

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Розробка бібліотеки для SSR рендерингу»

Виконав студент групи ІПЗм-22-1
ФРАНЧУК Владислав Олександрович

Керівник роботи:
КРАВЧЕНКО Світлана Миколаївна

Рецензент:
ВЛАСЕНКО Олег Васильович

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«18» вересня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: ФРАНЧУК Владислав Олександрович

Керівник роботи: КРАВЧЕНКО Світлана Миколаївна

Тема роботи: «Розробка бібліотеки для SSR рендерингу»,

затверджена Наказом закладу вищої освіти від «18» вересня 2023 р. № 479/с

Вихідні дані для роботи: бібліотека та приклад коду, який демонструє, як використовувати SSR бібліотеку в практиці.

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Кравченко С. М.	22.09.2023р.	22.09.2023р.
2	Кравченко С. М.	22.09.2023р.	22.09.2023р.
3	Кравченко С. М.	22.09.2023р.	22.09.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання технічного завдання. Опрацювання літературних джерел.	3 квітня 2023 – 20 квітня 2023	
2	Проектування структури бібліотеки	21 квітня 2023 – 12 травня 2023	
3	Написання програмного коду	13 травня 2023 – 25 травня 2023	
4	Тестування	26 травня 2023 – 4 червня 2023	
5	Захист		

Студент

(підпис)

Керівник

(підпис)

РЕФЕРАТ

Магістерська робота складається зі вступу, 3 розділів, висновків, списку використаних джерел на 28 позиції (60 сторінка), 3 додатків та 34 ілюстрацій у тексті. Повний обсяг роботи становить 67 сторінок, з них 62 сторінки основного тексту.

Мета розробки Server-Side Render (SSR) бібліотеки для Flutter полягає в створенні програмного засобу, який надає можливість використовувати SSR техніку у мобільних додатках, розроблених з використанням фреймворку Flutter. Головною метою є поєднання переваг Flutter, зокрема кросплатформенності та ефективної розробки інтерфейсу, з можливістю використання SSR для поліпшення продуктивності та SEO-дружності мобільних додатків.

В роботі визначено основні завдання на розробку системи, проаналізовано аналоги розробленої системи. Наведено сценарії роботи бібліотеки, розглянуті аналоги, опис алгоритмів взаємодії, також наводиться результати тестування бібліотеки.

КЛЮЧОВІ СЛОВА: FLUTTER, SERVER-SIDE RENDER, ІНТЕРНЕТ, АВТОМАТИЗАЦІЯ, DART, ДЕРЕВО ОБ'ЄКТІВ.

ABSTRACT

The master's thesis consists of an introduction, 3 chapters, conclusions, a list of used sources on 28 positions (60 pages), 3 appendices and 34 illustrations in the text. The full volume of the work is 67 pages, of which 62 pages are the main text.

The goal of developing the Server-Side Render (SSR) library for Flutter is to create a software tool that provides an opportunity to use the SSR technique in mobile applications developed using the Flutter framework. The main goal is to combine the advantages of Flutter, in particular cross-platform and efficient interface development, with the possibility of using SSR to improve the performance and SEO-friendliness of mobile applications.

The paper defines the main tasks for the development of the system, and analyzes the analogues of the developed system. Scenarios of the library's work are given, analogues are considered, a description of interaction algorithms is given, and the results of library testing are also given.

KEY WORDS: FLUTTER, SERVER-SIDE RENDER, INTERNET, AUTOMATION, DART, OBJECT TREE.

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ SSR ТА ФРЕЙМВОРКУ ДЛЯ РЕАЛІЗАЦІЇ	10
1.1 ПОСТАНОВКА ЗАДАЧІ.....	10
1.2 ОСНОВНІ КОНЦЕПТИ SSR БІБЛІОТЕК.....	10
1.3 АНАЛІЗ АНАЛОГІВ ПРОГРАМНОГО ПРОДУКТУ	14
1.4 МОБІЛЬНА РОЗРОБКА ЗА ДОПОМОГОЮ FLUTTER FRAMEWORK	22
Висновки до першого розділу	25
РОЗДІЛ 2. РОЗРОБКА БІБЛІОТЕКИ ТА ЇЇ ІНТЕГРАЦІЯ В ПРОЕКТ	27
2.1 ПІЗНАННЯ ДЕРЕВА ВІДЖЕТІВ В FLUTTER.....	27
2.2 ЗАГАЛЬНІ ПІДХОДИ ДЛЯ РОЗРОБКИ ДОДАТКІВ НА БАЗІ FLUTTER	29
2.3 ЗНАННЯ ТА ВИСНОВКИ З ІНТЕГРАЦІЇ ПОДІБНОЇ БІБЛІОТЕКИ	36
2.4 РЕЗУЛЬТАТ ОПРАЦЬОВАНОГО МАТЕРІАЛУ.....	40
Висновки до другого розділу	43
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З БІБЛІОТЕКОЮ.....	45
3.1 СТРУКТУРА БІБЛІОТЕКИ ТА ШЛЯХИ ВИКОРИСТАННЯ.	45
3.2 ДОКУМЕНТАЦІЯ БІБЛІОТЕКИ.	46
3.3 ПРИКЛАД ВИКОРИСТАННЯ БІБЛІОТЕКИ	49
3.4 ІНТЕГРАЦІЯ В ПРОЕКТ.....	51
3.5 ТЕСТУВАННЯ КОМПОНЕНТІВ БІБЛІОТЕКИ	55
Висновки до третього розділу	57
ВИСНОВОК.....	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	60
ДОДАТКИ.....	63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – Операційна система.

ПЗ – Програмне забезпечення.

SSR – Server-side render – візуалізація на стороні сервера.

MVVM – Model-View-ViewModel – шаблон проєктування, що застосовується під час проєктування архітектури застосунків (додатків).

CSR – Client-side render – візуалізація на стороні клієнта.

HTML – HyperText Markup Language – стандартизована мова розмітки

CSS – Cascading Style Sheets – мова стилю сторінок

SEO – пошукова оптимізація сайту

ВСТУП

Актуальність дослідження. Сучасний світ мобільних додатків ставить високі вимоги до продуктивності та швидкості завантаження. Кросплатформенна мобільна розробка з використанням фреймворку Flutter стає все більш популярною, однак для досягнення оптимальної продуктивності та забезпечення зручного інтерфейсу користувачам важливо використовувати техніку Server-Side Rendering (SSR).

SSR дозволяє генерувати код верстки на сервері та передавати його клієнту, що призводить до швидкого завантаження сторінок та поліпшення індексації пошуковими системами. З огляду на зростання кількості мобільних додатків на базі Flutter, інтеграція SSR стає важливою задачею для розробників.

Актуальність дослідження полягає у необхідності розробки SSR бібліотеки для фреймворку Flutter, що дозволить розробникам ефективно використовувати переваги цього фреймворку та одночасно забезпечувати високу продуктивність та швидке завантаження мобільних додатків.

Метою даної кваліфікаційної роботи є дослідження, розробка та інтеграція SSR бібліотеки для фреймворку Flutter з метою покращення продуктивності мобільних додатків, створених з використанням цього фреймворку. Робота також спрямована на створення інструментів та документації для зручного використання SSR техніки в мобільних додатках на Flutter, а також на порівняльний аналіз ефективності розробленої бібліотеки.

Завдання. Для досягнення цієї мети робота включає наступні завдання:

1. Аналіз напрямків використання ssr та фреймворку для реалізації.
2. Детальна аналіз та розробка клієнтського рендерингу.
3. Впровадження бібліотеки та оцінка результатів, включаючи використання SSR техніки в реальних мобільних додатках на Flutter.

Об'єктом дослідження є методи та засоби проектування програмного забезпечення та уніфікація процесу проектування.

Предметом дослідження є процес розробки та інтеграції Server-Side Rendering (SSR) бібліотеки для фреймворку Flutter.

Методи дослідження. В кваліфікаційній роботі було використано такі методи: порівняльний аналіз, математичне та технічне моделювання, розробка прототипів і демонстрація.

