Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему «Мобільний додаток конструктор односторінкових сайтів»

Виконала: студентка групи_____К19-1_____
Спеціальність_____122 «Комп'ютерні науки»___
_____Амеліна Олександра Олександрівна_____
(прізвище та ініціали)
Керівник ___доцент кафедри Мала Ю.А._____
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент _____
(місце роботи)
_____
(посада)
_____
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро − 2023

АНОТАЦІЯ

Амеліна О.О. Розробка мобільного додатку для створення односторінкових сайтів.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 122 «Комп'ютерні науки» – Університет митної справи та фінансів, Дніпро, 2023.

Дана кваліфікаційна робота присвячена розробці мобільного додатка для створення односторінкових сайтів для операційної системи iOS. Основною метою роботи є створення зручного та інтуїтивно зрозумілого інструменту для швидкої розробки сайтів на основі готових елементів сайту.

У процесі розробки було проведено дослідження ринку та аналіз наявних рішень. Проведено аналіз наявних технологічних рішень, для розв'язання поставленої задачі. Для розробки клієнтської частини було обрано мову програмування Swift, з використанням SwiftUI у середовищі Xcode, з використанням Playgrounds. Щоб створити серверну частину використано Python для написання функціонала, AWS – як платформа для розміщення та MongoDB - як база для збереження та керування даними.

Результатом роботи є мобільний застосунок з інтуїтивним інтерфейсом, який дозволяє створювати односторінкові сайти з мобільних пристроїв. Розроблений застосунок підтримує збереження та редагування створених сайтів, а також можливість розміщувати їх в Інтернеті. Всі ці функції дозволять користувачам без знань розробки вебсайтів швидко та легко створювати свої проєкти.

Ключові слова: застосунок, сайт, розробка, програмування, односторінковий, Swift.

ABSTRACT

Amelina O.O. Development of a mobile application for creating single-page websites.

Bachelor's degree qualification work in the specialty 122 "Computer Science" - Customs and Finance University, Dnipro, 2023.

This qualification work is dedicated to the development of a mobile application for creating single-page websites for the iOS operating system. The main goal of the work is to create a convenient and intuitive tool for fast website development based on ready-made website elements.

During the development process, market research and analysis of existing solutions were conducted. An analysis of available technological solutions was carried out to solve the stated problem. Swift programming language with SwiftUI in Xcode using Playgrounds was chosen for the development of the client-side. Python was used to create the server-side functionality, AWS was used as the hosting platform, and MongoDB was used as the database for data storage and management.

The result of the work is a mobile application with an intuitive interface that allows creating single-page websites from mobile devices. The developed application supports saving and editing created websites, as well as the ability to publish them on the Internet. All these features will enable users without web development knowledge to quickly and easily create their projects.

Keywords: application, website, development, programming, single-page, Swift.

ЗМІСТ

ВСТУП

Актуальність проблеми, пов'язаної з розробкою мобільного додатку для створення односторінкових сайтів, полягає в висхідному попиті на прості та ефективні інструменти для створення та публікації вебсторінок. Сьогодні все більше людей та бізнесів шукають швидкі та зручні способи створення інтернет-присутності для презентації свого контенту, товарів та послуг.

Односторінкові сайти - це один з найбільш актуальних трендів веброзробки останніх років. Ці сайти мають кілька переваг порівняно зі звичайними багатосторінковими сайтами: швидка завантаження сторінок, простота розробки та підтримки, зниження вартості розробки та розвитку проєктів.

Сучасний рівень розв'язання даного завдання є обмеженим та містить використання різних онлайн-інструментів та сервісів, які зазвичай потребують підписки або платіжної системи.

На ринку існує безліч застосунків для створення сайтів. Вони дозволяють користувачам швидко та ефективно створювати вебсторінки без необхідності використовувати веббраузер та комп'ютер. Такий ринок конкурентний і насичений, оскільки багато конкурентів пропонують свої рішення для забезпечення користувачів простим та швидким способом створення вебсторінок.

Застосування мобільних технологій для розробки такого типу сайтів буде оптимальним рішенням, оскільки сучасні смартфони й планшети стали повсякденними пристроями для багатьох людей, а мобільний додаток дозволить користувачам зручно створювати та керувати своїми вебсторінками прямо з мобільних пристроїв.

Мета даної роботи розробка мобільного додатку для створення односторінкових сайтів під операційну систему iOS з метою полегшення процесу створення вебсторінок та розвитку інтерне-присутності для користувачів.

Відповідно до мети сформоване завдання:

1. Провести аналіз конкурентного середовища для визначення основного функціоналу, особливостей, недоліків та переваг.

2. Провести детальний аналіз вимог до інтерфейсної частини та функціональності мобільного додатку для створення односторінкових сайтів під операційну систему iOS. Аудиторію операційної системи iOS обрано через наявний потенціал принести значний економічний ефект в подальших етапах розробки через високу платоспроможність користувачів iOS платформи. iOS аудиторія відома своєю високою платоспроможністю, оскільки користувачі цієї платформи насамперед шукають якісні та інноваційні продукти. Вони готові витратити гроші на високоякісні додатки, які задовольнять їхні потреби та нададуть значущу вартість. Це створює перспективи для успішної монетизації додатка в майбутньому шляхом отримання доходу від користувачів.

3. Використати мови програмування Swift та фреймворк SwiftUI для розробки інтуїтивного та ефективного користувацького інтерфейсу додатку, що надає ряд переваг:

– Swift є сучасною та мовою програмування, що швидко розвивається та має простий синтаксис і потужні можливості;

– SwiftUI забезпечує декларативний підхід до розробки користувацького інтерфейсу, що спрощує створення естетичних та інтуїтивно зрозумілих дизайнів.

4. Розробити серверну частину на Python з використанням бази даних MongoDB для забезпечення збереження та доступності редагування створених вебсторінок.

5. Реалізувати можливість швидкого та легкого створення та публікації вебсторінок з використанням шаблонів елементів та інструментів.

6. Забезпечити зручний та доступний спосіб редагування дизайну та контенту веб-сторінок через мобільний додаток.

Предмет дослідження - мобільний додаток для створення односторінкових сайтів під операційну систему iOS.

Об'єкт дослідження: процес створення вебсторінок та розвиток інтернет-присутності для користувачів, зокрема тих, хто має обмежені навички веброзробки.

Методи дослідження, які бути використані для реалізації даного проекту, включають:

1. Аналіз літератури і наукових джерел. Для початку дослідження слід здійснити огляд актуальних літературних джерел, статей, публікацій та наукових робіт, що стосуються розробки мобільних додатків та методів створення вебсторінок. Це допоможе зрозуміти поточний стан дослідження в даній області, існуючі технології та методи, а також ідентифікувати прогалини, на яких можна зосередитися.

2. Аналіз конкурентного середовища. Вивчити існуючі додатки для створення вебсторінок, зокрема Wix, Weedly, Squarespace та Jimbo. Проаналізувати їх функціональні можливості, інтерфейс, переваги та недоліки. Це дозволить зрозуміти, які особливості можна вдосконалити або які нові можливості можна додати у застосунок.

3. Визначення вимог. Перед розробкою мобільного додатку необхідно встановити ключові вимоги до його функціональності та інтерфейсу. Це може включати аналіз потреб цільової аудиторії, вимог щодо швидкості завантаження сторінок, зручності використання, можливостей редагування та публікації вебсторінок тощо.

4. Проектування архітектури: На основі вимог слід розробити архітектуру мобільного додатку. Це включає вибір мови програмування та фреймворку, розробку користувацького інтерфейсу, взаємодію з базою даних та інші технічні аспекти.

5. Реалізація додатка. За допомогою обраної мови програмування та фреймворку слід реалізувати мобільний додаток. Це включає розробку

функціональності створення та редагування вебсторінок, інтеграцію з базою даних та інші технічні аспекти.

Застосунок матиме потенціал вплинути на предметну область, надаючи простий та доступний інструмент для створення вебсторінок, що сприятиме розвитку інтернет-присутності та підтримці індивідуальних потреб користувачів та малого бізнесу. Крім того, його технічно-економічний ефект полягатиме в ефективному використанні ресурсів користувачами та зниженні затрат на веброзробку і публікацію односторінкових сайтів.

Таким чином, розробка мобільного додатку для створення односторінкових сайтів під операційну систему iOS є важливим кроком у напрямку полегшення процесу створення вебсторінок та розвитку інтернет-присутності для широкого кола користувачів.

Дана кваліфікаційна робота складається зі вступу, трьох розділів та висновків. Робота містить 50 сторінок, 9 рисунків, 30 літературних джерел.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд технологій для розробки мобільних застосунків

Розробка мобільних застосунків для різних пристроїв - це творчий та доволі складний процес, який вимагає не лише написання коду, але й врахування багатьох факторів. Створення застосунків для різних мобільних пристроїв потребує написання програмного коду для специфічних платформ, таких як Android і iOS. Але й одним із найважливіших завдань для розробників є забезпечення зручного та простого дизайну інтерфейсу, щоб користувачі могли з легкістю користуватися додатком та не витрачали занадто багато часу на пошук необхідних функцій.

Для створення мобільного додатку на платформі Android потрібно знати Java або Kotlin, а для iOS - мову програмування Swift або Objective-C. Для забезпечення безпеки додатка, розробник повинен володіти знаннями основ захисту від злому та зловживань.

Крім того, розробники можуть додавати нові функції та удосконалювати наявні, щоб зробити застосунок ще більш корисним та привабливим для користувачів. Варто звертати увагу на продуктивність застосунку, адже на це впливають різні фактори, такі як різні версії операційних систем, моделі пристроїв та інші технічні аспекти.

Щоб забезпечити швидкий та безперебійний доступ до необхідних функцій, розробники повинні зменшити кількість натискань користувача. Також слід враховувати, що різні операційні системи мають свої особливості, і код додатка повинен бути написаний з урахуванням цих відмінностей.

Отже, розробка мобільних додатків - це складний процес, який вимагає від розробників великої уваги до деталей та вміння працювати з різними технічними аспектами. Але, якщо все зроблено правильно, застосунок може

стати дуже популярним серед користувачів та приносити значну економічну ефективність [2].

1.1.1 Типи розробки мобільних застосунків

З появою мобільних пристроїв змінилося спілкування з технологіями в повсякденному житті. Від покупок до спілкування з близькими та навіть управління бізнес-операціями, мобільні пристрої стали невіднятною частиною життя для мільйонів людей по всьому світу. Мобільні застосунки стали необхідністю для більшості компаній, які працюють у галузі технологій. Ці додатки допомагають бізнесу підтримувати зв'язок зі своїми клієнтами та покупцями. І, якщо було вирішено розробити мобільний застосунок, необхідно визначитися з варіантом розробки: нативна або кросплатформна.

Нативна розробка (Native Development) передбачає створення програми для мобільного пристрою конкретною мовою програмування під конкретну платформу. Цей підхід забезпечує високу продуктивність, безпеку, стабільність та масштабованість, а також дозволяє розробникам повністю використовувати унікальні можливості кожної операційної системи. Так, для розробки додатків для Android використовують мови Java та Kotlin, а для iOS - Swift. До переваг нативної розробки можна віднести досить швидку реакцію на дії користувача, можливість мати прямий доступ до апаратної частини та розробити найбільш звичний для користувача конкретної платформи інтерфейс.

Однак, нативна розробка має і свої недоліки. Наприклад, вона потребує більше часу та ресурсів на розробку і підтримку, що призводить до високої вартості виготовлення застосунків. Крім того, розробка додатків під різні платформи може займати значно більше часу та зусиль, ніж розробка єдиного додатка, який підтримується на різних платформах. Це може бути особливо складно, якщо розробник має обмежені ресурси або відсутність відповідного

досвіду. При цьому, тестування різних версій застосунку займе багато часу та зусиль, що може вплинути на загальний час розробки та вартість проєкту.

Кросплатформна розробка (Cross-Platform Development) дозволяє економити час та гроші, уникнувши необхідності створювати окремі застосунки для кожної платформи. Кросплатформна розробка проводиться за допомогою web-технологій - HTML, CSS та JavaScript, які дозволяють розробити застосунків відразу на кілька платформ. Але для того, щоб застосунок працював згідно зі своєю платформою, його потрібно перекладати на зрозумілу платформі мову, або додавати проміжну ланку - яка стане перекладачем. До переваг кросплатформної розробки можна віднести низьку вартість розробки, адже для цього іноді достатньо буде залучити одного фахівця.

Однак, кросплатформна розробка має свої недоліки. Деякі функції можуть не працювати на всіх платформах, що може призвести до проблем з роботою застосунку. Також кросплатформні застосунки можуть бути повільними та мати простий інтерфейс, який потрібно буде додатково допрацьовувати.

Під час розробки мобільного додатку важливо враховувати функціональність, адаптивність, вартість та оптимізацію. Крім того, важливо слідкувати за останніми тенденціями та технологіями в галузі розробки мобільних застосунків, щоб забезпечити застосунок конкурентоспроможністю і корисністю для користувачів. Наприклад, можна розширити функціональність застосунку, додавши нові функції, які забезпечать конкурентні переваги перед іншими гравцями на ринку. Крім того, можна покращити адаптивність застосунку, щоб забезпечити безперебійність роботи на всіх пристроях з різними розмірами екранів. Також можна оптимізувати додаток, щоб забезпечити швидку та безперебійну роботу, а також знизити витрати на його розробку та підтримку.

Отже, вибір між нативною та кросплатформною розробкою залежить від потреб та можливостей. Перш за все, необхідно визначити цілі, що стоять

перед додатком, а потім обрати такий варіант розробки, який найбільш відповідає потребам та можливостям [9].

Для розробки мобільного застосунку по створенню односторінкових сайтів обрано нативний тип. З урахуванням вибору операційної системи iOS для застосунку потреб у розгляді кросплатформної розробки немає. Адже це забезпечить високу продуктивність, безпеку, стабільність та масштабованість, а також дозволить повністю використовувати унікальні можливості операційної системи.

Для подальшого масштабування продукту можна буде переводити розробку у кросплатформний тип для операційних систем: Android, Windows, Linux тощо.

1.1.2 Мови програмування та фреймворки для розробки.

Досліджуючи світ технологій розробки мобільних застосунків, можна зрозуміти, що він постійно змінюється і вимагає ретельного планування кожного етапу процесу. Відповідно до цього, важливо підібрати правильні технології розробки застосунків, які не тільки відповідають їхньому призначенню, але й допоможуть зменшити витрати та визначити необхідні ресурси. Зважаючи на це, розглянемо більш детально різні мови програмування та їх переваги та недоліки, які допоможуть вам зробити відповідний вибір.

Swift - це мова програмування від Apple для створення застосунків на різних платформах. Вона є потужним інструментом для розробки як нативних, так і вебзастосунків, завдяки чому знайома багатьом розробникам, які займаються розробкою програмного забезпечення. Swift підтримує сучасні функції, такі як дженерики та кложури, що дозволяє легко та ефективно програмувати на ній. Порівняно з Objective-C, Swift має вищу швидкість та покращену безпеку, а також спрощений спосіб виправлення помилок в коді та стабільність завдяки бібліотекам. Ще однією перевагою Swift є те, що вона

підтримує функціональне програмування, що дозволяє писати чистіший та більш структурований код.

