formik';
import CustomButton from '../components/CustomButton';
import * as yup from 'yup'
import auth from '@react-native-firebase/auth';
import { showMessage } from "react-native-flash-message";

const loginValidationSchema = yup.object().shape({
    email: yup
      .string()
      .email("Введіть дійсну електронну адресу")
      .required('Потрібна електронна адреса'),
    password: yup
      .string()
      .min(8, ({ min }) => `Пароль повинен містити щонайменше ${min} символів`)
      .required('Потрібен пароль'),
  })


const AuthScenes = (props) => {

    const login = (data) => {

        let response = auth()
        .signInWithEmailAndPassword(data.email, data.password);

        response
         .then(res => {
         })
         .catch(err => {
            showMessage({
                message: 'Увага',
                description: 'При авторизації сталася помилка',
                type: "danger",
                duration: 1500
              });
         });

    }

    return(
        <>
        <Auth login={login} navigation={props.navigation}/>
        </>
    )
}

const Auth = (props) => {
    const [status, setStatus] = useState(false);
```

```
51:
52:     return(
53:         <KeyboardAvoidingView
54:         behavior={Platform.OS === "ios" ? "padding" : "height"}
55:         style={{flex: 1}}
56:       >
57:         <Formik
58:         initialValues={{ email: '', password: '' }}
59:         onSubmit={(values, actions) => {
60:             props.login(values);
61:             actions.setSubmitting(false);
62:         }}
63:         validationSchema={loginValidationSchema}
64:       >
65:         {({ handleChange, handleBlur, handleSubmit, values, errors,
66:             isValid, touched }) => (
67:         <ScrollView contentContainerStyle={{flexGrow: 1}}>
68:         <View style={styles.container}>
69:             <View style={{paddingHorizontal: 16}}>
70:             <View style={styles.header}>
71:                 <Image style={styles.logo} source={require('../source/img/logo.png')}/>
72:                 <Text style={styles.title}>Логін</Text>
73:                 <Text style={styles.description}>Увійдіть для продовження вивчення
    улюбленої мови програмування</Text>
74:             </View>
75:             <View style={{paddingBottom: 20}}>
76:                 <View style={styles.inputContainer}>
77:                     <TextInput style={styles.input} placeholder='Імейл'
78:                             placeholderTextColor='#939496'
79:                             onChangeText={handleChange('email')}
80:                             onBlur={handleBlur('email')}
81:                             value={values.email}
82:                             keyboardType="email-address"
83:                             />
84:                     {(errors.email && touched.email) && <Text
    style={styles.errors}>{errors.email}</Text>}
85:                 </View>
86:                 <View style={styles.inputContainer}>
87:                     <View style={styles.inputContainer}>
88:                     <TextInput style={styles.input} placeholder='Пароль'
89:                             placeholderTextColor='#939496'
90:                             onChangeText={handleChange('password')}
91:                             onBlur={handleBlur('password')}
92:                             value={values.password}
93:                             secureTextEntry
94:                             />
95:                     <TouchableOpacity onPress={() =>
    props.navigation.navigate('PasswordReset')} style={styles.inputBt}>
96:                         <Text style={styles.btText}>Forgot password?</Text>
97:                     </TouchableOpacity>
98:                     </View>
99:                     {(errors.password && touched.password) && <Text
    style={styles.errors}>{errors.password}</Text>}
100:                 </View>
101:             </View>
102:             </View>
103:             <View style={{paddingHorizontal: 45, paddingVertical: 16}}>
104:             <CustomButton disabled={!isValid} action={handleSubmit} title='Війти'/>
105:             <View style={styles.footer}>
106:                 <Text style={styles.footerText}>Немає облікового запису? </Text>
107:                 <TouchableOpacity onPress={() => props.navigation.navigate('SignUp')}>
108:                     <Text style={styles.bt}>Створити</Text>
109:                 </TouchableOpacity>
110:             </View>
111:             </View>
112:         </View>
113:         </ScrollView>
114:          )}
```