Наукова новизна полягає у розробці та дослідженні Server-Side Rendering (SSR) техніки для фреймворку Flutter, розробці інструментів для її впровадження, а також в аналізі її впливу на продуктивність мобільних додатків. Бібліотек з відкритим кодом на дану технологію немає тому проект має перспективи для розвитку і новизну в сегменті Flutter розробників.

Практичне значення полягає у підвищенні продуктивності мобільних додатків на базі Flutter, полегшенні розробки та зниженні навантаження на сервер та мережу, що сприяє покращенню якості та доступності мобільних додатків для користувачів.

Публікації. Матеріали кваліфікаційної роботи доповідались на XVI Всеукраїнська науково-практична: “Web конференція аспірантів, студентів та молодих вчених – 2023 (KICM-2022)” 21-23 березня 2023 року[16] та VI Всеукраїнська науково-технічна конференція: “Комп’ютерні технології: інновації, проблеми, рішення – 2023” 29-30 листопада 2023 року[15].

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ SSR ТА ФРЕЙМВОРКУ ДЛЯ РЕАЛІЗАЦІЇ

1.1 Постановка задачі

У сучасному світі розробки мобільних додатків на платформі Flutter, проблеми продуктивності та швидкості завантаження сторінок додатків стають все більш актуальними. Одним з можливих шляхів вирішення цих проблем є використання техніки Server-Side Rendering (SSR).

Метою цієї кваліфікаційної роботи є розробка Server-Side Render бібліотеки для фреймворку Flutter з метою покращення продуктивності мобільних додатків та їх SEO-дружності. Основними завданнями роботи є:

1. Аналіз і дослідження SSR техніки в контексті мобільних додатків на Flutter.
2. Розробка бібліотеки для SSR з урахуванням особливостей фреймворку Flutter.
3. Інтеграція та тестування розробленої бібліотеки в додатку.
4. Підготовка документації для використання бібліотеки розробниками.

Очікуваними результатами роботи є створення ефективної SSR бібліотеки, яка може бути успішно використана для розробки високопродуктивних мобільних додатків на платформі Flutter. Також передбачається розробка документації та можливість інтеграції бібліотеки в реальні проекти.

1.2 Основні концепти SSR бібліотек

Однією з задач такої бібліотеки - це універсальний інструмент, який забезпечував би гнучкий флоу для різних проектів, використовуючи одну спільну кодову базу. Для цього було прийняте рішення використовувати Server-Side Rendering (SSR), що дало змогу генерувати та відправляти клієнтам індивідуальні реєстраційні форми в залежності від їхніх потреб.

Server-Side Rendering (SSR) є важливим аспектом розвитку веб-розробки та мобільних додатків. Цей підхід має глибокі корені в історії розробки програмного забезпечення та веб-технологій. Для кращого розуміння сучасного значення SSR, важливо розглянути його історичні витoki та етапи розвитку.

Початок веб-розробки був пов'язаний із створенням статичних веб-сайтів, які складалися лише з HTML-коду. Користувачі отримували готові сторінки без можливості взаємодії з ними. За допомогою серверних мов програмування, таких як PHP, Perl та Java, почали з'являтися можливості для створення динамічних веб-сторінок, де сервер генерував HTML-код та надсилав його клієнту.

Однак реальний прорив у розвитку веб-додатків став можливим завдяки JavaScript. Він дозволив створювати динамічні ефекти та інтерактивність на веб-сторінках за допомогою CSR(Client-side render). Однак великі обсяги JavaScript-коду на стороні клієнта призводили до повільної загрузки сторінок та негативно впливали на користувацький досвід.

Client-Side Rendering (CSR) - це метод, при якому весь контент веб-сторінок рендериться на стороні клієнта, тобто в браузері користувача. Основна ідея полягає в тому, що браузер завантажує мінімальний HTML-код і джерела JavaScript, а потім відповідальний за відображення вмісту сторінки.

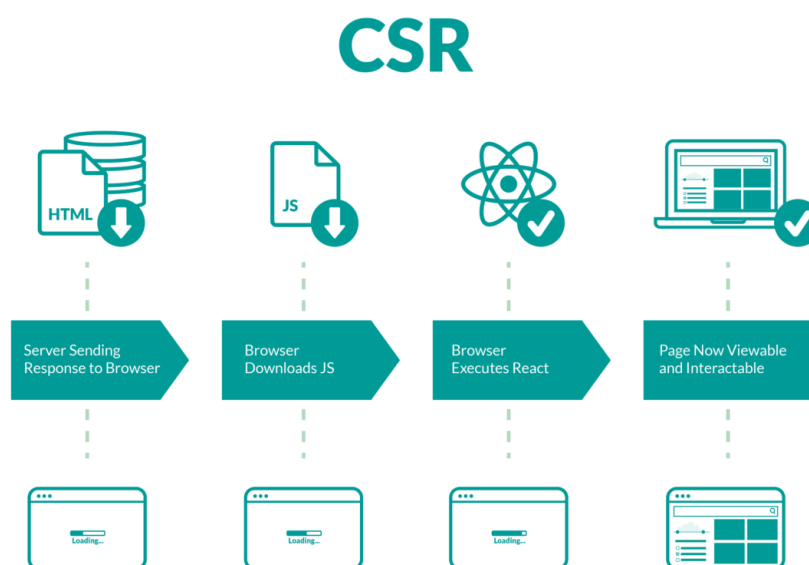


Рис. 1.1 – Принцип роботи Client-Side Rendering

Основні риси CSR включають в себе швидкість завантаження сторінок, значний обсяг JavaScript-коду, що виконується на клієнтському боці, можливі проблеми з оптимізацією для пошукових систем, а також відмінний користувацький досвід завдяки інтерактивності і динаміці сторінок.

CSR дозволяє створювати динамічні та інтерактивні веб-застосунки, проте він також може бути вимогливим до ресурсів і мати вплив на продуктивність, тому важливо враховувати ці аспекти при розробці та виборі підходу для вашого проекту.

З'явилося поняття Single Page Applications (SPA), які використовують клієнтський JavaScript для динамічної зміни вмісту сторінок без перезавантаження всієї сторінки. SPA врешті-решт стали популярними, але призводили до проблем з SEO та ресурсом на стороні клієнта.

Саме в цей момент виникає необхідність в Server-Side Rendering. SSR дозволяє генерувати вміст сторінок на сервері та надсилати готові сторінки клієнтам. Це сприяє як покращенню продуктивності, так і покращенню SEO.

Однією з ключових інновацій у світі мобільної розробки став фреймворк Flutter, який використовує SSR для рендерингу сторінок із сервера. Це відкриває нові можливості для створення швидких та динамічних мобільних додатків з покращеним користувацьким досвідом.

З історії Server-Side Rendering видно, як цей підхід став необхідним для подальшого розвитку веб-та мобільних додатків, забезпечуючи оптимальну продуктивність та якість взаємодії з користувачами.

Завдяки SSR під час завантаження сторінки реєстрації на сервері генерується вміст, який відповідає специфікаціям кожного проекту. Клієнт отримує зручний інтерфейс, який підлаштовується під його потреби. Це дозволяє нам заощадити ресурси обробки даних, уникнути непотрібних кроків у флоу реєстрації та забезпечити кожного клієнта нашого мобільного банкінгу індивідуальною та зручною реєстраційною формою.

Впровадження SSR в реєстраційні процеси стало здатним розширити можливості та функціональність нашого додатку. За допомогою динамічної

генерації наприклад реєстраційних форм на сервері, ми можемо забезпечити гнучкий підхід до реєстрації, де кожен клієнт може вибрати опції, що відповідають його потребам.

SSR дозволив створити спеціальну бібліотеку, що об'єднує різні аспекти реєстрації для різних проектів. Це значно спростило розробку та підтримку, а також дало можливість швидко реагувати на зміни та вимоги клієнтів.