Переглядаючи недоліки слід зазначити, що Swift переважно використовується для розробки застосунків для пристроїв Apple, тому для розробників, які працюють з іншими платформами, вона може бути не найкращим вибором. Також варто відзначити, що з попередніми версіями мови можуть виникати проблеми сумісності, що може перешкоджати розробці та підтримці додатків [26].

Kotlin - це мова програмування, що була створена компанією JetBrains у 2011 році для заміни Java, може використовуватися там, де працює Java. Kotlin має декілька переваг:

– компілюється в байт-код, тому її можна запускати на будь-якій платформі, яка підтримує Java;

– сумісна з наявними бібліотеками Java;

– підтримує як об'єктноорієнтований, так і функціональний стиль програмування.

Але є деякі недоліки використання Kotlin:

– компіляція займає більше часу, ніж для Java;

– менше бібліотек та фреймворків, ніж у Java;

– не така велика спільнота розробників, як у Java [18].

JavaScript - використовується в React Native - фреймворку для розробки мобільних та настільних додатків.

Його переваги включають:

– розроблена інфраструктура з широким вибором фреймворків та бібліотек;

– JavaScript-застосунки не потребують встановлення на комп'ютер користувача;

– будь-який браузер на будь-якій операційній системі підтримує цю мову, і не буде проблем з запуском, як на настільному комп'ютері, так і на мобільному пристрої.

Недоліки:

– відсутність продуктивності порівняно з нативними та кросплатформними додатками;

– не підтримує можливість роботи з файлами та не працює з потоками введення-виведення;

– неможливо знати, чи працює програма до того, як її виконання досягне потрібного рядка;

– має лише одне успадкування [20].

Dart - використовується в Flutter, фреймворку для створення мобільних, веб та настільних додатків для Android, iOS, Windows, macOS та Linux.

Переваги:

– код на Dart працює швидше, ніж на JavaScript;

– Dart має строгу синтаксичну структуру, що допомагає уникнути помилок;

– Google підтримує Dart, тому є багато матеріалів на цю мову програмування.

Недоліки:

– Dart - досить нова мова програмування, яка рідко використовується на ринку;

– Dart має обмежену парадигму об'єктів (класів);

– У Dart неможливо перейменувати функцію без написання іншого оператора присвоєння [6].

C# - об'єктноорієнтована мова програмування від Microsoft, яка використовується в Xamarin.

Переваги:

– розвивається добре завдяки зусиллям Microsoft;

– найбільш перспективна мова для початківців;

– додано функціональне програмування (F #);

– велика спільнота програмістів.

Недоліки:

— орієнтація лише на .NET;

— тільки малі бізнеси, студенти та окремі програмісти можуть користуватися ним безплатно [6].

1.2 Аналіз потреб користувачів у створенні односторінкових сайтів

Односторінковий сайт – це вебресурс, який складається з однієї сторінки, що містить усі необхідні для користувача відомості. На відміну від звичайних сайтів, які складаються зі сторінок, односторінковий сайт не передбачає переходу між різними сторінками на одному домені. Це забезпечує зручність користування та швидкість завантаження сторінки.

Найчастіше односторінкові сайти бувають лендінгами, які створюються з метою продажу певного продукту або послуги. Але не завжди односторінковий сайт повинен виконувати функції лендінгу. Цільовою може бути будь-яка сторінка будь-якого сайту, яка містить унікальний контент та є важливою для користувача.

Таким чином, можна стверджувати, що односторінковий сайт - це не просто лендінг або сайт-візитка, а досить гнучкий формат вебсторінок, який може бути створений для будь-якої мети та галузі.

Односторінкові сайти мають великий спектр застосування в різних галузях:

– для продажу - односторінковий сайт може бути використаний як цільова сторінка для продажу товарів або послуг, на якій розміщують інформацію про продукт та форму замовлення;

– для просування продукту – односторінковий сайт може бути майданчиком, на якому відвідувачі можуть ознайомитися з продуктом без можливості купівлі. Це важливий етап маркетингової стратегії;

– для перевірки гіпотез - для дослідження можливих відгуків потенційних споживачів, компанія може розмістити кілька варіантів односторінкових сайтів та спрямувати на них різні сегменти аудиторії;

– для збору контактів - односторінкові сайти підходять для збирання контактів "теплих" користувачів, які зацікавилися продуктом або компанією;

– для представлення особи чи бізнесу - такі сайти можуть виконувати функцію інтернет-візитки, представляючи підприємця, незалежного професіонала, фрилансера або компанію.

Залежно від призначення односторінкові сайти їх можна розділити на кілька груп, серед яких основні:

– для продажів - використовується для продажу одного товару чи послуги або невеликої номенклатури однотипних товарів (послуг);

– для реклами - ознайомлення з брендом або просування нового продукту;

– візитівка – односторінкові сайти такого типу покликані надати стислу інформацію про особу, установу, компанію.

Зазвичай, односторінкові сайти створюються для просування одного продукту або послуги. Такий формат сайту є найбільш ефективним, оскільки він дозволяє маркетологу вести потенційного клієнта шляхом, який він обрав. Клієнт не може переглянути або перейти на іншу сторінку сайту, оскільки йому не надається такої можливості. Інформація представлена на одній сторінці, і відвідувач іде нею згори донизу.

Односторінкові сайти є досить дешевими та швидко створюються, а також легко змінюються в залежності від потреб. Вони ідеально підходять для малого бізнесу, програмістів для представлення свого портфоліо та багатьох інших сфер застосування, через що є дуже популярними.

Однак, односторінкові сайти досить складні для пошукового просування. Це пов'язано з тим, що на такому сайті зазвичай міститься обмежена кількість інформації, і пошукові системи не мають достатньої кількості ключових слів, які можуть використовуватися для підвищення

рейтингу сайту. Однак, якщо ваш сайт міститиме унікальну інформацію, яку не можна знайти на інших сайтах, то він може зайняти провідні позиції в пошуковій видачі за декількома низькочастотними запитами.

Найголовніше на таких сайтах - правильна психологія потенційного клієнта, цікава пропозиція та добре написаний текст, що підведе гостя до потрібної дії [4].

Через основні особливості та переваги односторінкових вебресурсів, такі як: широкий спектр застосування, простота і швидкість у створенні, для створюваного мобільного додатку обрано саме такий тип сайтів.

1.3 Висновки до першого розділу. Постановка задач дослідження

З огляну на розділ "Огляд технологій для розробки мобільних застосунків" можна зробити висновок, що наразі існує багато підходів та технологій для реалізації мобільних застосунків. Від вибору підходу та технології залежить подальше розміщення додатку та його доступність різним операційним системам.

Окрім цього, з інформації у розділі "Огляд технологій для розробки мобільних застосунків" можна відмітити зростаючий попит на створення односторінкових сайтів. Вони мають широкий спектр застосування в різних галузях, що вказує на широкість аудиторії для додатку конструктора лендингів.

Також можна додати, що для ефективної реалізації мобільного додатку для сворення односотрінкових сайтів, потрібно як і вибрати відповідний тип та технології, так і правильно організувати роботу користувачів по створенню односторінкових сайтів. Враховуючи особливості розробки мобільних додатків під iOS, доцільним буде викристатти нативний тип розробки, що забезпечить можливість реалізації складного інтерфейсу.

Важливим аспектом є врахування потреб клієнтів та ринкових тенденцій при плануванні додатку конструктора односторінкових сайтів.

Наприклад, висхідна потреба фізичних осіб та бізнесів забезпечити присутність в Інтернеті потребує реалізації простого та інтуїтивного засосунку по створенню односотрінкових сайтів. Адже розуміння у веб технологіях та вміння розробки сайтів потребують багато знань та часу по реалізації.

Отже, для ефективної реалізації застосунку конструктотра односторінкових, потрібно враховувати не лише технологічні аспекти, але й потреби користувачів та ринкові тенденції. Відповідний аналіз та спланування допоможуть забезпечити успішне користування додаком та підвищить ефективність створення односторінкових сайтів.

РОЗДІЛ 2.
ОГЛЯД НАЯВНИХ ІНСТРУМЕНТІВ ДЛЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Аналіз конкурентних середовищ

Ринок мобільних застосунків зростає з великою швидкістю, а це означає, що користувачі мають широкий вибір конкурентних застосунків для створення вебсайтів. І хоча кожен має свої переваги та недоліки, необхідно знайти той, який найкраще відповідає потребам. Можна звернути увагу на застосунки, які мають додаткові функції для редагування графіки, щоб зробити вебсайт більш привабливим для відвідувачів. Або можна зосередитися на застосунках, які мають більші можливості для редагування коду, якщо сайт містить більше складних функцій. В будь-якому випадку, обирати застосунок необхідно з урахуванням потреб і вимог, щоб отримати максимальну ефективність в процесі розробки вебсайту. Необхідно розглянути детально декілька конкурентних застосунків.

Wix - це онлайн-платформа для створення вебсайтів зі широким спектром інструментів та функціоналом, яка була заснована у 2006 році. Компанія Wix має штаб-квартиру в Ізраїлі, і з часом вона стала однією з найбільших та найвідоміших платформ для створення вебсайтів у світі. Wix є доступним на різних платформах, включаючи веббраузери та мобільні пристрої, такі як iOS та Android.

Функціонал додатка Wix дуже різноманітний та містить наступні можливості:

– редактор вебсайту зі значним набором інструментів для дизайну та розміщення контенту, включаючи текст, фотографії, відео, графіку та інше;

– багато шаблонів для професійного дизайну вебсайту;

– вебхостинг та безплатний домен для вебсайту;

– інтерактивні елементи, такі як форми зворотного зв'язку, соціальні медіапосилання та кнопки дій;

– аналітика для вебсайту, що допомагає відстежувати трафік, конверсію та інші метрики;

– мобільна оптимізація вебсайту.

Попри те, що функціонал застосунку Wix має безліч можливостей, Wix має свої як переваги, так і недоліки.

До переваг використання Wix належать:

– простота використання та інтуїтивний інтерфейс;

– великий вибір шаблонів для вебсайту, які можна настроїти під свої потреби;

– зручність редагування і публікації контенту на вебсайті;

– безплатний план для користувачів, які хочуть спробувати Wix без зобов'язань.

Однак, до недоліків Wix можна віднести:

– обмежені можливості дизайну, які можуть обмежувати творчість користувача;

– невисока швидкість роботи вебсайту, особливо на безплатному плані;

– обмеження можливості зміни шаблону після створення вебсайту;

– оплата за деякі додаткові функції та інструменти, які можуть збільшувати витрати користувача [28].

Jimdo - онлайн-платформа для створення вебсайтів, заснована в Німеччині у 2007 році. Штаб-квартира Jimdo знаходиться в Гамбурзі. Ця платформа є провідною у світі та доступна на різних платформах, включаючи веббраузери і мобільні пристрої (iOS та Android).

Функціонал додатка Jimdo включає:

– шаблони для дизайну вебсайту;

– редактор вебсайту з інструментами для дизайну та розміщення контенту, включаючи текст, фотографії, відео, графіку та інші елементи;

– можливість додавати елементи, такі як форми зворотного зв'язку, соціальні медіапосилання, кнопки дій та інші;

– можливість інтегрувати зовнішні сервіси, такі як PayPal, Google Maps та інші;

– аналітика для вебсайту, яка допомагає відстежувати трафік, конверсію та інші метрики;

– мобільна оптимізація вебсайту та багато іншого.

До переваг використання Jimdo належать:

– простота використання і зручний інтерфейс;

– широкий вибір шаблонів для вебсайту, які можна настроїти під свої потреби;

– зручність редагування і публікації контенту на вебсайті;

– можливість безплатного використання сервісу.

Однак, до недоліків Jimdo можна віднести:

– обмежені можливості дизайну, які можуть обмежувати творчість користувача;

– невеликий вибір шаблонів порівняно з іншими платформами;

– обмеження можливості зміни шаблону після створення вебсайту;

– не дуже висока швидкість роботи вебсайту, особливо на безплатному плані;

– оплата за деякі додаткові функції та інструменти, які можуть збільшувати витрати користувача [7].

Weebly - платформа для створення вебсайтів, заснована у 2006 році в Сан-Франциско. Вона дозволяє користувачам редагувати та керувати своїми вебсайтами з будь-якого місця з доступом до Інтернету. Weebly має безплатні та платні плани з більшими можливостями. Можна додавати та редагувати контент, управляти контентом та оптимізувати його для пошукових систем. Можна створювати онлайн-магазини з каталогом товарів, налаштуванням

оплати та доставлення, купонами та іншими функціями для ефективного ведення бізнесу в Інтернеті.

Застосунок Weebly дозволяє легко:

– додавати матеріали на свій вебсайт з мобільного пристрою;

– редагувати дизайн і контент вебсайту;

– керувати замовленнями в онлайн-магазині;

– відстежувати аналітику вебсайту.

Основні переваги додатка Weebly:

– простота використання;

– багато інструментів для створення вебсайту та керування бізнесом в Інтернеті;

– оптимізований для мобільних пристроїв.

Основні недоліки додатка Weebly:

– обмежена можливість налаштування;

– деякі функції доступні тільки в вебверсії Weebly;

– вартість може бути вищою, ніж у деяких конкурентів;

– потребує підключення до Інтернету для роботи [3].

GoDaddy Website Builder - це онлайн-інструмент для створення вебсайтів від компанії GoDaddy. Застосунок дозволяє користувачам створювати вебсайти без необхідності знання кодування і дизайну.

Функціонал:

– вибір з понад 1500 готових макетів;

– налаштування дизайну: змінювання кольорів, шрифтів, розмірів та інших дизайн-елементів для створення унікального вебсайту;

– додаткові елементи: фотогалереї, форми зв'язку, соціальні мережі та інші, щоб розширити функціонал вебсайту;

– адаптивний дизайн.

Переваги:

– легко використовувати;

– швидке створення вебсайту;

– створення адаптивних вебсайтів.

Недоліки:

– обмежена свобода дизайну;

– обмежена функціональність;

– обмежена підтримка;

– вартість: додаткові функції та можливості можуть коштувати значну суму грошей [5].

Squarespace - це інструмент для створення вебсайтів та інтернет-магазинів, який дозволяє користувачам зручно та швидко створювати сайти без програмування та дизайну. Ось детальний опис функціонала, плюсів та мінусів Squarespace:

Функціонал:

– інтуїтивний конструктор сайтів, який дозволяє змінювати дизайн та макети сайту без програмування;

– безліч готових макетів та тем, що дозволяє швидко створити сайт на будь-який смак;

– інтеграція з соціальними мережами та платіжними системами;

– можливість додавати медіаелементів на сайт;

– безплатний SSL-сертифікат, що забезпечує безпечне з'єднання з сайтом;

– аналітика та інструменти SEO для покращення позицій сайту в пошукових системах.