```
            </Formik>
            </KeyboardAvoidingView>
        )
    }

    const styles = StyleSheet.create({
        container: {
            flex: 1,
            backgroundColor: '#fff',
            justifyContent: 'space-between'
        },
        logo: {
            width: 55,
            height: 50,
            resizeMode: 'contain'
        },
        title: {
            fontSize: 24,
            paddingTop: 22,
            paddingBottom: 16,
            fontWeight: '500'
        },
        description: {
            textAlign: 'center',
            fontSize: 14,
            color: '#939496',
            paddingHorizontal: 16
        },
        header: {
            alignItems: 'center',
            paddingVertical: 40
        },
        footerText: {
            color: 'rgba(60, 60, 67, 0.6)',
            fontSize: 15,
        },
        bt: {
            color: '#014EFE',
            fontWeight: '500',
            fontSize: 15,
        },
        footer: {
            flexDirection: 'row',
            justifyContent: 'center',
            paddingVertical: 35
        },
        input: {
            paddingVertical: 19,
            borderBottomWidth: 0.5,
            borderBottomColor: '#D6DBE5',
            color: '#939496',
            fontWeight: '500',
            fontSize: 14
        },
        inputContainer: {
            position: 'relative',
            justifyContent: 'center'
        },
        inputBt: {
            position: 'absolute',
            right: 0
        },
        btText: {
            color: '#014EFE',
            fontWeight: '600',
            fontSize: 13,
        },
        disabled: {
            width: 16,
```

```
        height: 16,
        borderRadius: 4,
        borderWidth: 2,
        borderColor: '#C2C2C6',
        marginRight: 8
    },
    active: {
        width: 16,
        height: 16,
        alignItems: 'center',
        justifyContent: 'center',
        backgroundColor: '#007AFF',
        borderRadius: 4,
        marginRight: 8
    },
    icon: {
        width: 12,
        height: 9,
        resizeMode: 'contain'
    },
    confirmText: {
        fontSize: 13,
        color: 'rgba(60, 60, 67, 0.6)'
    },
    confirmContainer: {
        flexDirection: 'row',
        alignItems: 'center',
        marginTop: 15
    },
    errors: {
        fontSize: 11,
        color: '#F3494C',
        paddingTop: 8
    }
});

export default AuthScenes;
```

Лістинг коду файлу ChangeTheLanguage.js

```
import React, {useState, useEffect} from 'react';
import {View, Text, StyleSheet, Image, TouchableOpacity,
        TextInput,
        FlatList} from 'react-native';

import CustomButton from '../components/CustomButton';
import database from '@react-native-firebase/database';
import { useDispatch, useSelector } from "react-redux";
import {updateUser} from '../reducers/User';

const ChangeTheLanguage = (props) => {
    const [languages, setLanguages] = useState([]);

    let [active, setActive] = useState(null);
    const dispatch = useDispatch();
    const user = useSelector(state => state.user.user);

    useEffect(() => {
        if(user.language || user.language === 0){
            setActive(user.language);
        };
```

```
22:
23:         database()
24:         .ref('languages')
25:         .once('value')
26:         .then(snapshot => {
27:           setLanguages(snapshot.val());
28:         });
29:     }, []);
30:
31:     const saveLanguage = () => {
32:         database()
33:         .ref(`/users/${user.uid}`)
34:             .update({
35:                 language: active
36:             })
37:         .then(() => {
38:             dispatch(updateUser({...user, language: active}));
39:             props.navigation.navigate('Home');
40:         });
41:     };
42:
43:     return(
44:         <View style={styles.container}>
45:             <View>
46:                 <TouchableOpacity onPress={() => props.navigation.goBack()}
    style={styles.leftBt}>
47:                     <Image style={styles.leftBtIcon}
    source={require('../source/img/back.png')}/>
48:                 </TouchableOpacity>
49:                 <Text style={styles.title}>Змінити мову</Text>
50:             </View>
51:             <FlatList
52:                 data={languages}
53:                 keyExtractor={(dta, index) => String(index)}
54:                 renderItem={(data) => {
55:                     let item = data.item;
56:                     return(
57:                         <TouchableOpacity onPress={() => setActive(item.id)}
    style={styles.checkboxContainer}>
58:                             <View style={styles.checkboxRow}>
59:                                 <Text>{item.name}</Text>
60:                                 {item.id !== active ? <View style={styles.disabled}/>
61:                                 : <View style={styles.active}>
62:                                     <Image style={styles.icon}
    source={require('../source/img/tick.png')}/>
63:                                 </View>}
64:                             </View>
65:                         </TouchableOpacity>
66:                     )
67:                 }}
68:             />
69:             <View style={{paddingHorizontal: 45, paddingBottom: 16}}>
70:                 <CustomButton action={() => saveLanguage()}
71:                         disabled={active === false ? true : false} title={'Обрати'}/>
72:             </View>
73:         </View>
74:     )
75: }
76:
77: const styles = StyleSheet.create({
78:     container: {
79:         flex: 1,
80:         backgroundColor: '#fff',
81:         justifyContent: 'space-between'
82:     },
83:     leftBt:{
84:         width: 50,
85:         height: 50,
```