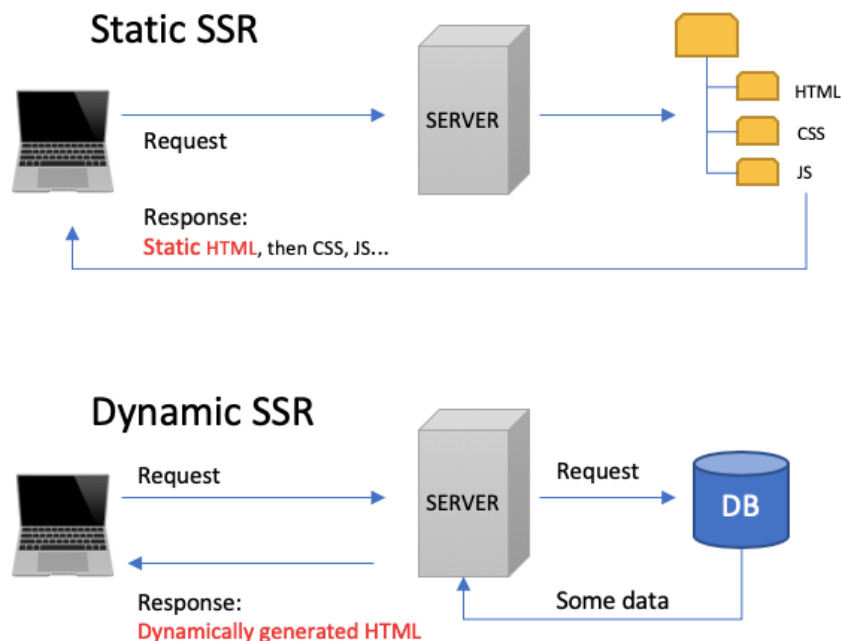


Рис. 1.2 – Процес роботи SSR

Однією з ключових переваг SSR є можливість створення індивідуальних реєстраційних форм для кожного клієнта. Ми здатні адаптувати процес під потреби різних груп користувачів, створюючи приємне та особисте враження.

Гнучкість і персоналізація є необхідними в мобільному банкінгу, де вимоги користувачів можуть значно відрізнятися. За допомогою SSR ми можемо забезпечити гармонійний баланс між загальною логікою та індивідуальним підходом до кожного клієнта.

Загалом, використання серверного рендеру для гнучкої реєстрації виявилось ключовим чинником у розвитку мобільної аплікації. Цей підхід став можливістю реалізувати візію ідеального користувацького досвіду, де кожен

клієнт може отримати унікальну та зручну форму, а компанія зберегти ресурси при розробці.

1.3 Аналіз аналогів програмного продукту

Процес Server-Side Rendering (SSR) активно використовується багатьма великими компаніями та проектами, які прагнуть покращити продуктивність, швидкість завантаження сторінок та користувацький досвід. Деякі приклади організацій та фреймворків, які використовують SSR:

Google (Angular Universal): Google використовує Angular Universal для реалізації SSR у своїх Angular додатках. Це допомагає забезпечити швидке завантаження сторінок та поліпшити SEO.

Angular Universal - це ініціатива, яка спрямована на розширення можливостей Angular для рендерингу на стороні сервера (Server-Side Rendering, SSR). Цей підхід дозволяє покращити продуктивність та SEO-оптимізацію веб-додатків, розроблених на Angular, шляхом генерації статичних HTML-сторінок на сервері перед їхнім відправленням клієнтам.

Google, як компанія, розуміє важливість високої продуктивності та швидкості завантаження веб-сторінок для забезпечення кращого досвіду користувачів. Angular Universal - один з інструментів, які допомагають досягти цієї мети.

Angular Universal від Google надає веб-розробникам зручний спосіб реалізації SSR для Angular-додатків. Це дозволяє покращити ранг сторінок у пошукових системах та забезпечити високу продуктивність завдяки передачі вже згенерованого контенту на стороні сервера.

Однією з основних переваг використання Angular Universal у Google є те, що це надає можливість впроваджувати SSR без значних змін в існуючих Angular-додатках. Розробники можуть залишити велику частину коду незмінною та просто додати SSR, що допомагає в ефективному використанні ресурсів та скороченні часу впровадження.

Усього це робить Angular Universal важливим інструментом для розробників та компаній, що працюють з Angular, особливо для тих, хто прагне покращити продуктивність та SEO-параметри своїх веб-додатків. Скріншот роботи Angular Universal у Google:

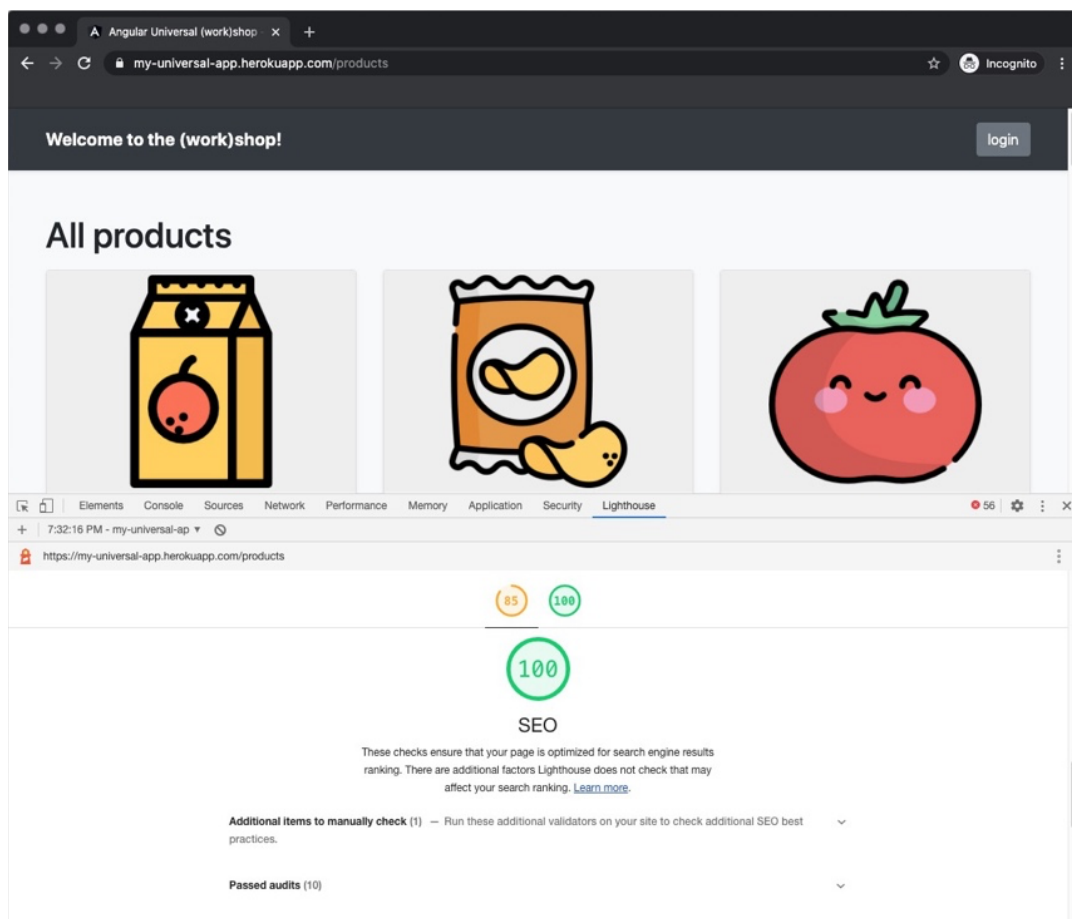


Рис. 1.3 – Тестовий проект для аналізу SEO оптимізації

Facebook (Next.js): Next.js є популярним фреймворком для розробки React додатків, що використовують SSR. Facebook використовує Next.js для створення зовнішніх та внутрішніх проектів, включаючи їхню платформу для роботи з оголошеннями. Next.js - це популярний фреймворк для розробки веб-додатків на основі React, який створено компанією Vercel (раніше ZEIT) та активно використовується Facebook для побудови своїх веб-сайтів та додатків.

Facebook використовує Next.js завдяки його високій продуктивності та можливостям SSR. Фреймворк дозволяє рендерити веб-сторінки на стороні сервера, а не на клієнтському браузері. Це призводить до зменшення часу

завантаження сторінок та покращення індексації їхнього контенту пошуковими системами, що має важливе значення для SEO.