Переваги:

– простота використання та інтуїтивний інтерфейс;

– безплатний пробний період та недорогі тарифні плани;

– висока якість дизайну та графічних елементів;

– широкі можливості інтеграції з іншими сервісами та додатками;

– хороша підтримка клієнтів та велика база знань.

Недоліки:

– обмежена свобода дизайну порівняно з іншими платформами для створення сайтів;

– обмеження розширеного функціонала, оскільки додатки та плагіни менш доступні, ніж у відкритих системах;

– відсутність можливості перенесення сайту на іншу платформу без значних зусиль [1].

Strikingly - це онлайн-сервіс для швидкого створення вебсайтів, який дозволяє користувачам з мінімальними навичками вебдизайну створювати професійного вигляду сайти. Застосунок дозволяє створювати сайти за допомогою готових шаблонів, з можливістю налаштування кольорової гами, типографіки та зображень.

Основний функціонал додатка Strikingly включає наступне:

– велика бібліотека готових шаблонів для різних видів вебсайтів;

– можливість додавання та редагування тексту, зображень, відео та інших медіаелементів;

– редагування кольорової гами та типографіки;

– налаштування мобільної версії сайту;

– SEO-оптимізація;

– інтеграція з соціальними мережами, формами зворотного зв'язку та іншими сторонніми сервісами.

До переваг додатка Strikingly можна віднести:

– простий та інтуїтивно зрозумілий інтерфейс;

– велика кількість готових шаблонів для різних видів вебсайтів;

– можливість налаштування мобільної версії сайту;

– гнучка настройка кольорової гами та типографіки;

– зручний редактор зображень та відео.

Серед мінусів можна виділити:

– обмежена можливість настройки дизайну шаблонів;

– обмежена функціональність для розширення можливостей сайту;
– високі ціни на преміальні тарифні плани;
– обмежена можливість інтеграції зі сторонніми сервісами [22].

Порівнявши вище зазначений список застосунків для створення вебсайтів, можна зробити висновок про те, що існує багато конкурентних інструментів для створення вебсайтів. Деякі з них мають багато функцій, тоді як інші спрощені та легкі у використанні.

Зокрема, серед найпопулярніших додатків можна виділити Wix, GoDaddy, Squarespace, Jimdo, Weebly та Strikingly, Кожен з цих застосунків має свої переваги та недоліки, тому вибір конкретного інструменту залежить від потреб користувача.

Не зважаючи на те, що мобільні застосунки для створення вебсайтів можуть бути дуже зручними та простими у використанні, важливо пам'ятати про те, що вони мають забезпечити всі основні необхідні функції та можливості для створення високоякісного вебсайту. Тому перш ніж почати розробку мобільного додатку, варто ретельно проаналізувати вимоги до нього в порівнянні з іншими інструментами.

2.2 Інструменти для розробки клієнтської частини

Для створення мобільних додатків під операційну систему iOS доступні декілька мов програмування: Objective-C, Swift, React Native, Xamarin та Flutter. Кожна мова має свої переваги та недоліки.

Objective-C - стара мова програмування для iOS зі складним синтаксисом та повільним виконанням.

Swift - одна з найпопулярніших мов програмування для iOS, яка має простіший синтаксис, покращену безпеку та швидкість виконання.

React Native - фреймворк на JavaScript для швидкого розвитку з простішим синтаксисом порівняно з Objective-C та Swift, але може мати проблеми зі сумісністю з деякими плагінами та бібліотеками.

Xamarin - фреймворк для C#, який забезпечує швидкий розвиток та можливість використання спільного коду для додатків на різних платформах, але менш продуктивний та складний у використанні порівняно з Swift.

Flutter - фреймворк для мови програмування Dart зі швидким розвитком та можливістю створювати красивий та інтерактивний інтерфейс, але може мати проблеми зі сумісністю з деякими плагінами та бібліотеками.

Найкращим варіантом для створення мобільних додатків під операційну систему iOS може бути Swift, який має простіший синтаксис, покращену безпеку та швидкість виконання порівняно з Objective-C.

Тому у кваліфікаційній роботі була використана мова програмування Swift, яка є однією з найбільш популярних мов для розробки застосунків під операційну систему iOS. Її використання дозволяє створювати додатки швидко та ефективно, забезпечуючи високу продуктивність. Розглянемо технології для розробки клієнтської частини додатка більш детально.

Swift - це мова програмування, яка була розроблена компанією Apple з метою створення застосунків для операційних систем iOS, macOS, watchOS і tvOS. Swift поєднує в собі найкращі риси сучасних мов програмування, таких як безпека, швидкість та простота використання, що робить її однією з найпопулярніших мов програмування у світі.

Swift має простіший синтаксис порівняно з Objective-C, що дозволяє програмістам більше часу приділяти розробці функціональності, а не синтаксису. Однією з найбільш важливих особливостей Swift є безпека, що забезпечується за допомогою механізму опціонів та обов'язкової перевірки типів, що дозволяє програмістам зменшити кількість помилок у коді та забезпечити високу якість застосунків.

Swift також має багато інших переваг, наприклад, має вбудовану підтримку для паралельного програмування, що робить її дуже потужною для розробки високопродуктивних застосунків. Крім того, Swift має велику кількість стандартних бібліотек та фреймворків, що дозволяє програмістам швидко розробляти складні застосунки з великою кількістю функціонала. Всі

ці переваги роблять Swift однією з найбільш популярних мов програмування у світі, яку використовують для розробки різних застосунків від ігор до бізнес-додатків.

Швидкість та продуктивність Swift дозволяють розробникам створювати додатки швидко та ефективно. Swift має декілька базових принципів, які дозволяють розробникам ефективно та швидко створювати додатки. Ось декілька з них:

– безпека: Swift має вбудовані функції безпеки, які допомагають розробникам запобігати помилкам в коді та забезпечують безпечну роботу з пам'яттю. Наприклад, опціональні типи даних дозволяють розробникам виявляти та запобігати підвищенню помилковості. Swift має вбудовані функції безпеки, такі як контроль доступу до коду, який дозволяє забезпечити безпеку та надійність додатка;

– простота: Swift має простий та зрозумілий синтаксис, який забезпечує зручну роботу розробників з кодом. Це дозволяє зменшити кількість коду та зробити його більш зрозумілим для розробників;

– швидкість: Swift - це швидка мова програмування, яка може допомогти розробникам створювати додатки, які працюють досить швидко. Swift використовує прогресивний підхід до оптимізації роботи з пам'яттю та кодом;

– інтерактивний досвід: Swift має вбудований інтерактивний досвід у середовищі розробки Xcode, який дозволяє розробникам перевіряти та тестувати свій код в режимі реального часу. Це дозволяє зменшити час на налагодження та покращити продуктивність розробки.

Swift - це мова програмування, яка підтримує об'єктноорієнтоване та функціональне програмування. Це дозволяє розробникам створювати складні та динамічні застосунки з великою кількістю можливостей. Swift має кілька інструментів та функцій, які дозволяють розробникам ефективно працювати з мовою програмування.

Це мова програмування з відкритим вихідним кодом, що дозволяє розробникам створювати та співпрацювати над великими проєктами. Розробників приваблює активна спільнота розробників, яка постійно працює над покращенням та розвитком мови програмування.

Swift є дуже потужною та ефективною мовою програмування для розробки мобільних додатків під операційну систему iOS. Її швидкість виконання, безпека та інтерактивність роблять її ідеальним вибором для розробки додатків під iOS. Нарешті, Swift має велику підтримку від компанії Apple, що забезпечує розробникам високий рівень підтримки та оновлень. Також Swift має широку підтримку у великих компаніях та стартапах, що забезпечує розробникам можливість знайти роботу та співпрацювати з іншими розробниками. Більш того, Swift дозволяє розробникам створювати додатки для більшої кількості платформ, що робить її більш універсальною мовою програмування [23].

SwiftUI - це нова мова програмування для створення інтерфейсів для додатків Apple. Вона була представлена у 2019 році на конференції WWDC і замінює стару мову програмування - UIKit.

SwiftUI має декларативний стиль програмування, що дозволяє описувати інтерфейс без написання коду. SwiftUI автоматично розраховує розміри, положення та інші атрибути компонентів. Також підтримує різні платформи Apple, такі як iOS, iPadOS, macOS, watchOS та tvOS, що дозволяє використовувати один код для різних платформ.

SwiftUI має вбудовану підтримку анімації та інших візуальних ефектів, підтримує різні типи компонентів інтерфейсу та дозволяє розробникам створювати власні компоненти. Це дозволяє створювати складні та інтегровані додатки з великою кількістю функцій та особливостей.

Додатково, SwiftUI дозволяє розробникам зосередитися на більш важливих аспектах розробки, таких як функціональність та взаємодія з користувачем, замість того, щоб тратити час на деталі реалізації інтерфейсу [14].

SwiftUI Playgrounds є потужним інтерактивним середовищем розробки мовою програмування SwiftUI, створене для розробки графічних інтерфейсів додатків для платформ Apple. Дозволяє розробникам ефективно експериментувати з різними аспектами своїх додатків та швидко виявляти та виправляти проблеми в них.

За допомогою цього інтерактивного середовища, розробники зможуть швидко створити прототипи графічних інтерфейсів додатків без потреби налаштовувати середовище розробки або писати багато коду. Вони можуть створювати нові проєкти, встановлювати віджети та взаємодіяти з ними. Розробники можуть створювати свої власні віджети та використовувати їх у своїх додатках.

Однією з найбільш корисних функцій середовища розробки SwiftUI Playgrounds є можливість перегляду та зміни властивостей віджетів у режимі реального часу. Розробники можуть змінювати розміри, кольори, текст та інші параметри віджетів та бачити результати своїх змін миттєво. Це дозволяє розробникам швидко виявляти та виправляти проблеми у своїх додатках та ефективно коригувати їх зовнішній вигляд.

Ще однією корисною функцією SwiftUI Playgrounds є можливість інтеграції з середовищем Xcode. Розробники можуть розпочати розробку свого додатка в середовищі SwiftUI Playgrounds, а потім перейти до Xcode, щоб додати більше функціоналу та налаштувати свій застосунок для публікації. Це дозволяє розробникам працювати з широким спектром інструментів, що збільшить ефективність та прискорить процес розробки.

Узагальнюючи, SwiftUI Playgrounds є потужним інтерактивним середовищем розробки, що дозволяє розробникам швидко створювати та експериментувати з графічними інтерфейсами додатків для платформ Apple. Це інтерактивне середовище дозволяє розробникам негайно бачити результати своїх змін та експериментувати з різними аспектами своїх додатків. SwiftUI Playgrounds може бути важливим інструментом для розробників мобільних

додатків, що дозволяє їм працювати більш ефективно та збільшує їх можливості при розробці та випуску додатків на платформі Apple [27].

Xcode - це інструмент для розробки програм на платформі Apple. Xcode є популярним середовищем розробки для додатків для iOS, macOS, watchOS та tvOS. Використання Xcode для розробки на мові Swift має кілька переваг.

Xcode містить редактор коду, інструменти відлагодження, профілювання та інші, що дозволяє розробникам ефективно створювати та відлагоджувати свої додатки. Xcode також підтримує інтеграцію з іншими інструментами розробки, такими як CocoaPods та Carthage.

Xcode має вбудовану підтримку мови програмування Swift, підсвічування синтаксису, автодоповнення коду та інші інструменти, які допомагають розробникам писати код ефективніше та швидше.

Xcode має вбудовану інтеграцію зі службами розробки додатків для платформи Apple, що дозволяє розробникам легко створювати та редагувати інтерфейс користувача та документувати свої додатки. Легка інтеграція з Git та іншими системами контролю версій дозволяє керувати кодом. Також має підтримку локалізації, що дозволяє легко створювати додатки для різних мов та регіонів.

Отже, Xcode - це потужний інструмент для розробки додатків для платформи Apple з вбудованою підтримкою мови програмування Swift та інтеграцією з іншими інструментами розробки. Ці переваги роблять Xcode найбільш популярним середовищем розробки для платформи Apple [16][30].

2.3 Інструменти для розробки серверної частини

Створення серверної частини мобільного додатку - важливий етап в розробці, де сервер забезпечує збереження, обробку та публікацію даних. Для розробки серверної частини мобільних додатків для операційної системи iOS можна використовувати багато мов програмування, включаючи Java, Ruby,

Node.js та інші. Розглянемо Python та доведемо, чому ця мова програмування може бути найкращим вибором.

Python - мова програмування з великою кількістю сторонніх бібліотек та фреймворків, що дозволяє розробникам зосередитися на функціональності своїх додатків замість деталей реалізації сервера. Python підтримує багатопотоковість, що дозволяє розробникам створювати серверні додатки, які можуть обслуговувати багато запитів одночасно. Крім того, Python має простий та зрозумілий синтаксис, що дозволяє розробникам швидко створювати серверні додатки.

Python має багато фреймворків для розробки серверних додатків, таких як Django, Flask та Pyramid. Django має вбудовану адміністративну панель, що дозволяє розробникам швидко створювати моделі та CRUD-операції з ними. Крім того, Django має велику кількість сторонніх бібліотек, що дозволяє розробникам розширювати функціональність своїх додатків. Flask має дуже простий та легкий використання синтаксис, що дозволяє розробникам швидко створювати серверні додатки. Крім того, Flask має велику кількість сторонніх бібліотек та плагінів, що дозволяє розробникам розширювати функціональність своїх додатків. Pyramid має дуже гнучкий та розширюваний архітектурний стиль, що дозволяє розробникам створювати серверні додатки з різними структурами. Крім того, Pyramid має велику кількість сторонніх бібліотек та плагінів, що дозволяє розробникам розширювати функціональність своїх додатків [12][13].

Python - перша мова програмування для розробки серверних додатків, яку використовують розробники, що підтверджує її ефективність та популярність серед розробників. Python є ефективним інструментом для розробки серверної частини мобільних додатків під операційну систему iOS. Компанії, такі як Instagram та Dropbox, використовують Python для розробки серверної частини своїх додатків. Отже, Python може бути найкращим варіантом для розробки серверної частини мобільних додатків під операційну систему iOS [24].

JSON (JavaScript Object Notation) є форматом даних для обміну між сервером та клієнтом в інтернет-додатках. JSON легкий, легко читається як людьми та легко обробляється комп'ютером.

Однією з переваг використання JSON є те, що він дуже легко оброблюється Python. Python має вбудовану бібліотеку для роботи з JSON, що дозволяє розробникам швидко та ефективно конвертувати дані зі Swift у формат JSON та навпаки. Крім того, JSON підтримується більшістю мов програмування, тому його можна використовувати для обміну даними між додатками, написаними на різних мовах програмування.

Формат JSON дозволяє передавати різні типи даних, такі як числа, рядки, масиви та об'єкти. Це дозволяє передавати будь-які дані між додатками та зберігати структуру даних, що дозволяє швидко та ефективно обробляти передані дані.

Використання JSON може бути оптимальним рішенням для конвертації коду Swift в HTML з використанням Python на сервері для розробки мобільного додатку для створення односторінкових сайтів з можливістю їх публікації.