```
        alignItems: 'center',
        justifyContent: 'center'
    },
    leftBtIcon: {
        height: 16,
        width: 8,
        resizeMode: 'contain'
    },
    title: {
        paddingHorizontal: 16,
        paddingVertical: 16,
        fontSize: 24,
        fontWeight: '500'
    },
    checkboxRow: {
        marginLeft: 16,
        paddingVertical: 15,
        borderBottomWidth: 0.5,
        borderBottomColor: '#D6DBE5',
        flexDirection: 'row',
        justifyContent: 'space-between',
        alignItems: 'center',
        paddingRight: 16
    },
    disabled: {
        width: 24,
        height: 24,
        borderRadius: 50,
        borderWidth: 2,
        borderColor: '#C2C2C6'
    },
    active: {
        width: 24,
        height: 24,
        alignItems: 'center',
        justifyContent: 'center',
        backgroundColor: '#007AFF',
        borderRadius: 50
    },
    icon: {
        width: 12,
        height: 9,
        resizeMode: 'contain'
    }
});

export default ChangeTheLanguage;
```

Лістинг коду файлу Home.Js

```
import React, {useState, useEffect} from 'react';
import {View, Text, StyleSheet, Image, TouchableOpacity,
        ActivityIndicator, Dimensions,
        FlatList, Linking} from 'react-native';
import { useDispatch, useSelector } from "react-redux";

let width = Dimensions.get('window').width;
import database from '@react-native-firebase/database';
import { useIsFocused } from '@react-navigation/native'
```

```
10:
11: let tabs = [
12:     {
13:         title: 'Статті',
14:         type: 'articles'
15:     },
16:     {
17:         title: 'Відео',
18:         type: 'videos'
19:     },
20:     {
21:         title: 'Книги',
22:         type: 'books'
23:     },
24:     {
25:         title: 'Курси',
26:         type: 'courses'
27:     }
28: ];
29:

30: const Home = (props) => {
31:
32:     let [activeTab, setActiveTab] = useState(0);
33:     let [data, setData] = useState([]);
34:     let [load, setLoad] = useState(false);
35:     const user = useSelector(state => state.user.user);
36:     const isFocused = useIsFocused()
37:
38:     useEffect(() => {
39:             setActiveTab(0);
40:             getData(0);
41:       }, [isFocused]);
42:
43:     const getData = (index) => {
44:         setLoad(true);
45:         setActiveTab(index);
46:         database()
47:         .ref(`materials/${tabs[index].type}`)
48:         .orderByChild('language')
49:         .equalTo(user.language)
50:         .once('value')
51:         .then(snapshot => {
52:           let res = snapshot.val();
53:           setData(res ? res : []);
54:           setLoad(false);
55:         });
56:     }
57:
58:
59:
60:     return(
61:         <View style={styles.container}>
62:             <View>
63:                 <Text style={styles.title}>React</Text>
64:             </View>
65:             <View style={{flex: 1}}>
66:             <View style={styles.tabs}>
67:                 {tabs.map((item, index) => {
68:                     return(
69:                         <TouchableOpacity
70:                             onPress={() => getData(index)}
71:                             style={[activeTab === index && {backgroundColor: '#007AFF'},
    styles.tab]}
72:                             key={index}>
73:                             <Text style={[styles.tabTitle, activeTab === index && {color:
    '#fff'}]}>{item.title}</Text>
74:                         </TouchableOpacity>
```