Next.js надає функціонал для легкого налаштування SSR, а також можливість використовувати статичний генерацій контенту (Static Site Generation, SSG). Це означає, що Facebook може забезпечити високу швидкість завантаження веб-сторінок та відмінний користувацький досвід для своїх користувачів.

Додатково, Next.js має активну спільноту розробників та надає інструменти для швидкого розгортання та оновлення веб-додатків. Це робить фреймворк досить зручним і ефективним рішенням для Facebook у веб-розробці. Скріншот роботи Next.js у Facebook:

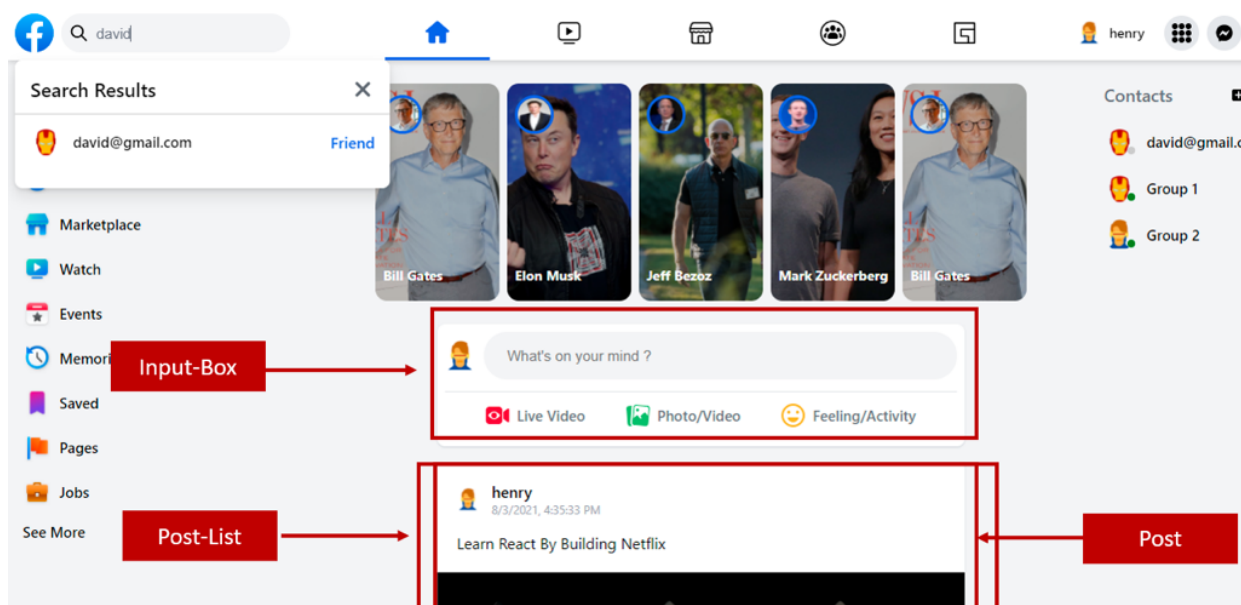


Рис. 1.4 – Скріншот з компонентами які відмальовуються з сервера

Netflix (Next.js): Netflix також використовує Next.js для створення свого веб-інтерфейсу. SSR допомагає покращити завантаження сторінок, що важливо для забезпечення приємного користувацького досвіду на їхньому веб-сайті.

На Netflix використовують Next.js для створення динамічних та ефективних веб-сайтів. Next.js - це популярний фреймворк для розробки React-додатків, який надає ряд переваг, особливо коли йдеться про масштабні веб-проекти.

Скріншот роботи Next.js у Netflix:

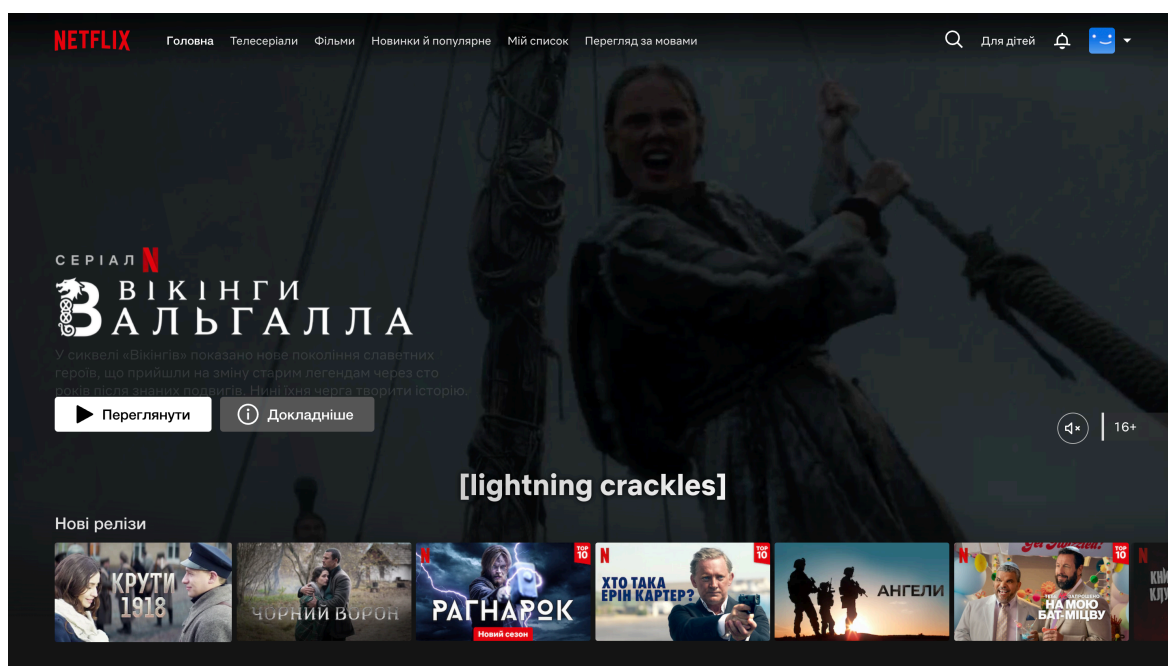


Рис. 1.5 – Рендер головної сторінки Netflix

Однією з ключових переваг Next.js є можливість використовувати Server-Side Rendering (SSR) і Static Site Generation (SSG). SSR дозволяє генерувати вміст сторінок на стороні сервера, що поліпшує швидкість завантаження сторінок і SEO. SSG дозволяє попередньо генерувати сторінки під час збирання проекту, що дозволяє збільшити продуктивність та масштабованість додатків.

Netflix використовує Next.js для різних аспектів свого веб-екосистеми, зокрема для створення сторінок, де користувачі шукають фільми та серіали, переглядають рейтинги, отримують рекомендації та багато іншого. SSR допомагає забезпечити швидку відповідь на запити користувачів та покращити їхній досвід взаємодії з платформою.

Крім того, Netflix використовує Next.js для створення статичних сторінок, які не змінюються часто, та для реалізації інших розширених функціональностей на своєму веб-сайті.

Загалом, Next.js допомагає Netflix забезпечувати швидку, стабільну та високопродуктивну роботу своїх веб-додатків, що є важливим фактором для забезпечення задоволення користувачів та успіху компанії.

Twitter (Gatsby): Twitter використовує Gatsby для розробки веб-сайту. Gatsby допомагає створювати швидкі та оптимізовані сторінки, які можуть бути

рендеровані на сервері або побудовані як статичний контент. Twitter використовує фреймворк Gatsby для створення свого веб-сайту та забезпечення високої продуктивності та якості веб-додатків. Gatsby - це популярний фреймворк для розробки веб-сайтів, який базується на React та використовує концепцію Static Site Generation (SSG).

Висока швидкість завантаження: Gatsby генерує статичні HTML-сторінки з попередньо побудованим контентом, що дозволяє забезпечити швидке завантаження сторінок. Це особливо важливо для соціальних мереж, де швидкість доступу до контенту має велике значення.

Gatsby надає потужні інструменти для оптимізації пошукової видимості веб-сайту. Twitter може легко керувати метатегами, URL-ами та іншими аспектами SEO для кожної сторінки.