HTML є мовою розмітки, що використовується для створення вебсторінок. Щоб створити вебсторінку з даними, які були конвертовані зі Swift у JSON, можна використати JavaScript для динамічного створення HTML-коду на основі даних у форматі JSON. Це дозволяє створювати динамічні вебсторінки, які оновлюються автоматично без перезавантаження сторінки.

Використання JSON для конвертації коду Swift в HTML з використанням Python на сервері може допомогти зменшити час розробки та підтримки додатка [17][29].

AWS (Amazon Web Services) - платформа хмарних обчислень, яка дозволяє легко зберігати, обробляти та передавати дані. AWS забезпечує ряд інструментів та сервісів, що дозволяють зосередитись на створенні своїх додатків. Використання AWS як сервера для мобільного додатку для

створення односторінкових сайтів є оптимальним рішенням з причини його можливості публікації та економії часу та коштів на налаштування та підтримку інфраструктури.

AWS забезпечує високу доступність та масштабованість завдяки своїй глобальній інфраструктурі, що дозволяє уникнути збоїв в одному регіоні. Також AWS дозволяє легко масштабувати додатки за допомогою автоматичного масштабування.

AWS має багатошарову модель безпеки, що забезпечує захист від різноманітних загроз, а автоматичне резервне копіювання даних забезпечує безпеку та захист даних.

AWS має широкий набір інструментів та сервісів, які дозволяють легко та ефективно зберігати, обробляти та передавати дані, такі як Amazon S3 для зберігання даних та Amazon EC2 для обробки даних. AWS також дозволяє легко підключати свої додатки до інших сервісів AWS, таких як Amazon RDS, Amazon DynamoDB та Amazon SQS.

AWS є відкритим та гнучким сервісом, який дозволяє вибирати необхідні сервіси та інструменти для додатків та використовувати різні мови програмування та фреймворки, такі як Python, Ruby, Java, Node.js та багато інших.

AWS може бути ідеальним вибором сервера для мобільного додатку, який дозволяє створювати односторінкові сайти та публікувати їх. Цей формат забезпечує високу доступність та масштабованість, високий рівень безпеки. AWS особливо корисний для роботи з великим обсягом даних, за необхідності швидко та ефективно обробляють ці дані [10][15].

MongoDB — одна з найпопулярніших NoSQL баз даних, яка відрізняється високою швидкістю, масштабованістю та гнучкістю для роботи з великими обсягами даних. MongoDB зберігає дані у вигляді документів JSON, що дозволяє зберігати дані у структурованому та неструктурованому вигляді. Використання MongoDB дозволяє розробникам зменшити час на обробку та збереження даних, що робить її оптимальним вибором для

розробки мобільних додатків на платформі iOS з використанням мови програмування Python.

Інша база даних - PostgreSQL. Вона надійна та забезпечує цілісність даних. PostgreSQL підтримує ACID транзакції та має потужний SQL двигун, що робить її популярною серед розробників. Також є варіантом SQLite. Вона зберігає дані у файлі на пристрої та підтримує більшість стандартних SQL команд, що дозволяє зберігати дані в структурованому вигляді. SQLite легкий та простий у використанні, що робить його популярним серед розробників мобільних додатків.

Для розробки мобільного додатку для iOS на Python, MongoDB є найкращим варіантом бази даних. MongoDB працює з JSON-документами, що забезпечує швидку та зручну обробку даних. MongoDB також масштабується, що дозволяє розширювати базу даних за потреби.

MongoDB має декілька переваг над іншими базами даних, наприклад, гнучку схему даних, яка дозволяє змінювати структуру даних без необхідності змінювати схему бази даних.

– MongoDB дозволяє розміщувати дані на різних серверах, що дозволяє розширювати базу даних за потреби;

– MongoDB має гнучку розширюваність та підтримку спільноти, що дозволяє швидко знаходити відповіді на будь-які запитання та проблеми, що виникають під час розробки;

– MongoDB дозволяє ефективно створювати мобільні додатки на базі iOS з високою швидкістю та гнучкістю обробки та збереження даних;

– база даних MongoDB забезпечує високу масштабованість та дозволяє працювати з даними на високому рівні.

Використання офіційної бібліотеки для Python - pymongo - зробить процес розробки більш зручним та простим та дозволить швидко створювати мобільні додатки для створення односторінкових сайтів на платформі iOS з використанням мови програмування Python [19][21].

2.4 Висновки до другого розділу

З огляду на розділ "Аналіз конкурентних середовищ" можна зробити висновок, що існує багато сервісів та платформ по створенню односотрінкових сайтів. Деякі з них мають багато функцій, тоді як інші спрощені та легкі у використанні. Деякі з них мають багато функцій, тоді як інші спрощені та легкі у використанні. Виділено основні аспекти розглянутих застосунків: більше функціоналу доступно на вебверсії продукту, не всі мають можливість розміщення створеного сайту на сервері сервісу, обмеженість функціонала в деяких сервісів через наявність лише шаблонів.

Розділи "Інструменти для розробки клієнтської частини" та "Інструменти для розробки серверної частини" описують технології, що допоможуть реалізувати мобільний застосунок для створення односторінкових сайтів. З урахуванням мети даної кваліфікаційної, у процесі розробки додатка планується використання найсучасніших технологій та тенденцій, що дозволять підвищити якість та ефективність його роботи.

Застосунок буде складатися з двох частин: клієнтської та серверної. Клієнтська частина додатка буде написана на Swift з використанням Swift UI у програмному середовищу Swift Playground та Xcode. Для забезпечення високої швидкості та якості роботи додатка планується використання найновіших фреймворків та інструментів.

Серверна частина додатка буде написана на Python з використанням JSON формату, AWS та бази даних MongoDB. Для забезпечення високої доступності та надійності додатка планується використання технологій хмарного зберігання та обробки даних. Крім того, планується розробка власних алгоритмів для конвертування створеного сайту мовою програмування Swift у HTML за допомогою JSON формату, що надасть змогу користувачам публікувати сайт в Інтернеті.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

3.1 Формування вимог для створюваного мобільного застосунку

Перед тим, як розпочати розробку мобільного додатку для створення односторінкових сайтів, необхідно визначити вимоги, які гарантуватимуть високу якість та зручність використання для користувачів.

1. Клієнтська частина додатка повинна мати інтуїтивний та простий інтерфейс, щоб користувачі могли створювати вебсайти без особливих знань в галузі веброзробки.

2. Застосунок повинен бути розроблений з врахуванням принципів UI / UX та мати привабливий та сучасний дизайн.

3. Застосунок повинен мати можливість вибору шаблону елементів для розміщення на сторінку, що дозволить користувачам швидко створювати вебресурси з різноманітним дизайном.

4. Користувачі повинні мати можливість додавати текстовий та мультимедійний контент на створювані сторінки, а також зберігати та відновлювати свої проєкти.

5. Застосунок повинен мати можливість попереднього перегляду вебсторінки перед її публікацією, щоб користувачі могли переконатися, що їхні сторінки відображаються належним чином.

6. Повинна бути забезпечена можливість редагування та видалення сторінок, створених користувачами.

7. Користувачі повинні мати можливість публікації своїх вебсторінок в інтернеті.

8. Застосунок повинен бути оптимізований для мобільних пристроїв та працювати на платформі iOS.

9. Застосунок повинен бути розроблений з використанням найновіших технологій та забезпечувати ефективну роботу на мобільних пристроях iOS.

10. Застосунок повинен бути сумісний з різними версіями операційної системи iOS та бути перевірений на відповідність вимогам App Store перед публікацією.

11. Повинна бути забезпечена можливість оновлення додатка з App Store, щоб користувачі могли отримувати нові функції та виправлення помилок.

12. Для розробки клієнтської частини додатка буде використовуватися мова програмування Swift та фреймворки Swift UI, середовище розробки Swift Playground та Xcode. Для розробки серверної частини додатка буде використана мова програмування Python з використанням JSON формату, AWS та бази даних MongoDB.

13. Серверна частина додатка повинна забезпечувати безпеку даних та стійкість до навантаження, тому буде використовуватися AWS та MongoDB.

14. Серверна частина додатка повинна забезпечувати можливість зберігання та відновлення проєктів користувачів, а також забезпечувати стійкість до збоїв.

15. Застосунок повинен мати можливість зберігати та відновлювати резервні копії проєктів користувачів, щоб уникнути втрати даних.

16. Застосунок повинен бути захищений від потенційних кібератак та використання вразливостей.

17. Застосунок повинен мати вбудовані функції забезпечення безпеки даних, такі як шифрування та забезпечення конфіденційності користувачів.

18. Застосунок повинен бути розроблений з врахуванням змін в технологіях та стандартах програмування, щоб бути актуальним на довгий період часу.

Розглянуті вимоги забезпечують функціональність застосунку, його зручність в користуванні та захищеність. Виконання цих вимог дозволить створити високоякісний, інтуїтивний у використанні та популярний серед користувачів застосунок.

3.2 Розробка дизайну та інтерфейсу застосунку

Дизайн та інтерфейс застосунків є одним з найважливіших елементів успіху будь-якої програми. Саме візуальне враження від програмного продукту може вплинути на його популярність, а також на задоволеність користувачів. Особливо важливо дотримуватися цього правила при створенні мобільних додатків, що повинні мати простий та зрозумілий інтерфейс для забезпечення найбільшої зручності користувачів.

Мета розділу "Розробка дизайну та інтерфейсу застосунку" у цій кваліфікаційній роботі полягає у створенні привабливого та легкозрозумілого інтерфейсу для мобільного додатку, який дозволяє створювати односторінкові сайти. Необхідно врахувати найбільш сучасні тенденції в дизайні мобільних додатків та забезпечити застосунок високим рівнем функціональності для кращого впливу на користувачів.

При розробці інтерфейсу були дотримані вимоги користувацької зручності та доступності, що дозволить забезпечити максимальний рівень комфорту користувачів. Використано найновіші методики розробки дизайну та елементи UI/UX для досягнення цих цілей. Також, розширили функціональність додатка, орієнтуючись на досвід конкурентів, які допоможуть зробити його ще більш корисним та зручним для користувачів. У результаті роботи отримано застосунок з інтуїтивно зрозумілим та зручним інтерфейсом, який забезпечить найвищу якість користувацького досвіду для користувачів та зробить його більш привабливим для широкої аудиторії.

Основна мета розробленого додатка полягає в тому, щоб допомогти користувачам створювати прості, але ефективні вебсайти без потреби великих

зусиль. Це досягається завдяки можливості створення односторінкових сайтів, які можна легко зібрати з готових блоків. Ці блоки автоматично адаптуються до різних розмірів екрана та мають зрозумілі назви (обкладинка, меню, форма, текст, зображення тощо), що полегшує їх використання.

Більш того, користувачі можуть самостійно створювати власні блоки, використовуючи редактор. Редактор містить набір модулів, таких як текст, форма, геометрична фігура та зображення, що дає можливість розширити можливості додатка та зробити його більш гнучким у використанні. Таким чином, додаток може задовольнити потреби різних користувачів з різними рівнями досвіду у роботі з вебсайтами, забезпечуючи зручний та простий інтерфейс для створення сайтів.

На початковому екрані додатка відразу ж видно розбиття на дві частини: верхню і нижню (рисунок 1).

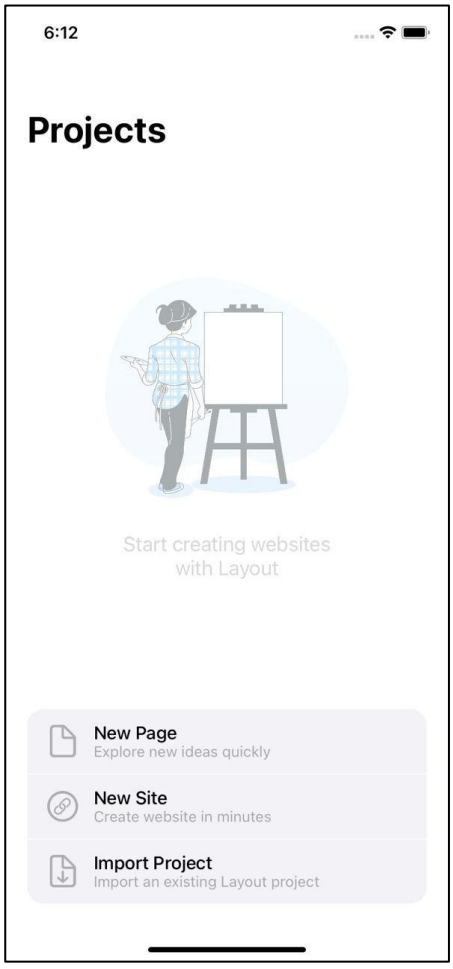

Рисунок 3.1 – Початковий екран

У верхній частині розміщено привітання до користувача, яке забезпечує перший контакт між додатком і його користувачем. Крім того, в цій частині додатка користувачі зможуть знайти список проєктів, над якими вони працювали раніше (рисунок 2).

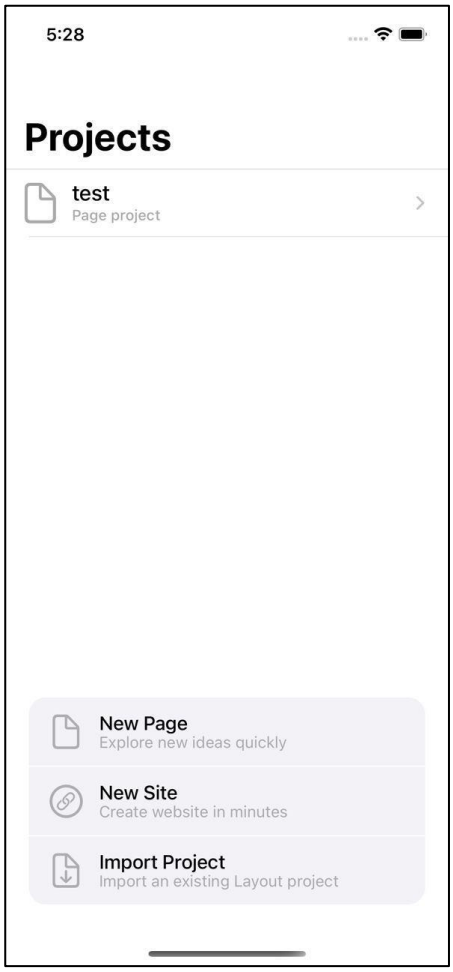

Рисунок 3.2 – Початковий екран з проєктами

Це дає можливість швидко повернутися до проєкту, щоб продовжити роботу над ним або доробити його остаточну версію. На нижній половині екрана знаходяться три функціональні кнопки, що дають можливість користувачам взаємодіяти з додатком. Зокрема, одна з цих кнопок призначена для створення нової сторінки, щоб дозволити користувачам додавати новий контент. Інша кнопка дозволяє створювати новий сайт з нуля, забезпечуючи широкий спектр налаштувань для забезпечення відповідності потребам користувача. Третя кнопка дозволяє завантажувати вже наявний проєкт, щоб

змогти змінювати та оновлювати його згідно з потребами користувача. Завдяки такому інтерфейсу користувачам додатка стає дуже просто та зручно працювати зі створенням та редагуванням своїх проєктів.