```
                    )
                })}
            </View>
            {load && <ActivityIndicator style={{marginVertical: 24}} size="small" color="#014EFE"/>}
            {(!load && data.length === 0) && <Text style={styles.error}>Дані не знайдені</Text>}
            {activeTab === 2 ? <FlatList
                horizontal={false}
                data={!load ? data : []}
                numColumns={2}
                style={{
                     paddingHorizontal: 16,
                     paddingBottom: 16,
                     marginTop: 16,
                     flex: 1
                    }}
                renderItem={(data) => {
                    let item = data.item;
                    return(
                        <TouchableOpacity onPress={() => Linking.openURL(item.url)} style={[{paddingBottom: 16, flex: 0.5, marginHorizontal: 15}]}>
                            <View style={styles.imgContainer}>
                                <Image style={[styles.img, {
                                    width: '100%'
                                }]} source={{uri: item.img}}/>
                                {activeTab === 1 && <View style={styles.play}>
                                    <Image style={styles.playIcon} source={require('../source/img/play.png')}/>
                                </View>}
                            </View>
                            <Text style={styles.itemTitle}>{item.name}</Text>
                            {activeTab === 2 && <Text style={styles.author}>{item.author}</Text>}
                            {activeTab === 0 || activeTab === 1 ? <View style={styles.timeContainer}>
                                <Text style={styles.time}>{item.minutes} хв</Text>
                            </View> : null}
                        </TouchableOpacity>
                    );
                }}
                key={'_'}
                keyExtractor={(id, index) => `_${String(index)}`}
            /> : <FlatList
            data={!load ? data : []}
            style={{
                 paddingHorizontal: 16,
                 paddingBottom: 16,
                 marginTop: 16
                }}ы
            keyExtractor={(id, index) => String(index)}
            renderItem={(data) => {
                let item = data.item;
                return(
                    <TouchableOpacity onPress={() => Linking.openURL(item.url)} style={[{paddingBottom: 16}]}>
                        <View style={styles.imgContainer}>
                            <Image style={[styles.img, {
                                width: '100%'
                            }]} source={{uri: item.img}}/>
                            {activeTab === 1 && <View style={styles.play}>
                                <Image style={styles.playIcon} source={require('../source/img/play.png')}/>
                            </View>}
                        </View>
                        <Text style={styles.itemTitle}>{item.name}</Text>
                        {activeTab === 3 && <Text style={styles.author}>{item.author}</Text>}
```

```
                            {activeTab === 0 || activeTab === 1 ? <View
style={styles.timeContainer}>
                                <Text style={styles.time}>{item.minutes} хв</Text>
                            </View> : null}
                        </TouchableOpacity>
                    );
                }}
            />}
                </View>
            </View>
        )
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        backgroundColor: '#fff',
        justifyContent: 'space-between'
    },
    leftBt:{
        width: 50,
        height: 50,
        alignItems: 'center',
        justifyContent: 'center'
    },
    leftBtIcon: {
        height: 16,
        width: 8,
        resizeMode: 'contain'
    },
    title: {
        paddingHorizontal: 16,
        paddingVertical: 16,
        fontSize: 24,
        fontWeight: '500'
    },
    tabs: {
        flexDirection: 'row',
        justifyContent: 'space-between',
        paddingHorizontal: 23
    },
    tabTitle: {
        color: '#1C2026',
        fontSize: 13
    },
    tab: {
        paddingHorizontal: 12,
        paddingVertical: 7,
        borderRadius: 8
    },
    img: {
        height: 190,
        resizeMode: 'cover',
        borderRadius: 4,
    },
    itemTitle: {
        fontSize: 14,
        paddingTop: 12,
        fontWeight: '500',
        paddingBottom: 4
    },
    timeContainer: {
        backgroundColor: '#C2C2C6',
        borderRadius: 4,
        width: 49,
        height: 20,
        alignItems: 'center',
        justifyContent: 'center'
    },
```

```
202:     time: {
203:         fontSize: 12
204:     },
205:     play: {
206:         position: 'absolute',
207:         backgroundColor: 'rgba(28, 32, 38, 0.8)',
208:         width: 48,
209:         height: 48,
210:         borderRadius: 50,
211:         alignItems: 'center',
212:         justifyContent: 'center'
213:     },
214:     imgContainer: {
215:         position: 'relative',
216:         height: 190,
217:         alignItems: 'center',
218:         justifyContent: 'center'
219:     },
220:     playIcon: {
221:         width: 15,
222:         height: 20
223:     },
224:     author: {
225:         fontSize: 12,
226:         color: 'rgba(28, 32, 38, 0.6)'
227:     },
228:     error: {
229:         fontSize: 16,
230:         color: '#F3494C',
231:         paddingVertical: 22,
232:         paddingHorizontal: 16
233:     }
234: });
235: 
236: export default Home;
```