Gatsby дозволяє ефективно робити запити до різних джерел даних, таких як бази даних, API або файли. Twitter може інтегрувати різні джерела даних для побудови свого веб-сайту.

Gatsby базується на концепції "віджетів", що дозволяє розробникам створювати перевикористовувані компоненти для веб-сторінок. Це полегшує розробку та підтримку коду.

Gatsby використовує статичні файли, що робить сайт менш вразливим до багатьох видів атак.

Загалом, використання Gatsby дозволяє Twitter створювати продуктивні та швидкі веб-додатки, які задовольняють потреби своїх користувачів і покращують загальний досвід взаємодії з соціальною мережею як зі сторони розробників так зі сторони користувачів.

Скріншот роботи Gatsby у Twitter:

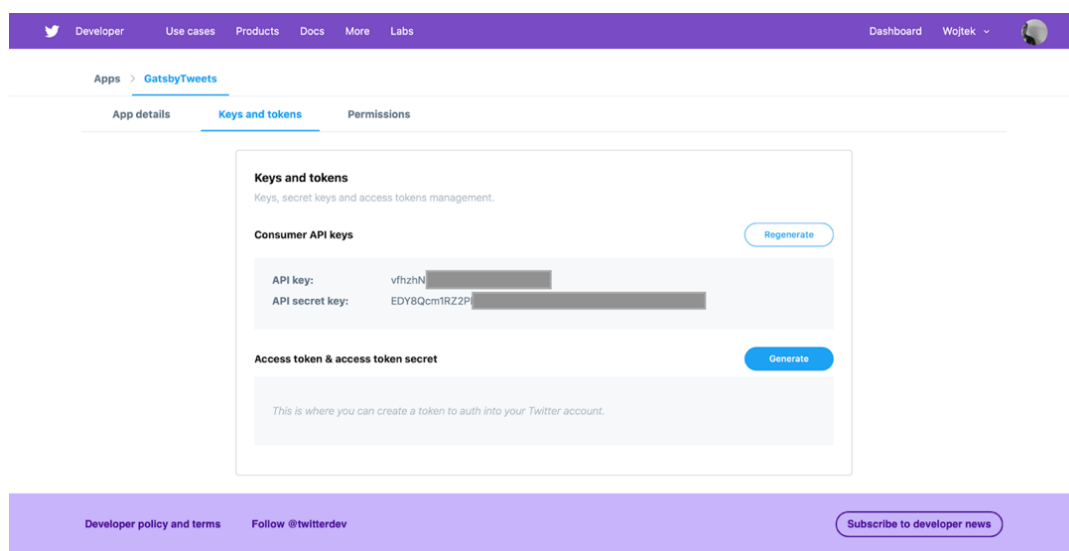


Рис 1.6 – Можливість нативної інтеграції компонентів Gatsby для розробників

Uber (Next.js): Uber використовує Next.js для своєї веб-сторінки. Використання SSR допомагає забезпечити швидке завантаження важливих елементів сторінки, що може бути важливим для клієнтів, які шукають інформацію та послуги Uber. Однією з ключових переваг Next.js для Uber є можливість серверного рендерингу (SSR), яка дозволяє вирішити проблему повільного завантаження сторінок та покращити індексацію контенту пошуковими системами. Це особливо важливо для компанії, яка надає послуги великих міст та повинна забезпечити швидку та зручну навігацію для своїх користувачів.

Також Uber використовує функціонал Next.js, пов'язаний з генерацією статичних сторінок (Static Site Generation, SSG). Це дозволяє передзавантажувати сторінки на сервері та обслуговувати запити користувачів більш ефективно, особливо при великому навантаженні на веб-сайт.

Крім того, Next.js має активну спільноту розробників та багато розширень та плагінів, що полегшують роботу з фреймворком. Це дозволяє Uber швидко розгортати нові функції та оновлення на своїх веб-сайтах та платформах.

Загалом, використання Next.js в Uber є важливою складовою успішної веб-розробки та забезпечує швидку та надійну роботу їхніх онлайн-сервісів. Скріншот роботи Next.js у Uber:

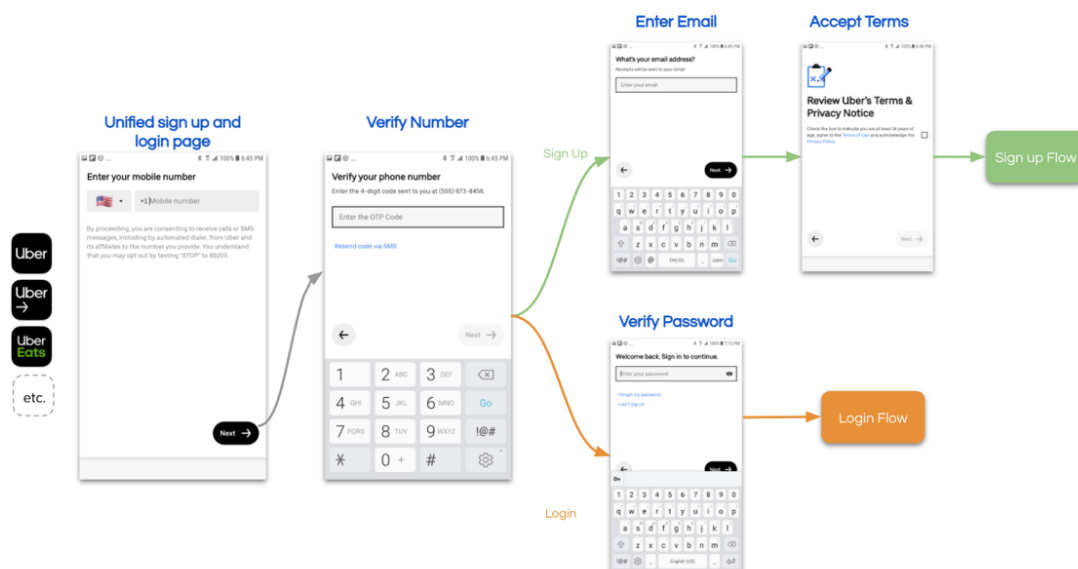


Рис. 1.7 – Флоу реєстрації яке динамічно підтягується з серверу

Shopify (Nuxt.js): Shopify, одна з найбільших платформ для онлайн-торгівлі, використовує Nuxt.js для вдосконалення продуктивності та користувацького досвіду на своїй платформі. Nuxt.js допомагає забезпечити ефективну роботу магазинів та зменшити час завантаження сторінок. Nuxt.js - це популярний фреймворк для розробки веб-додатків на основі Vue.js. Він знайшов своє застосування в компанії Shopify, яка є однією з найбільших онлайн-торгових платформ у світі. Використання Nuxt.js у Shopify має кілька ключових переваг, які допомагають поліпшити продуктивність та розвиток магазинів на платформі.

Nuxt.js дозволяє використовувати серверний рендерінг, що сприяє швидкому завантаженню сторінок та поліпшеній продуктивності магазину. Це особливо важливо для електронних магазинів, де швидкість завантаження має велике значення для користувачів і впливає на конверсію.

Окрім серверного рендерінгу, Nuxt.js підтримує статичне генерування сторінок, що дозволяє створювати статичні HTML-сторінки для поліпшення SEO та загальної продуктивності. Shopify може генерувати тисячі сторінок для продуктів, категорій та інших частин магазину, завчасно підготовлені для клієнтів.