Основна сторінка для створення вебсторінки має досить складну структуру, щоб забезпечити максимальну ефективність та зручність для користувачів. Ця сторінка складається з трьох основних частин, кожна з яких відповідає за певний етап створення вебсторінки (рисунок 3):

- на панелі управління розміщені різні інструменти, що дозволяють копіювати та вставляти елементи, а також здійснювати повернення до початкового екрана, користувач може отримати посиланням на свою створену сторінку, щоб ділитись нею з іншими;
- робоча панель надає широкі можливості для редагування та створення вебсторінок, користувач може додавати різні блоки, які автоматично адаптуються під мобільні пристрої та мають смислові категорії, такі як обкладинка, меню, форма, текст, зображення тощо, а допомогою редактора можна створити свій власний блок, додаючи окремі модулі, такі як текст, форма, геометрична фігура, зображення тощо;
- у панелі інструментів знаходяться різноманітні інструменти, що дозволяють вставляти різні елементи на сторінку, кожен інструмент відповідає за розміщення конкретного елемента, такого як фото, текст, фігури, Z-стек, макет центрування, представлення стека. Наприклад, Z-стек дозволяє розмістити елементи один поверх одного, створюючи ефект 3D-зображення, а макет центрування дозволяє відцентрувати елементи за різними параметрами, щоб створити чітке та збалансоване композиційне рішення.

Під час редагування сторінок користувач може легко взаємодіяти з різними елементами, які включені до створеної сторінки. Кожен елемент може бути вибраний окремо, та редагований, щоб відповідати потребам користувача. При виборі будь-якого елемента, відкривається вікно редагування, де можна встановлювати різноманітні налаштування: змінювати кольори, розміри, вирівнювання, шрифти та багато іншого (рисунок 4).

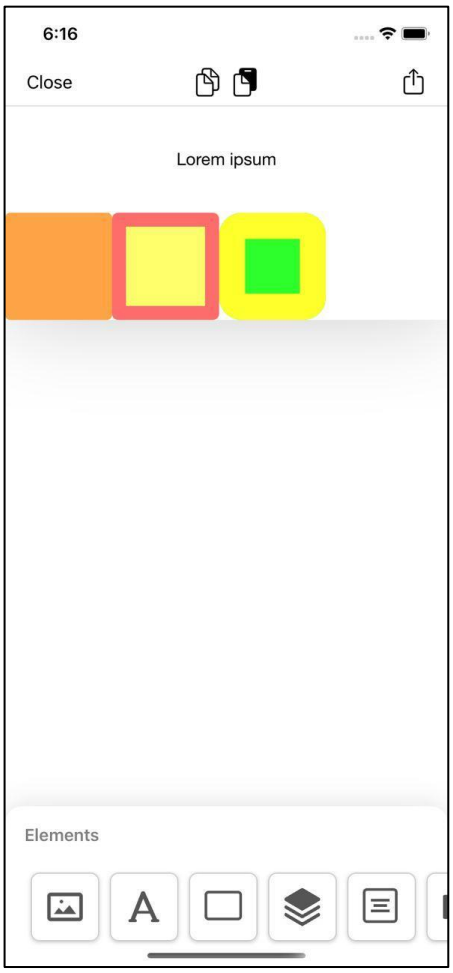

Рисунок 3.3 – Головний екран редагування елементів

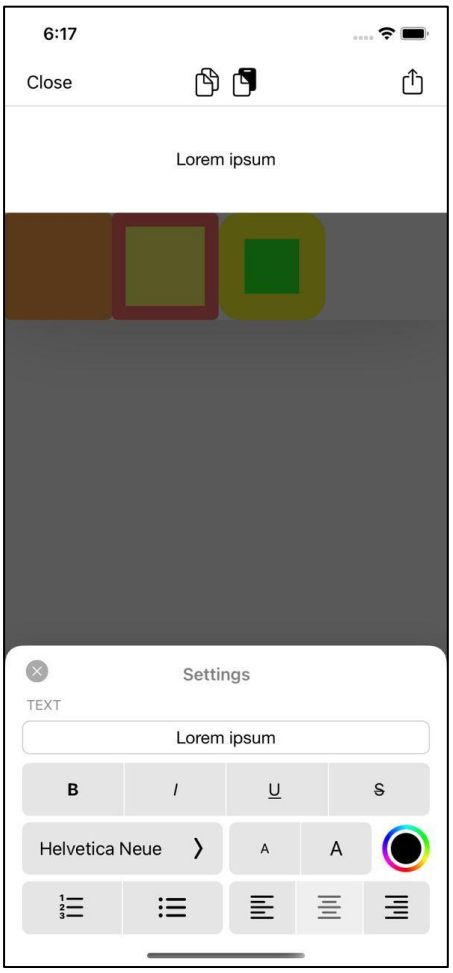

Рисунок 3.4 – Поле з інструментами

Завдяки цьому користувач може зробити сторінку якомога більш індивідуальною та відповідною його потребам. Крім того, відображення вікна редагування забезпечує зручність користування та легкість управління, щоб користувач міг легко знайти необхідні параметри та налаштування.

Екран розміщення сайту на сервері можна відкрити натиснувши кнопку поділитись у верхньому блоці основної сторінки. Цей екран розбито на п’ять умовних блоків, що відцентровано горизонтально (рисунок 8):

• у першому розміщено іконку для відображення функціоналу;

• у другому розміщено заголовок «Share Page»;

• у третьому розташовано блок для виводу посилання та функціональною кнопкою, що відкриває системне модальне вікно;

• у четвертому блоці розміщено кнопку для отримання посилання «Get public link»;

• у п’ятому блоці розміщено поле з інформаційним текстом «Link will be active 24 hours».

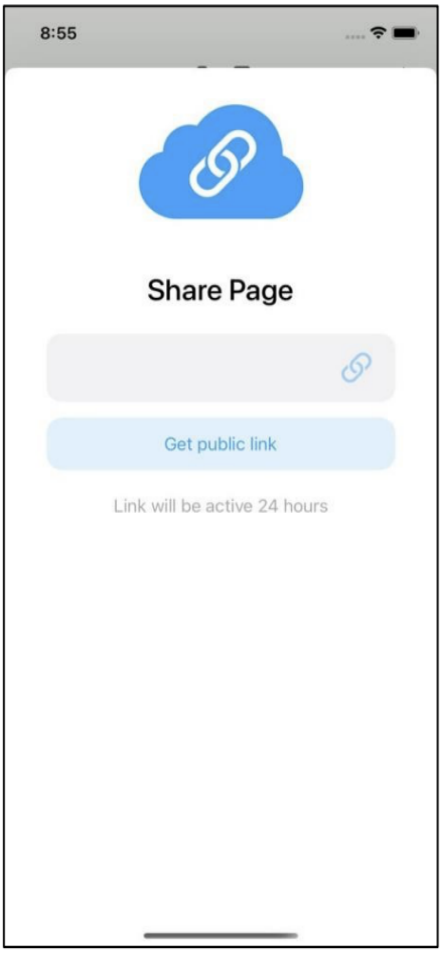

Рисунок 3.5 – Екран розміщення сайту на сервері

При натисканні «Get public link» створений сайт відправляється на сервер, під час виконання цього процесу напис на кнопці змінюється на «Uploading..» (рисунок 9). Після того як сайт успішно розміщено у поле виводиться сформоване посилання на сайт, після чого кнопка для отримання посилання «Get public link» вже вміщує «Upload changes» (рисунок 10).

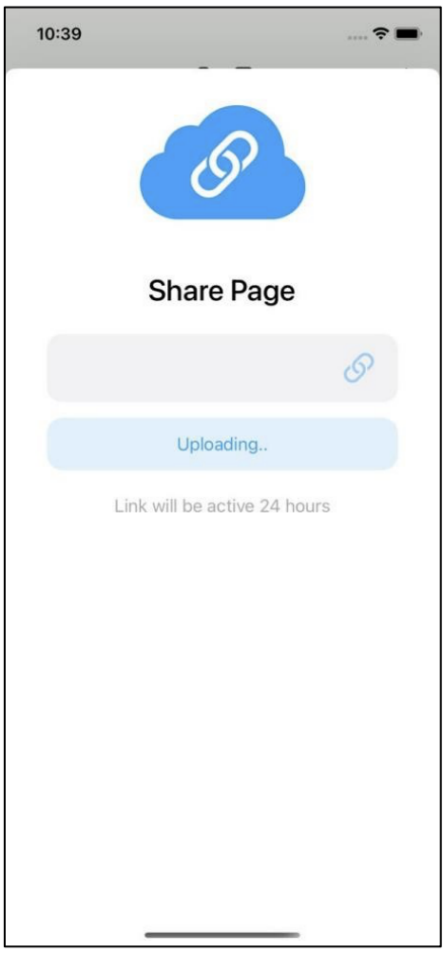

Рисунок 3.6 – Вигляд екрану під час розміщення сайту на сервері

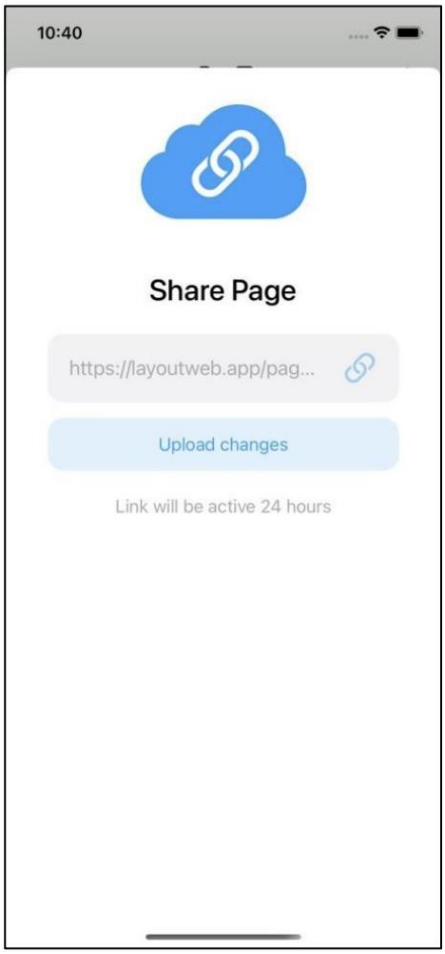

Рисунок 3.7 – Вигляд екрану після розміщення сайту на сервері

При виникненні помилки під час розміщення сайту відкривається вікно, що спливає з інформацією "Error", "Unable to upload page, try again later." (рисунок 11).

Після натискання на кнопку «скопіювати» відкривається системне модельне вікно експорту посилання, де користувач зможе обрати спосіб, щоб відправити посилання (рисунок 12).

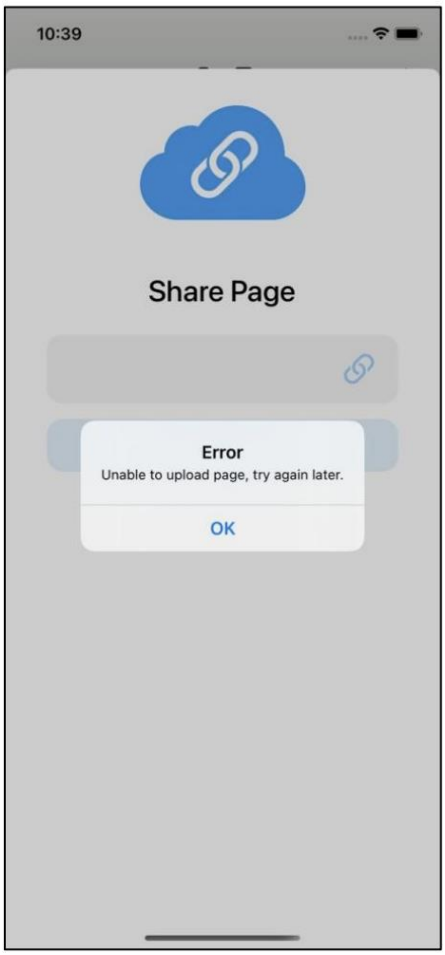

Рисунок 3.8 – Вигляд екрану при виникненні помилок розміщення

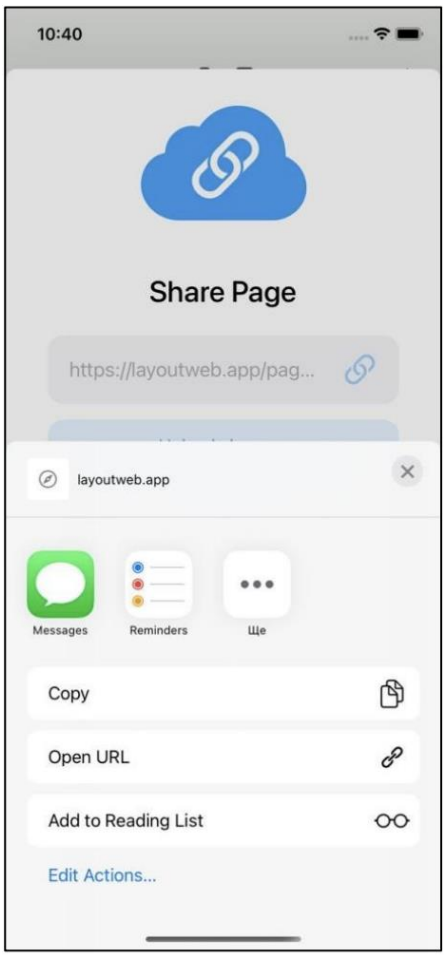

Рисунок 3.9 – Вигляд екрану після відкриття модального вікна експорту

3.3 Покрокове створення програми

У цьому розділі описано покроковий процес створення програмного продукту з використанням різноманітних технологій та інструментів. Розглянуто основні етапи розробки програмного забезпечення, включаючи проєктування інтерфейсу, налаштування бази даних, розробку функціонала та тестування продукту. Кожен етап розглянуто детально, з наведенням прикладів та описом необхідних кроків для його успішної реалізації.

Проєктування інтерфейсу:

– створення детального макета інтерфейсу перед початком розробки, щоб уникнути зайвих затримок.

– розробка інтерфейсу є одним з найважливіших етапів в розробці будь-якого додатка. У процесі проєктування інтерфейсу необхідно забезпечити якість та зручність використання додатка для користувача. За допомогою Swift та SwiftUI, ми створили основні сторінки додатка, додаючи різні елементи, щоб створити зручний та привабливий інтерфейс.

– для забезпечення більшого функціонала додатка, ми створимо модальне вікно, яке дозволить користувачу додавати та редагувати елементи, використовуючи бібліотеку візуальних об’єктів. Таким чином, ми забезпечуємо користувачам простий та зручний інтерфейс, який робить використання додатка приємним та ефективним.

– для забезпечення простоти використання для кінцевих користувачів рекомендується провести тестування фокус-групою. Також можна провести маркетингове дослідження, щоб визначити ключові функції та вподобання цільової аудиторії та проаналізувати подібні продукти на ринку. Для покращення інтерфейсу та відповідності очікуванням користувачів рекомендується регулярно отримувати відгуки під час розробки.

Зберігання даних:

– створення бази даних на MongoDB для збереження сайтів, які створюють користувачі. База даних буде забезпечувати зберігання даних в безпечному та організованому форматі, що дозволить користувачам швидко та легко знайти потрібну інформацію.

– написання функції, яка перетворює Swift-код з вебсайту користувача у формат JSON та додає його до бази даних на сервері. Це забезпечить користувачам можливість зберігати їхні вебсайти та додавати нові, що зберігає час та забезпечує організованість.

– захист даних користувачів за допомогою шифрування та аутентифікації. Для забезпечення високого рівня безпеки, буде застосовано сучасні методи шифрування та аутентифікації, що захистять дані користувачів від несанкціонованого доступу.

– розширення функціональності додатка, додавши можливість видаляти вебсайти користувачів. Це дозволить користувачам легко видаляти застарілі вебсайти, щоб забезпечити більш організоване та ефективне управління їхніми даними.

Створення сервера:

– створення сервера, який містить базу даних MongoDB. Це дозволить нам зберігати дані користувачів та інформацію про створені ними вебсайти.

– для забезпечити зручність користування сервером, він повинен мати можливість отримувати запити від користувачів та відображати створені ними вебсайти. Це зробить простим для користувачів управляти своїми вебсайтами та відстежувати їх прогрес.

– для комунікацій між нашим додатком та сервером, ми розробимо API. Це дозволить двом системам обмінюватися інформацією та працювати разом безперешкодно.

– додати можливості користувачам створювати облікові записи та встановлювати права доступу. Це підвищить безпеку нашої системи та забезпечить захист даних користувачів.

Конвертування коду в HTML виконуватиметься через створену Python-функцію, яка буде обробляти записи бази даних та перетворювати збережений формат JSON у HTML. Цю функцію можна розробити з можливістю повторного використання, щоб вона могла бути викликана кілька разів при додаванні нових даних у базу даних.

Тестування додатка на різних пристроях та версіях операційної системи, щоб переконатися в його стабільності та зручності використання.

3.4 Висновок до третього розділу

У ході розробки мобільного додатку констректора односорінкових сайтів було розроблено функціонал для побудови односторінкових сайтів з використанням шаблоних елементів та можливістю публікації створених сайтів в Інтернет. Клієнтську частину застосунку було реалізовано мовою програмування Swift з допомогою фреймворку SwiftUI. Серверна частина додатку розроблена з використанням мови програмування Python, бази даних MongoDB та хмарну платформу AWS.

Додаток дозволяє користувачам створювати безліч сайтів, редагувати їх та видаляти. Також з урахуванням створеного функціоналу з використанням JSON формату по конвертації коду мовою програмування Swift у вигляд розмітки HTML, користувачі мають змогу за потреби розміщувати створені односторінкові сайти на сервер та отримувати доступ до них в Інтернеті.

Одним з неділіків застосунку полягає у відсутності готових доступних шаблонів сайтів з можливістю редагування для пришвидшення створення сайтів. Також додаток може бути удосконаленний при реалізації на інші операційні системи, такі як Android, Windows та Linux.

Загалом, додаток є корисним інструментом для створення односторінкових сайтів та їх публікації в Інтернет. Він може бути використаний різним типом кристувачів в різних галузях, де необхідно створини односторінковий сайт. З урахуванням зазначених недоліків та можливостей для удосконалення, додаток може бути удосконалений.

ВИСНОВКИ

У результаті дослідження головного завдання було встановлено, що розробка мобільних додатків для створення вебсайтів є необхідним інструментом для підприємців, малих бізнесів та простих користувачів для створення та управління своєю онлайн-присутністю. Через основні особливості та переваги односторінкових вебресурсів, такі як: широкий спектр застосування, простота і швидкість у створенні, для створюваного мобільного додатку обрано саме такий тип сайтів.

З допомогою результатів дослідження предметної області, аналізу технологій програмного забезпечення та розробки мобільного додатку конструктора односторінкових сайтів, було досягнуто поставленої мети дослідження, а саме - створення інтуїтивного та простого застосунку для створення лендінгів. Результати дослідження можуть бути використані користувачами для покращення ефективності та якості сворення сайтів.

Під час створення програмного забезпечення було проведено аналіз конкурентного середовища, що допомогло визначити ключові функції та рішення, які необхідно реалізувати у створюваному продукті. Існує багато конкурентних інструментів для створення вебсайтів. Виділено основні аспекти розглянутих застосунків: більше функціоналу доступно на вебверсії продукту, не всі мають можливість розміщення створеного сайту на сервері сервісу, обмеженість функціонала в деяких сервісів через наявність лише шаблонів.

Для реалізації поставленої задачі проведено порівняльне дослідження наявних інструментів для розробки додатка. З урахуванням проведеного аналізу типів розробки застосунків обрано нативний тип. Це забезпечить високу продуктивність, безпеку, стабільність та масштабованість, а також дозволить повністю використовувати унікальні можливості операційної системи.

Проведено детальний аналіз вимог до інтерфейсної частини та функціональності мобільного додатка. У результаті було отримано перелік вимог, який посприяв створенюю інтуїтивного та простого інтерфейсу застосунку.

Основним інструментом для створення інтерфейсної частини обрано Swift через перевагу над іншими мовами програмування в основних принципах роботи, а саме: безпека, простота, швидкість та інтерактивний досвід. SwiftUI став допоміжною мовою під час розробки інтерфейсу. Як середовище розробки використано Xcode та Playgrounds. Для реалізації можливості розміщення створених сайтів в інтернеті необхідно було створити серверну частину. Реалізації серверної частини проведено на AWS за допомогою Python та MongoDB. Одним з ключових елементів процесу розробки було створення функціонала для конвертації створеного сайту мовою програмування Swift у HTML за допомогою JSON формату, що надасть змогу користувачам публікувати сайту в Інтернеті.

Під час тестування додатка було встановлено, що він працює швидко та точно, забезпечує швидку роботу з елементами, їх доданням на сторінкку, редагуванням та видаленням сайтів. Також було встановлено, що розміщення серверної частини на AWS з функціоналом конвертації мови програмування Swift у розмітку HTML за допомогою JSON формату, забезпечує розміщення сайтів в Інтернеті із загальним доступом.

Загальна функціональність додатка, дозволить користувачам керувати створеними проєктами. За допомогою застосунку користувачі можуть зберігати всі свої вебсайти в одному місці, що дозволяє легко отримувати доступ до них та управляти всім їхнім онлайн-контентом. Крім того, користувачі можуть легко додавати нові вебсайти до своєї колекції, що забезпечує їхню постійну актуальність. Якщо вебсайт застарів, застосунок дозволяє користувачам видаляти ці вебсайти, звільняючи місце та забезпечуючи організований та ефективний доступ до даних.

Також можна зазначити, що розроблений застосунок система має потенціал для подальшого розвитку та вдосконалення. Наприклад, можливо додати шаблони односторінкових сайтів з можливістю редагування та додання унікального контенту. Також можливо ввести підписочний тип монетизації застосунку для його комерційності, що забезпечить економічний ефект.

Для спрощення створення сайтів користувачами основний функціонал додатка містить простий та інтуїтивний інтерфейс. Для створення сайтів різної складності додано основні компоненти, які дозволять це реалізовувати користуючись безліччю доступних властивостей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Андерсон М. Squarespace. Website planet. URL: https://www.websiteplanet.com/uk/website-builders/squarespace/. (Дата звернення: 18.05.23)

2. Розробка мобільних додатків. DAN.IT Education. Дата публікації: 01.01.2021. URL: https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/. (Дата звернення: 18.05.23)

3. Мадрі Д. Weebly. Website planet. URL: https://www.websiteplanet.com/uk/website-builders/weebly/. (Дата звернення: 18.05.23)

4. Односторінкові сайти: види, застосування, приклади. ВМБ. Дата публікації: 24.10.2021. URL: https://vmb.net.ua/552-odnostorinkovi-sajty-vydy-zastosuvannya-pryklady/. (Дата звернення: 18.05.23)

5. Склоттмен А. GoDaddy Website Builder. Website planet. URL: https://www.websiteplanet.com/uk/website-builders/godaddy-website-builder/. (Дата звернення: 18.05.23)

6. C#. Спадковість. Основні поняття. Переваги та недоліки. Загальна форма. Найпростіші приклади. Модифікатор доступу Protected. BestProg. Дата публікації: 24.02.2020. URL: https://www.bestprog.net/uk/2020/02/24/c-inheritance-basic-concepts-advantages-and-disadvantages-general-form-the-simplest-examples-access-modifier-protected-ua/. (Дата звернення: 18.05.23)

7. Хьюмен Х. Jimdo. Website planet. URL: https://www.websiteplanet.com/uk/website-builders/jimdo/#overview. (Дата звернення: 18.05.23)

8. 10 найкращих програм для створення веб-сайтів для пристроїв Android та iPhone. Ciksiti. Дата публікації: 07.12.2021. URL: https://ciksiti.com/uk/chapters/9804-top-10-best-website-builder-apps-for-android-and-iphone-devi . (Дата звернення: 18.05.23)

9. All about top mobile app development technologies in 2023. Nix. Дата публікації: 13.01.2023. URL: https://nix-united.com/blog/all-about-top-mobile-app-development-technologies-in-2023/. (Дата звернення: 18.05.23)

10. AWS Documentation. AWS. URL: https://docs.aws.amazon.com/. (Дата звернення: 18.05.23)

11. Dart documentation. Dart. URL: https://dart.dev/guides. (Дата звернення: 18.05.23)

12. Django Documentation. Django. URL: https://docs.djangoproject.com/en/4.2/. (Дата звернення: 18.05.23)

13. Flask Documentation. Flask. URL: https://flask.palletsprojects.com/en/2.3.x/. (Дата звернення: 18.05.23)

14. Framework SwiftUI: Documentation. Developer. URL: https://developer.apple.com/documentation/swiftui/. (Дата звернення: 18.05.23)

15. How AWS works. AWS Startup Solutions Architects. URL: https://aws.amazon.com/startups/start-building/how-aws-works/. (Дата звернення: 18.05.23)

16. How to Code Xcode. Buildfire. URL: https://buildfire.com/xcode-tutorial/. (Дата звернення: 18.05.23)

17. JavaScript JSON. W3Schools. URL: https://www.w3schools.com/js/js_json_intro.asp. (Дата звернення: 18.05.23)

18. Kotlin docs. Kotlin. URL: https://kotlinlang.org/docs/home.html. (Дата звернення: 18.05.23)

19. MongoDB Documentation. MongoDB. URL: https://docs.mongodb.com/. (Дата звернення: 18.05.23)

20. Mobile Apps (JavaScript). Payrexx. Дата публікації: 18.05.2019. URL: https://developers.payrexx.com/docs/mobile-apps-javascript. (Дата звернення: 18.05.23)

21. MySQL Documentation. MySQL. URL: https://dev.mysql.com/doc/. (Дата звернення: 18.05.23)

22. Newcomer C. Strikingly Review: Is It the Best Website Builder for You. Designbombs. Дата публікації: 09.01.2023. URL: https://www.designbombs.com/strikingly-review/. (Дата звернення: 18.05.23)

23. Programming in Swift. Kodeco. Дата публікації: 21.08.2018. URL: https://www.kodeco.com/5994-basic-swift-language-features-part-1. (Дата звернення: 18.05.23)

24. Python 3.11.3 documentation. Python. URL: https://docs.python.org/3/. (Дата звернення: 18.05.23)

25. Quick Start Guides. Hacking with Swift. URL: https://www.hackingwithswift.com/quick-start. (Дата звернення: 18.05.23)

26. Swift community. Swift. URL: https://swift.org/. (Дата звернення: 18.05.23)

27. Swift Playgrounds: Documentation. Developer. URL: https://developer.apple.com/documentation/swift-playgrounds/. (Дата звернення: 18.05.23)

28. Wix. URL: https://uk.wix.com/. (Дата звернення: 18.05.23)

29. Working with JSON. MDN Contributors. Дата публікації: 10.05.2023. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON. (Дата звернення: 18.05.23)