Nuxt.js також полегшує розробку та підтримку магазинів на Shopify завдяки зручному розробничому процесу, гарячій перезавантаженню, розширенням та плагінам. Розробники можуть швидко створювати власні UI-елементи та функціональність для магазину, використовуючи екосистему Vue.js. Скріншот роботи Nuxt.js у Shopify:

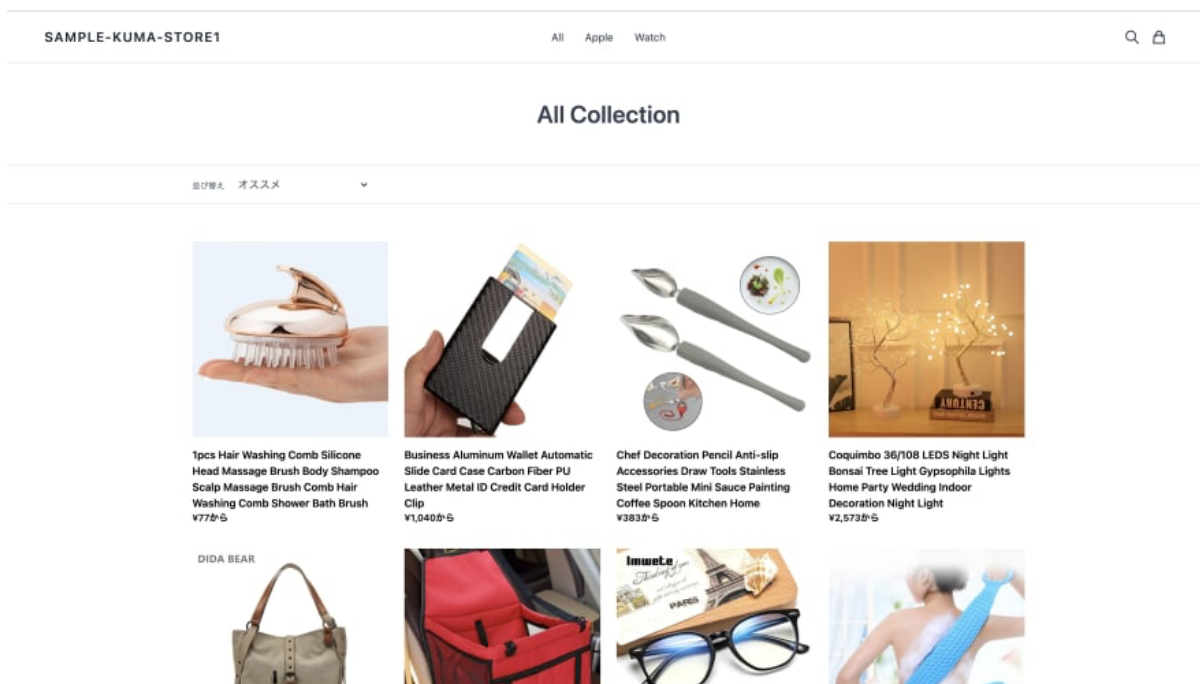


Рис. 1.8 – Тут використовується підхід для SEO оптимізації

Використання Nuxt.js у Shopify дозволяє покращити продуктивність, спростити розробку та підтримку магазину, а також забезпечити високу швидкість завантаження сторінок для користувачів, що сприяє покращенню їхнього досвіду покупок.

На даний момент немає прямого аналогу Server-Side Rendering для Flutter в контексті мобільної розробки. Оскільки Flutter спрямований на створення нативних додатків, концепція серверного рендерінгу для нього може бути не такою релевантною, оскільки вона стосується веб-додатків із відображенням на сервері.

Однак Flutter має свої підходи до оптимізації продуктивності та користувацького досвіду. Наприклад, Flutter дозволяє передварительно завантажувати деякі частини інтерфейсу, використовувати кешування та локальне сховище для збереження даних між сесіями. Також Flutter постійно

розвивається, і можливо, у майбутньому можуть з'явитися нові інструменти для оптимізації роботи зі швидкістю та продуктивністю.

Якщо у вас є конкретні потреби щодо швидкості завантаження та оптимізації Flutter-додатків, варто звернутися до документації Flutter та спільноти розробників для отримання рекомендацій та підходів, що відповідають вашим потребам.

1.4 Мобільна розробка за допомогою Flutter Framework

Flutter - це нова і перспективна платформа, яка вже привернула увагу великих компаній і дозволяє розробляти мобільні додатки з вражаючою продуктивністю та швидкістю роботи.

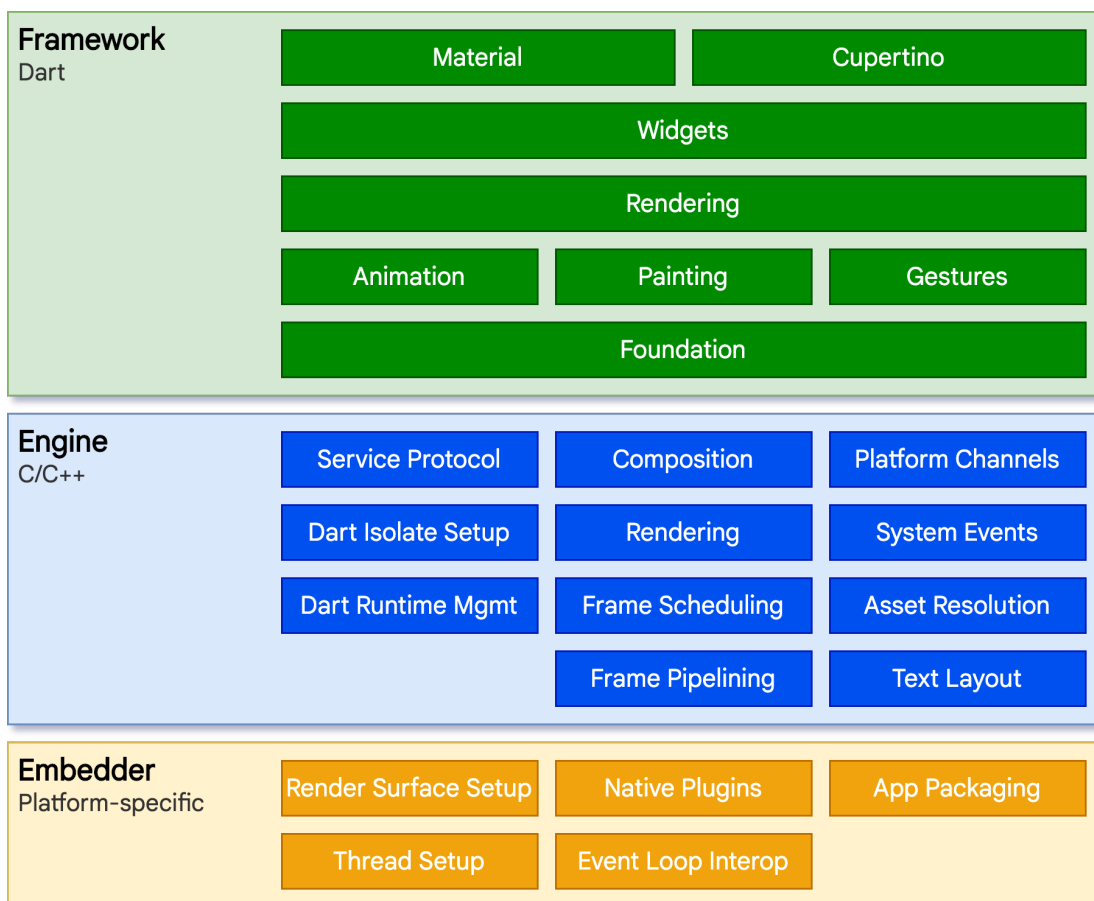


Рис 1.9 – Взаємодія Flutter фреймвока з платформним кодом

Однією з переваг Flutter є його простота у порівнянні з іншими фреймворками для розробки веб-застосунків. Архітектура Flutter відрізняється

від інших програмних каркасів (React, Apache Cordova) тим, що він не використовує для побудови інтерфейсу мови HTML, CSS та Javascript, відповідно і вбудований рушій WebView. Використовується власний рушій для рендерингу.

У відмінну від деяких інших мобільних платформ, Flutter не використовує JavaScript. Замість цього, він використовує мову програмування Dart, яка компілюється в бінарний код, що дозволяє досягти високої швидкості виконання операцій, порівнянної з іншими мовами, такими як Objective-C, Swift, Java або Kotlin.

Один із ключових аспектів Flutter - це відсутність використання рідних компонентів мобільних платформ. Замість цього, весь інтерфейс малюється самостійно в графічному рушії Flutter, подібно до підходу, що використовується в ігровій розробці.

Dart - це сучасна мова програмування, яка була створена компанією Google та вперше представлена у 2011 році. Вона призначена для розробки веб-додатків, мобільних додатків та інших програмних продуктів. Dart є мовою високого рівня, що означає, що вона надає зручний та зрозумілий синтаксис для розробників.

Розвиток Dart як окремої мови, альтернативної JavaScript і безпосередньо підтримуваної у браузерях, визнано недоцільним. Замість цього Dart рухатиметься у бік якіснішої інтеграції з JavaScript і генерації оптимального JavaScript-коду. При цьому розробку віртуальної машини Dart VM буде продовжено, але вона позиціонуватиметься в основному для створення серверних і мобільних застосунків.

Однією з основних особливостей Dart є компіляція в бінарний код. Це означає, що програми, написані на Dart, перетворюються в оптимізований машинний код, що дозволяє досягти високої продуктивності та швидкості виконання. Такий підхід робить Dart підходящим для розробки мобільних додатків, де швидкість роботи є критичним чинником.

Dart є об'єктно-орієнтованою мовою програмування, що дозволяє створювати складні програми з використанням об'єктів та класів. Вона також

підтримує асинхронне програмування, що робить її ідеальним вибором для розробки додатків зі складними асинхронними операціями.

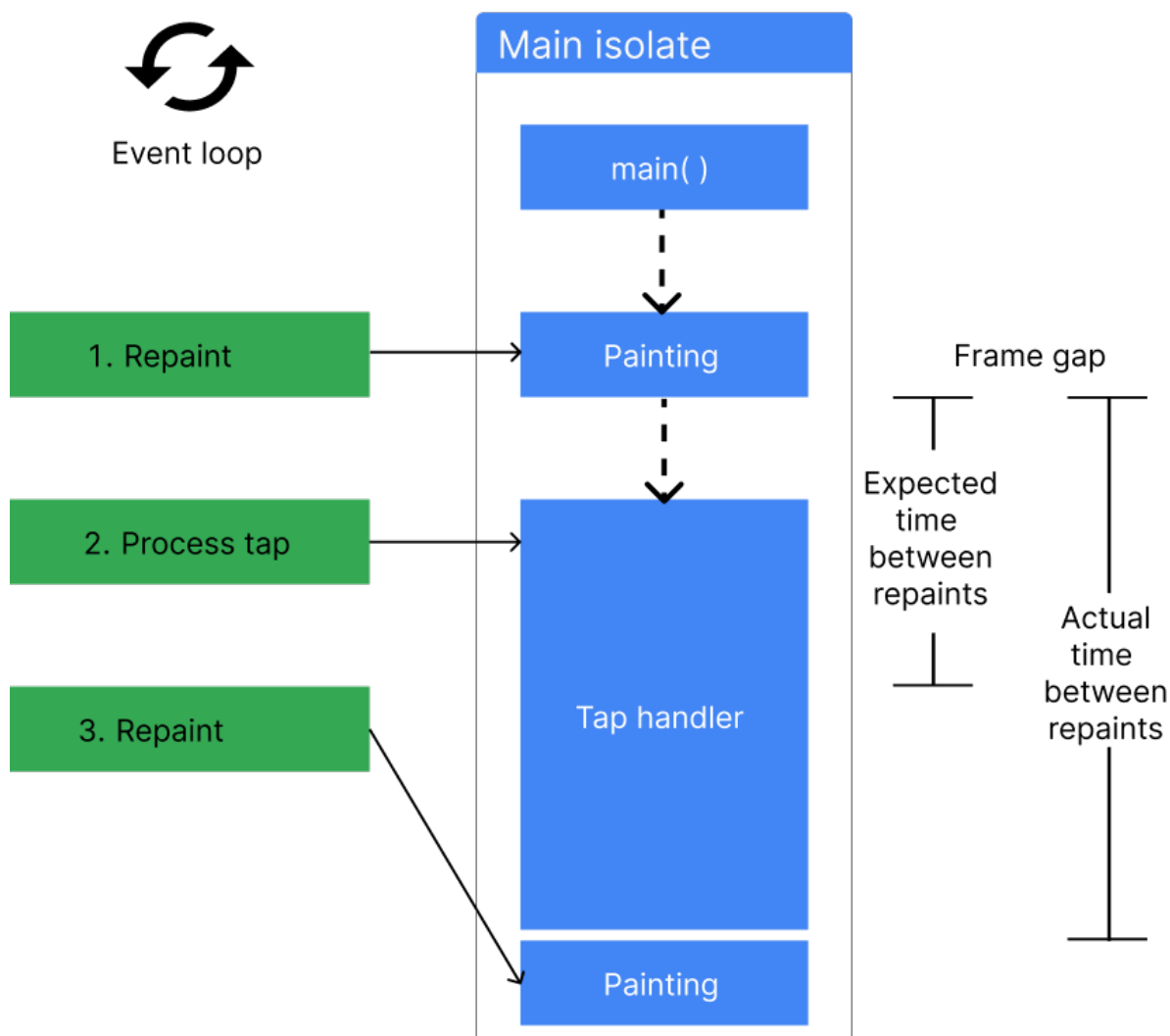


Рис. 1.10 – Приклад роботи Dart ізолята

Однією з ключових переваг Dart є кросплатформеність. Завдяки фреймворку Flutter, який також розроблений Google, ви можете використовувати Dart для створення мобільних додатків для різних платформ, таких як iOS та Android, не переписуючи код заново. Це значно спрощує розробку та підтримку додатків для різних пристроїв.

Dart має велику стандартну бібліотеку, яка містить різні корисні інструменти та пакети для різних завдань. Це дозволяє розробникам швидко та ефективно створювати програми без необхідності писати багато коду з нуля.

У підсумку, Dart - це сучасна, продуктивна та кросплатформена мова програмування, яка є ідеальним вибором для розробки мобільних та веб-додатків. Її зручний синтаксис, висока продуктивність та підтримка асинхронного програмування роблять її привабливою для розробників.

Для побудови інтерфейсу Flutter використовує декларативний підхід, де інтерфейс описується за допомогою віджетів (аналогічно до компонентів у ReactJS). Ця система включає в себе оптимізацію, де віджети перерисовуються тільки при зміні їх стану, що позитивно впливає на продуктивність.

Крім цього, Flutter включає в себе інструмент Hot-reload, який дозволяє розробникам миттєво бачити зміни в коді без перезавантаження додатку.

Один із Android розробників поділився досвідом перепису свого додатку з Java на Dart, і відзначив значний зріст продуктивності та ефективності розробки. Flutter надає документацію та приклади готових застосунків для вивчення технічних особливостей платформи.

У підсумку, Flutter - це молода та обіцяюча платформа, яка зробила суттєвий крок у полегшенні розробки мобільних додатків та забезпечує вражаючу продуктивність та швидкість роботи.

Висновки до першого розділу

У першому розділі кваліфікаційної роботи було розглянуто актуальність та перспективність використання серверно-сайд рендерінгу (SSR) в мобільній розробці, а також представлено основні поняття та особливості фреймворку Flutter.

Зазначені в цьому розділі аналоги застосування SSR та розглянуті можливості Flutter дають підстави для наступних висновків. Використання SSR в мобільних додатках стає актуальним завдяки його здатності прискорювати завантаження та оновлення контенту, що позитивно впливає на досвід користувачів і зменшує навантаження на сервери.

Фреймворк Flutter, у свою чергу, надає розробникам потужний інструментарій для швидкої та ефективної розробки мобільних додатків з динамічним та привабливим інтерфейсом.

Завдяки взаємодії цих технологій, можливий створення мобільних додатків, які не лише працюють ефективно, але й динамічно оновлюють свій контент, що позитивно впливає на залучення та утримання аудиторії. Таким чином, розділ допомагає зрозуміти переваги та перспективи використання SSR разом із Flutter у мобільній розробці та створити основу для подальшого дослідження та розробок.

РОЗДІЛ 2. РОЗРОБКА БІБЛІОТЕКИ ТА ЇЇ ІНТЕГРАЦІЯ В ПРОЕКТ

2.1 Пізнання дерева віджетів в Flutter.

Дерево віджетів - це ключова концепція у Flutter, яка визначає структуру та вигляд вашого мобільного додатка. Воно подібне до структури DOM (Document Object Model) у веб-розробці, але має свої особливості і переваги, оскільки Flutter працює на рівні нативного коду і надає високу продуктивність.