30. Xcode: Documentation. Developer. URL: https://developer.apple.com/documentation/xcode. (Дата звернення: 18.05.23)

ДОДАТКИ

ДОДАТОК А

Лістинг коду програми

A.1 WebBuilder.swift

```
import UIKit
class WebBuilder: UIView {
    public static var current: WebBuilder!
    public var assets: [Asset] = []
    public var delegate: WebBuilderControllerDelegate?
    public var webElements: [WebElement] = [
        WebElement(name: "imageElement", image: UIImage(named: "image")!, title: "image", type: ImageElement.self),
        WebElement(name: "textElement", image: UIImage(named: "text")!, title: "Текст", type: TextElement.self),
        WebElement(name: "viewElement", image: UIImage(named: "block")!, title: "Блок", type: ViewElement.self),
        WebElement(name: "zLayout", image: UIImage(named: "zStack")!, title: "зет лаяют", type: ZLayout.self),
        WebElement(name: "centerLayout", image: UIImage(named: "centering")!, title: "центеринг лаяют", type: CenterLayout.self),
        WebElement(name: "linearLayout", image: UIImage(named: "horizontalStack")!, title: "линеар лаяют", type: LinearLayout.self),]
    private var scrollView: UIScrollView!
    public lazy var selectorView: UIView = {
        let view = UIView()
        return view}()
    public lazy var mainLayout: LinearLayout = {
        let mainLayout = LinearLayout(frame: bounds, alignment: .vertical)
        mainLayout.backgroundColor = .clear
```

```
            mainLayout.setSize(ElementSettingSize(width: .matchParent, height: .wrapContent))
            let paddings = UIEdgeInsets(top: 0, left: 0, bottom: 0, right: 0)
            mainLayout.settings.append(ElementSetting(
               name: "paddings",
               parent: mainLayout,
               title: "Paddings",
               singleInput: SettingInput(value: paddings)))
            let longPress = UILongPressGestureRecognizer(target: self, action: #selector(viewMove(_:)))
            longPress.minimumPressDuration = 0.15
            mainLayout.addGestureRecognizer(longPress)
            mainLayout.addGestureRecognizer(UITapGestureRecognizer(target: self, action: #selector(viewSelected(_:))))
            return mainLayout}()

         private lazy var shadowView: UIView = {
            let view = UIView()
            view.backgroundColor = .white
            view.layer.shadowColor = UIColor.black.cgColor
            view.layer.shadowOffset = .init(width: 0, height: 0)
            view.layer.shadowRadius = 40
            view.layer.shadowOpacity = 0.2
            view.layer.masksToBounds = false
            view.clipsToBounds = false
            return view}()
         private var selectedView: WebUIElement?
         private var navigationControllerLineVisible: Bool = true
         required init?(coder aDecoder: NSCoder) {
            super.init(coder: aDecoder)
```

```
            setup()}
        override init(frame: CGRect) {
            super.init(frame: frame)
            setup()}
        private func setup() {
            scrollView = UIScrollView(frame: bounds)
            scrollView.translatesAutoresizingMaskIntoConstraints = false
            scrollView.clipsToBounds = false
            scrollView.delegate = self
            addSubview(scrollView)
            scrollView.addSubview(shadowView)
            scrollView.addSubview(mainLayout)
            let tapGesture = UITapGestureRecognizer(
                target: self, action: #selector(viewSelected(_:)))
            self.scrollView.addGestureRecognizer(tapGesture)
            WebBuilder.current = self}
        override func layoutSubviews() {
            super.layoutSubviews()
            if let navigationController = window?.rootViewController as?
UINavigationController {
                if scrollView.contentOffset.y >= 40.3 &&
!navigationControllerLineVisible {
                    navigationController.navigationBar.shadowImage =
UIColor(hexString: "#cecece").as1pt()
                    navigationControllerLineVisible = true
                } else if scrollView.contentOffset.y < 40.3 &&
navigationControllerLineVisible {
                    navigationController.navigationBar.shadowImage =
UIColor.clear.as1pt()
                    navigationControllerLineVisible = false}}
```

```
            UIView.animate(withDuration: 0.15) {
                self.mainLayout.frame = CGRect(
                    x: 0,
                    y: 0,
                    width: self.bounds.width,
                    height: self.mainLayout.getSize().height)
                self.shadowView.frame = self.mainLayout.frame
                var size = self.mainLayout.bounds.size
                size.height -= self.scrollView.safeAreaInsets.top -
                    self.scrollView.safeAreaInsets.bottom
                size.height += 250
                self.scrollView.contentSize = size}
            self.layoutSelectorView()}
        private func layoutSelectorView() {
            if let lSelectedView = self.selectedView, lSelectedView.superview != nil, let window = window {
                self.selectorView.frame = window.convert(window.bounds, to: scrollView)
                self.selectorView.frame.size.height += 1000
                let backgroundLayer: CAShapeLayer
                if let bgLayer = selectorView.layer.sublayers?.first as? CAShapeLayer {
                    backgroundLayer = bgLayer
                } else {
                    backgroundLayer = CAShapeLayer()
                    backgroundLayer.fillColor = UIColor.black.withAlphaComponent(0.65).cgColor
                    self.selectorView.layer.addSublayer(backgroundLayer)
                }
```

```
            let selectedFrame = lSelectedView.convert(lSelectedView.bounds, to:
window)
            let path = UIBezierPath(rect: window.bounds)
            path.append(UIBezierPath(roundedRect:          selectedFrame,
cornerRadius: lSelectedView.layer.cornerRadius).reversing())
            backgroundLayer.path = path.cgPath}}
        override func updateConstraints() {
          super.updateConstraints()
          // Scroll view
          scrollView.bottomAnchor.constraint(equalTo:  bottomAnchor).isActive
= true
          scrollView.rightAnchor.constraint(equalTo:  rightAnchor).isActive  =
true
          scrollView.leftAnchor.constraint(equalTo: leftAnchor).isActive = true
          scrollView.topAnchor.constraint(equalTo: topAnchor).isActive = true}
        private var movingView: BaseWebUIElement?
        private var startLocation: CGPoint = .zero
        private var initialScale: CGAffineTransform?
        private var lastLayout: WebUILayout?
        private var bottomSheetLastView: UIView?
        @objc func viewMove(_ gesture: UILongPressGestureRecognizer) {
          switch gesture.state {
          case .began:
            var        lSelectedView        =        findNearestElement(forPoint:
gesture.location(in: self))
            if lSelectedView == mainLayout {
              return}
            let tempSelectedLayout = (selectedView as? WebUILayout) ??
mainLayout
            while (lSelectedView != tempSelectedLayout as UIView)
```

```
                && lSelectedView.parent != nil
                && (lSelectedView.parent! != tempSelectedLayout as UIView)
                && lSelectedView.parent! != mainLayout as UIView {
                    lSelectedView = lSelectedView.parent!}
            let size = lSelectedView.getSize()
            let location = gesture.location(in: lSelectedView)
            var marginsFromCenter = CGPoint()
            marginsFromCenter.x = location.x - lSelectedView.frame.width / 2
            marginsFromCenter.y = location.y - lSelectedView.frame.height / 2
            initialScale = lSelectedView.transform
            startLocation = gesture.location(in: self)
            lSelectedView.removeFromParent()
            addSubview(lSelectedView)
            lSelectedView.frame.size = size
            // Setup view
            movingView = lSelectedView
            movingView!.frame.origin.x = (startLocation.x -
movingView!.frame.width / 2) - marginsFromCenter.x
            movingView!.frame.origin.y = (startLocation.y -
movingView!.frame.height / 2) - marginsFromCenter.y
            UIView.animate(
                withDuration: 0.2,
                delay: 0,
                usingSpringWithDamping: 0.5,
                initialSpringVelocity: 5,
                options: .curveEaseInOut,
                animations: {
                    lSelectedView.transform =
lSelectedView.transform.scaledBy(x: 1.1, y: 1.1)
                    lSelectedView.alpha = 0.7
```

```
                lSelectedView.layer.shadowColor = UIColor.black.cgColor
                lSelectedView.layer.shadowRadius = 4
                lSelectedView.layer.shadowOpacity = 0.3
                lSelectedView.layer.shadowOffset = .zero
                lSelectedView.layoutSubviews()
            }, completion: nil)
        UIView.animate(withDuration: 0.15, animations: {
            self.selectorView.layer.opacity = 0})
        UIImpactFeedbackGenerator(style: .medium).impactOccurred()
    case .changed:
        if gesture.location(in: self) != startLocation, let movingView = movingView, let initialScale = self.initialScale {
            let location = gesture.location(in: self)
            UIView.animate(withDuration: 0.1) {
                movingView.frame.origin.x = (location.x - movingView.frame.width / 2)
                movingView.frame.origin.y = (location.y - movingView.frame.height / 2)
                movingView.transform = initialScale.scaledBy(x: 0.8, y: 0.8)
                movingView.layoutSubviews()}
            // Find nearest layout
            let nearestLayout = findNearestLayout(forPoint: location)
            nearestLayout.showPlaceForMovingElement(at: convert(movingView.frame, to: nearestLayout))
            if let lastLayout = lastLayout, (nearestLayout as UIView) != (lastLayout as UIView) {
                lastLayout.hidePlaceForMovingElement()}
            lastLayout = nearestLayout}
    case .ended:
        if let movingView = movingView, let initialScale = self.initialScale {
```

```
                let location = gesture.location(in: self)
                UIView.animate(withDuration: 0.1, animations: {
                    // Reset view
                    movingView.alpha = 1
                    movingView.layer.shadowRadius = 0
                    movingView.layer.shadowColor = nil
                    movingView.transform = initialScale
                }) { _ in
                    // Find new frame
                    let nearestLayout = self.findNearestLayout(forPoint: location)
                    let frameInLayout = self.convert(movingView.frame, to:
nearestLayout)
                    let newFrame =
nearestLayout.findFrameForMovingElement(element: movingView, atFrame:
frameInLayout, justCreated: false)
                    // Move view to new frame
                    UIView.animate(withDuration: 0.35, delay: 0,
usingSpringWithDamping: 0.85,
                                   initialSpringVelocity: 5, options: .curveEaseInOut,
animations: {
                        movingView.frame = nearestLayout.convert(newFrame, to:
self)
                        movingView.layoutSubviews()
                    }) { _ in
                        nearestLayout.addElement(movingView, frame:
frameInLayout, justCreated: false)
                        self.movingView = nil
                        // Selector view
                        UIView.animate(withDuration: 0.15, animations: {
                            self.selectorView.layer.opacity = 1})}}}
```

```
            default:
                break}}
        @objc func viewSelected(_ gesture: UITapGestureRecognizer) {
            var lSelectedView = findNearestElement(forPoint: gesture.location(in:
self))
            if lSelectedView == mainLayout {
                if selectedView != nil {
                    self.deselectElement()}
                return}
            let tempSelectedLayout = (selectedView as? WebUILayout) ??
mainLayout
            while (lSelectedView != tempSelectedLayout as UIView)
                && lSelectedView.parent != nil
                && (lSelectedView.parent! != tempSelectedLayout as UIView)
                && lSelectedView.parent! != mainLayout as UIView {
                lSelectedView = lSelectedView.parent!}
            if selectedView == lSelectedView as UIView {
                return}
            UIImpactFeedbackGenerator(style: .medium).impactOccurred()
            if selectorView.superview == nil {
                self.selectorView.alpha = 0
                scrollView.addSubview(selectorView)}
            self.selectedView = lSelectedView
            self.layoutSelectorView()
            UIView.animate(withDuration: 0.25, animations: {
                self.selectorView.alpha = 1})
            delegate?.webBuilder(didSelect: lSelectedView)}
        public func deselectElement() {
            guard selectedView != nil else { return }
            selectedView = nil
```

```
            delegate?.webBuilderDidDeselect()
            UIView.animate(withDuration: 0.3, animations: {
               self.selectorView.layer.opacity = 0
            }, completion: { _ in
               self.selectorView.removeFromSuperview()})}
         public func findNearestLayout(forPoint point: CGPoint) -> WebUILayout
{
            var lSelectedLayout: WebUILayout = mainLayout
            while(true) {
               let location = convert(point, to: lSelectedLayout)
               let hit = lSelectedLayout.hitTest(location, with: nil)
               if let hitView = hit as? WebUILayout {
                  if (hitView as UIView) != (lSelectedLayout as UIView) {
                     lSelectedLayout = hitView
                  } else {
                     break}
               } else if let hitView = hit as? WebUIElement,
                     let hitSuperview = hitView.superview?.superview as?
WebUILayout,
                     (hitSuperview as UIView) != (lSelectedLayout as UIView) {
                  lSelectedLayout = hitSuperview
               } else if let hitSuperview = hit?.superview as? WebUILayout {
                  lSelectedLayout = hitSuperview
                  break
               } else {
                  break}}
            return lSelectedLayout}
         public func findNearestElement(forPoint point: CGPoint) ->
BaseWebUIElement {
            selectorView.isHidden = true
```

```
            var nearestElement: UIView = mainLayout
            while(true) {
                let location = convert(point, to: nearestElement)
                let hit = nearestElement.hitTest(location, with: nil)
                if let hitView = hit, hitView != nearestElement {
                    nearestElement = hitView
                } else {
                    if !(nearestElement is WebUIElement), let superview =
nearestElement.superview as? BaseWebUIElement {
                        nearestElement = superview}
                    break}}
            selectorView.isHidden = false
            return nearestElement as! BaseWebUIElement}

        public func serialize() -> BuilderBundle {
            let encoder = JSONEncoder()
            let data = try? encoder.encode(mainLayout)
            return BuilderBundle(pageData: data ?? Data(), assets: assets)}
        public func load(bundle: BuilderBundle) {
            self.assets = bundle.assets
            let decoder = JSONDecoder()
            guard let data = bundle.pageData,
                let baseElement = try? decoder.decode(
                    BaseWebUIElement.self,
                    from: data),
                let element = baseElement.mirror as? LinearLayout
            else { return }
            mainLayout.elements = element.elements
            mainLayout.layoutSubviews()}
        public func copySelected() -> String {
```

```
            let encoder = JSONEncoder()
            if let selectedView = selectedView as? BaseWebUIElement,
              let data = try? encoder.encode(selectedView),
              let result = String(data: data, encoding: .utf8) {
               return result}
            return ""}
          public func paste(json: String) {
            let decoder = JSONDecoder()
            if let data = json.data(using: .utf8), let element = try? decoder.decode(BaseWebUIElement.self, from: data).mirror {
              let selectedLayout = selectedView as? WebUILayout ?? selectedView?.parent ?? mainLayout
              selectedLayout.elements.append(element)
              selectedLayout.layoutSubviews()}}}
        extension Encodable {
          func toJSONData() -> Data? { try? JSONEncoder().encode(self) }}
        extension WebBuilder: UIScrollViewDelegate {
          func scrollViewDidScroll(_ scrollView: UIScrollView) {
            setNeedsLayout()}}
```

A.2 NewProjectView.swift

```
        import UIKit
        protocol InputView: UIView {
          func getValue() -> Any?
          func highlightError()}
        class NewProjectView: UIView {
          struct Settings {
            static let exitButtonSize: CGFloat = 30
            static let outerPaddings: CGFloat = 20
            static let inputViewPaddings: CGFloat = 16
```

```
    static let inputViewHeight: CGFloat = 65}
enum Input {
    case text(placeholder: String, defaultValue: String = "", required: Bool =
true)}
public var completion: (([Any?]) -> Void)?
public var onCancel: (() -> Void)?
public var title: String? {
    didSet {
        titleLabel.text = title}}
public var inputs: [Input] = [] {
    didSet {
        var newViews: [InputView] = []
        for input in inputs {
            switch input {
            case let .text(placeholder, defaultValue, _):
                let view = NewProjectInputTextView.instantiate(
                    placeholder: placeholder,
                    defaultValue: defaultValue)
                newViews.append(view)}}
        self.views = newViews}}
private var navigationBackgroundView: UIVisualEffectView = {
    let view = UIVisualEffectView(effect: UIBlurEffect(style: .regular))
    return view}()
private var titleLabel: UILabel = {
    let label = UILabel()
    label.font = .systemFont(ofSize: 20, weight: .medium)
    return label}()
private var exitButton: UIButton = {
    let button = UIButton(type: .close)
    return button}()
```

```
        private var mainScrollView: UIScrollView = {
            let scrollView = UIScrollView()
            return scrollView}()
        private var views: [InputView] = [] {
            didSet {
                subviews.forEach { $0.removeFromSuperview() }
                for view in views {
                    mainScrollView.addSubview(view)}}}
        private var completeButton: UIButton = {
            let button = UIButton(type: .system)
            button.backgroundColor = UIColor.systemGray5
            button.layer.cornerRadius = 10
            button.setTitle("Complete", for: .normal)
            button.setTitleColor(.label, for: .normal)
            return button}()
        override init(frame: CGRect) {
            super.init(frame: frame)
            setup(title: "New Project", inputs: [])}
        required init?(coder: NSCoder) {
            super.init(coder: coder)
            setup(title: "New Project", inputs: [])}
        init(title: String, inputs: [Input], completion: @escaping ([Any?]) -> Void,
onCancel: @escaping () -> Void) {
            super.init(frame: .zero)
            self.completion = completion
            self.onCancel = onCancel
            setup(title: title, inputs: inputs)}
        private func setup(title: String, inputs: [Input]) {
            self.title = title
            self.inputs = inputs
```

```
        completeButton.addTarget(self, action:
#selector(completeButtonTapped), for: .touchUpInside)
        exitButton.addTarget(self, action: #selector(exitButtonTapped), for:
.touchUpInside)
        addSubview(mainScrollView)
        addSubview(navigationBackgroundView)
        navigationBackgroundView.contentView.addSubview(titleLabel)
        navigationBackgroundView.contentView.addSubview(exitButton)
        mainScrollView.addSubview(completeButton)}
    @objc private func completeButtonTapped() {
        let uncompletedInputViews = inputs.enumerated()
            .map { (input: $0.element, view: views[$0.offset]) }
            .filter { isInputCompleted($0.input, view: $0.view) }
            .map { $0.view }
        if uncompletedInputViews.count > 0 {
            uncompletedInputViews.forEach { $0.highlightError() }
            highlightError()
        } else {
            let feedbackGenerator = UIImpactFeedbackGenerator(style: .light)
            feedbackGenerator.impactOccurred()
            endEditing(true)
            completion?(views.map { $0.getValue() })}}
    @objc private func exitButtonTapped() {
        let feedbackGenerator = UIImpactFeedbackGenerator(style: .light)
        feedbackGenerator.impactOccurred()
        endEditing(true)
        onCancel?()}
    private func isInputCompleted(_ input: Input, view: InputView) -> Bool {
        switch input {
        case let .text(_, _, required) where required:
```

```
                return view.getValue() == nil
            default:
                return false}}
        private func highlightError() {
            let shakeAnimation = CABasicAnimation(keyPath: "position.x")
            shakeAnimation.duration = 0.07
            shakeAnimation.repeatCount = 2
            shakeAnimation.autoreverses = true
            shakeAnimation.fromValue = completeButton.center.x - 3
            shakeAnimation.toValue = completeButton.center.x + 3
            completeButton.layer.add(shakeAnimation, forKey:
shakeAnimation.keyPath)
            let feedbackGenerator = UINotificationFeedbackGenerator()
            feedbackGenerator.notificationOccurred(.error)}
        override func layoutSubviews() {
            super.layoutSubviews()
            mainScrollView.frame = bounds
            mainScrollView.contentSize.height = intrinsicContentSize.height
            navigationBackgroundView.frame = CGRect(
                x: 0, y: 0, width: bounds.width, height: 60)
            titleLabel.frame = CGRect(
                x: Settings.outerPaddings,
                y: Settings.outerPaddings,
                width: bounds.width - Settings.exitButtonSize -
Settings.outerPaddings * 2,
                height: 35)
            exitButton.frame = CGRect(
                x: titleLabel.frame.maxX,
                y: titleLabel.frame.minY,
                width: Settings.exitButtonSize,
```

```
            height: Settings.exitButtonSize)
        var lastFrame: CGRect = CGRect(
            x: Settings.outerPaddings,
            y: titleLabel.frame.maxY,
            width: bounds.width - Settings.outerPaddings * 2,
            height: 0)
        for view in views {
            let newFrame = CGRect(
                x: lastFrame.origin.x,
                y: lastFrame.maxY + Settings.inputViewPaddings,
                width: lastFrame.width,
                height: Settings.inputViewHeight)
            view.frame = newFrame
            lastFrame = newFrame}
        completeButton.frame = CGRect(
            x: lastFrame.origin.x,
            y: lastFrame.maxY + Settings.inputViewPaddings,
            width: lastFrame.width,
            height: 50)}
    override var intrinsicContentSize: CGSize {
        let titleLabelHeight: CGFloat = 35
        let inputViewsHeight: CGFloat = (Settings.inputViewPaddings +
Settings.inputViewHeight) * CGFloat(views.count)
        let completebuttonHeight: CGFloat = Settings.inputViewPaddings + 50
        let finalHeight: CGFloat = titleLabelHeight + inputViewsHeight +
completebuttonHeight + Settings.outerPaddings * 2
        return CGSize(width: 450, height: finalHeight)}}
```

A.3 SitesViewController.swift

```
import UIKit
```

```
class SitesPhoneViewController: UIViewController, UITableViewDelegate,
UITableViewDataSource, ProjectsViewControllerProtocol {
    struct Settings {
    public static let actionViewVerticalPaddings: CGFloat = 20}
    @IBOutlet var tableView: UITableView!
    @IBOutlet var noProjectsView: UIView!
    @IBOutlet var actionView: UIView!
    @IBOutlet var actionViewHeight: NSLayoutConstraint!
    @IBOutlet var actionViewBottomMargin: NSLayoutConstraint!
    @IBOutlet var actionButtonsView: UIView!
    @IBOutlet var newPageButton: ScaleButtonView!
    @IBOutlet var newSiteButton: ScaleButtonView!
    @IBOutlet var importProjectButton: ScaleButtonView!
    private var mainRefreshControl: UIRefreshControl = {
      let refreshControl = UIRefreshControl()
      return refreshControl}()
    private var newProjectView: NewProjectView?
    private var selectedProjectController: ProjectController?
     private var loadedProjects: [LayoutProjectBase] = []
    var projectsDataProvider: ProjectsDataProviderProtocol?
    var onProjectSelected: ((LayoutProject) -> Void)?
    override func viewDidLoad() {
      super.viewDidLoad()
      mainRefreshControl.addTarget(self, action: #selector(reloadData), for:
.valueChanged)
      tableView.refreshControl = mainRefreshControl
      newPageButton.onTap = newPageButtonTapped
      newSiteButton.onTap = newSiteButtonTapped
      importProjectButton.onTap = importProjectButtonTapped
```

```
            NotificationCenter.default.addObserver(self, selector:
#selector(keyboardWillShow), name:
UIResponder.keyboardWillShowNotification, object: nil)
            NotificationCenter.default.addObserver(self, selector:
#selector(keyboardWillHide), name: UIResponder.keyboardWillHideNotification,
object: nil)
            // For buttons
            tableView.contentInset.bottom = 250
            tableView.delegate = self
            tableView.dataSource = self
            self.reloadData()
        private func newPageButtonTapped() {
            self.newProjectView = NewProjectView(
                title: "New Page",
            inputs: [.text(placeholder: "Title", defaultValue: "")],
                completion: createNewPage(inputValues:),
                onCancel: hideNewProjectView)
            self.showNewProjectView()}
        private func newSiteButtonTapped() {
            self.newProjectView = NewProjectView(
                title: "New Site",
                inputs:
    .text(placeholder: "Soon!", defaultValue: "Try pages instead"),
    .text(placeholder: "Choose your fighter!"),
    .text(placeholder: "REQUIRED input", required: true),
    .text(placeholder: "You still here?", defaultValue: "Yes"),],
                completion: createNewSite(inputValues:),
                onCancel: hideNewProjectView)
            self.showNewProjectView()}
        private func importProjectButtonTapped() {
```

```
            let alert = UIAlertController(title: "Error", message: "Soon! Try pages instead.", preferredStyle: .alert)
            alert.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
            present(alert, animated: true)}
        private func createNewPage(inputValues: [Any]) {
      guard let title = inputValues[0] as? String else { return }
            let newProject = PageProject(title: title)
            projectsDataProvider?.createProject(newProject)
            hideNewProjectView()
            reloadData()
            openPage(newProject)}
        private func createNewSite(inputValues: [Any]) {
            hideNewProjectView()}
        private func showNewProjectView() {
            guard let newProjectView = self.newProjectView else { return }
            let maxActionViewHeight = view.bounds.height -
                Settings.actionViewVerticalPaddings -
                view.safeAreaInsets.bottom -
                view.safeAreaInsets.top
            let newProjectSize = newProjectView.intrinsicContentSize
            let newProjectHeight = min(maxActionViewHeight, newProjectSize.height)
                    actionView.addSubview(newProjectView)
            newProjectView.alpha = 0
            newProjectView.backgroundColor = .systemGray6
            newProjectView.frame.size.width = actionView.frame.width
            newProjectView.frame.size.height = newProjectHeight
            newProjectView.layoutIfNeeded()
            actionViewHeight.constant = newProjectHeight
            actionView.backgroundColor = .systemGray6
```

```
    UIView.animate(
        withDuration: 0.35,
        delay: 0,
        usingSpringWithDamping: 0.8,
        initialSpringVelocity: 1,
        options: .curveEaseInOut,
        animations: {
            newProjectView.alpha = 1
            self.actionButtonsView.alpha = 0
            self.view.layoutIfNeeded()          })    }
private func hideNewProjectView() {
    guard let newProjectView = self.newProjectView else { return }
    UIView.animate(
        withDuration: 0.35,
        delay: 0,
        usingSpringWithDamping: 0.8,
        initialSpringVelocity: 1,
        options: .curveEaseInOut,
        animations: {
            self.actionViewHeight.constant = 0
            self.actionButtonsView.alpha = 1
            self.actionView.backgroundColor = .clear
            newProjectView.alpha = 0
            self.view.layoutIfNeeded()
        }, completion: { _ in
            newProjectView.removeFromSuperview()
            self.newProjectView = nil          })    }
private func openPage(_ page: PageProject) {
    self.onProjectSelected?(page)    }
@objc func keyboardWillShow(notification: NSNotification) {
```

```
            guard let info = notification.userInfo,
                let keyboardFrame = info[UIResponder.keyboardFrameEndUserInfoKey] as? CGRect,
                let newProjectView = newProjectView
                else { return }
            navigationController?.setNavigationBarHidden(true, animated: true)
            UIView.animate(withDuration: 0.2, animations: {
              let bottomMargin = Settings.actionViewVerticalPaddings + keyboardFrame.height
                self.actionViewBottomMargin.constant = bottomMargin
                self.actionViewHeight.constant = newProjectView.intrinsicContentSize.height
                self.view.layoutIfNeeded()       })    }
        @objc func keyboardWillHide(notification: NSNotification) {
            navigationController?.setNavigationBarHidden(false, animated: true)
            UIView.animate(withDuration: 0.2) {
                self.actionViewBottomMargin.constant = Settings.actionViewVerticalPaddings
                self.actionViewHeight.constant = min(
                  self.actionViewHeight.constant,
                self.newProjectView?.intrinsicContentSize.height ?? 0)
                self.view.layoutIfNeeded()       }    }
        override func viewDidLayoutSubviews() {
            super.viewDidLayoutSubviews()
            self.newProjectView?.frame.size.height = self.actionView.frame.height
            self.newProjectView?.setNeedsLayout()
            self.newProjectView?.layoutIfNeeded()    }
        @objc func reloadData() {
            projectsDataProvider?.getProjects { projects in
                DispatchQueue.main.async {
```

```
                if #available(iOS 13, *) {
                    var insertIndecies: [Int] = []
                    var removeIndecies: [Int] = []
        for change in projects.difference(from: self.loadedProjects) {
                        if case let .insert(offset, _, _) = change {
                            insertIndecies.append(offset)}
                        if case let .remove(offset, _, _) = change {
                            removeIndecies.append(offset) }    }
                    self.loadedProjects = projects
                    if insertIndecies.count > 0 {
                        self.tableView.insertRows(at:        insertIndecies.map        {
IndexPath(row: $0, section: 0) }, with: .automatic)}
                    if removeIndecies.count > 0 {
                        self.tableView.deleteRows(at:       removeIndecies.map       {
IndexPath(row: $0, section: 0) }, with: .automatic)              }             } else {
                    self.loadedProjects                    =                    projects
self.tableView.reloadSections(IndexSet(arrayLiteral: 0), with: .automatic)          }
                self.mainRefreshControl.endRefreshing()
                self.updateBackgroundMessage()   }       }    }
            private func updateBackgroundMessage() {
                if self.loadedProjects.count == 0 {
                    self.noProjectsView.isHidden = false
                    self.tableView.isHidden = true
                } else {
                    self.noProjectsView.isHidden = true
                    self.tableView.isHidden = false       }    }
            func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
                return loadedProjects.count    }
```

```
        func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
            guard let cell = tableView.dequeueReusableCell(
                withIdentifier: "siteCell",
                for: indexPath) as? SiteTableViewCell
            else {
               preconditionFailure("Wrong cell identifier"       }
            let project = loadedProjects[indexPath.row].project
            if let siteProject = project as? SiteProject {
               cell.titleLabel.text = siteProject.title
               cell.iconView.image = UIImage(systemName: "link.circle.fill")
               cell.domainLabel.text = siteProject.domain
            } else if let pageProject = project as? PageProject {
               cell.titleLabel.text = pageProject.title
               cell.iconView.image = UIImage(systemName: "doc")
               cell.domainLabel.text = "Page project"       }
            return cell    }
        func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) -> CGFloat {
            return 60    }
        func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
            tableView.deselectRow(at: indexPath, animated: true)
            let project = loadedProjects[indexPath.row]
            if let page = project.project as? PageProject {
               openPage(page)       }    }}
```

A.4 SharePageViewController.swift

```
import UIKit
class SharePageViewController: UIViewController {
```

```
        @IBOutlet var publicLinkView: UIView!
        @IBOutlet var publicLinkLabel: UILabel!
        @IBOutlet var getPublicLinkButton: UIButton!
        private var project: PageProject!
        public static func instantiate(page: PageProject) -> SharePageViewController {
            let viewController = UINib(nibName: "SharePageView", bundle: .main)
                .instantiate(withOwner: self, options: nil)
                .first as! SharePageViewController
            viewController.project = page
            viewController.setup()
            return viewController    }
        private func setup() {
            updatePublicLink()
            let tapGesture = UITapGestureRecognizer(target: self, action: #selector(didTapPublicLinkView))
            publicLinkView.addGestureRecognizer(tapGesture)    }
        private func updatePublicLink() {
            if let publicLink = project.publicLink {
                publicLinkLabel.text = publicLink
      getPublicLinkButton.setTitle("Upload changes", for: .normal)
            } else {
      getPublicLinkButton.setTitle("Get public link", for: .normal)}}
        @IBAction func didTapGetPublicLinkButton() {
            if project.publicLink == nil {getPublicLinkButton.setTitle("Uploading..", for: .normal)
                project.upload { result in
                    switch result {
                    case .success(_):
                        self.updatePublicLink()
```

```
                    case .failure(_):
                        self.updatePublicLink()
                        let alert = UIAlertController(title: "Error", message: "Unable to upload page, try again later.", preferredStyle: .alert)
                        alert.addAction(UIAlertAction(title: "OK", style: .cancel, handler: nil))
                        self.present(alert, animated: true, completion: nil)                }
}
            } else {
                getPublicLinkButton.setTitle("Uploading..", for: .normal)
                project.update { result in
                    switch result {
                    case .success(_):
                        self.updatePublicLink()
                    case .failure(_):
                        self.updatePublicLink()
                        let alert = UIAlertController(title: "Error", message: "Unable to upload page, try again later.", preferredStyle: .alert)
                        alert.addAction(UIAlertAction(title: "OK", style: .cancel, handler: nil))
                        self.present(alert, animated: true, completion: nil)                }
}       }    }
        @objc func didTapPublicLinkView() {
            guard let publicLinkString = project.publicLink, let publicLink = URL(string: publicLinkString) else { return }
            let openActivity = ActivityViewCustomActivity(title: "Open URL", imageName: "link") {
                UIApplication.shared.open(publicLink, options: [:], completionHandler: nil)       }
            let activityController = UIActivityViewController(
```

```
            activityItems: [publicLink],
            applicationActivities:                    [openActivity])
activityController.popoverPresentationController?.sourceView = publicLinkView
        present(activityController, animated: true, completion: nil)
    }}
```