Кожен Flutter додаток має свій кореневий віджет, який є точкою входу в додаток. Цей віджет зазвичай є екземпляром `'MaterialApp'` або `'CupertinoApp'`, які визначають тему додатка, основний віджет і інші параметри. Кореневий віджет визначає загальний стиль та поведінку додатка.

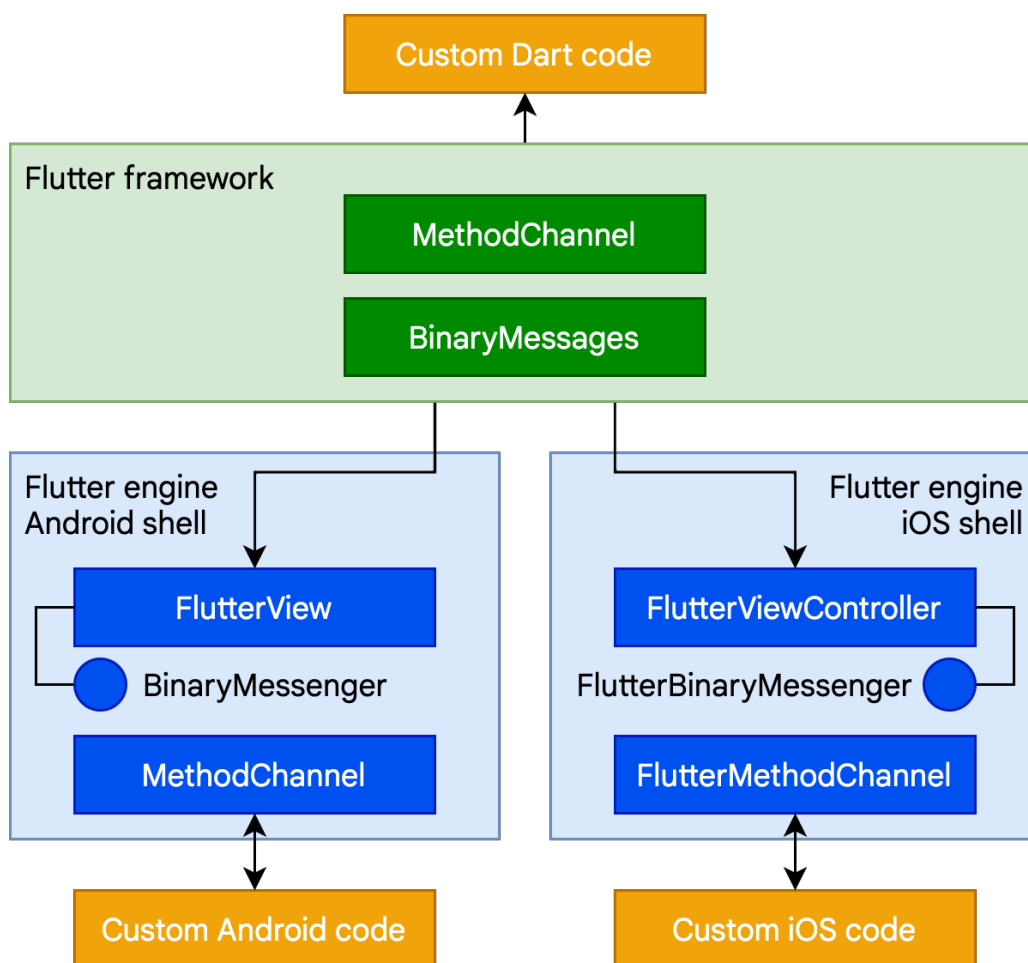


Рис. 2.1 – Архітектура Flutter додатку

Віджети в Flutter утворюють ієрархію, де кожен віджет може мати декілька дочірніх віджетів. Наприклад, ви можете мати віджет `Column`, який містить в собі вкладені віджети `Text`, `Image`, `Button` тощо. Ця вкладеність відображає структуру інтерфейсу та взаємозв'язки між елементами.

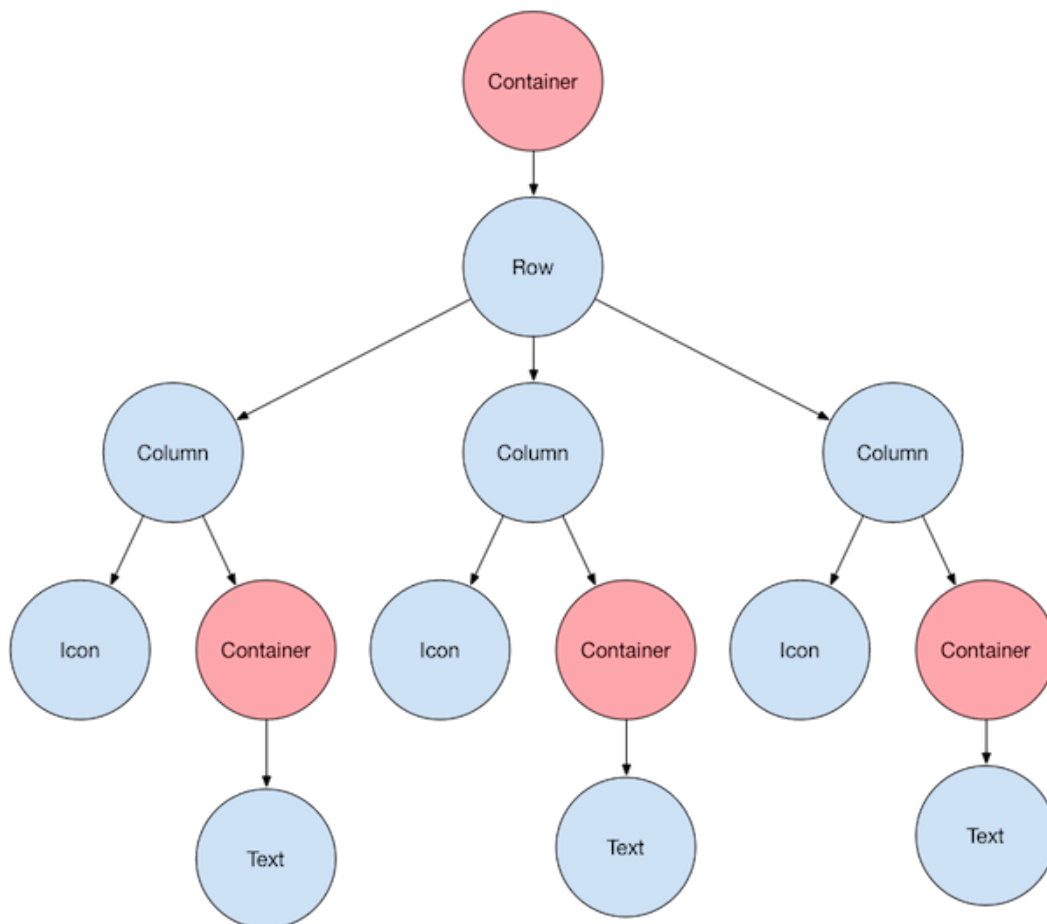


Рис. 2.2 – Приклад дерева віджетів

Кожен внутрішній віджет також може містити свої вкладені віджети, утворюючи глибшу ієрархію. Це дозволяє розробникам побудувати складні інтерфейси з багатьма компонентами. Віджети можуть бути вкладені глибоко, створюючи зручний спосіб організації UI-елементів.

У Flutter є два основних типи віджетів. Віджети зі станом можуть змінювати свій вигляд під час роботи додатка, що дозволяє створювати інтерактивні елементи. Віджети без стану лише відображають статичну інформацію та не мають внутрішнього стану. Вибір між цими типами віджетів залежить від конкретних потреб додатка.

Віджети можуть взаємодіяти один з одним шляхом передачі даних та обробки подій. Це дозволяє створювати складні взаємодіючі інтерфейси та обробляти дії користувачів.

Віджети зі станом можуть оновлювати свій вигляд після зміни даних або подій. Це робить Flutter дуже потужним для створення реактивних додатків, які реагують на дії користувачів в реальному часі.

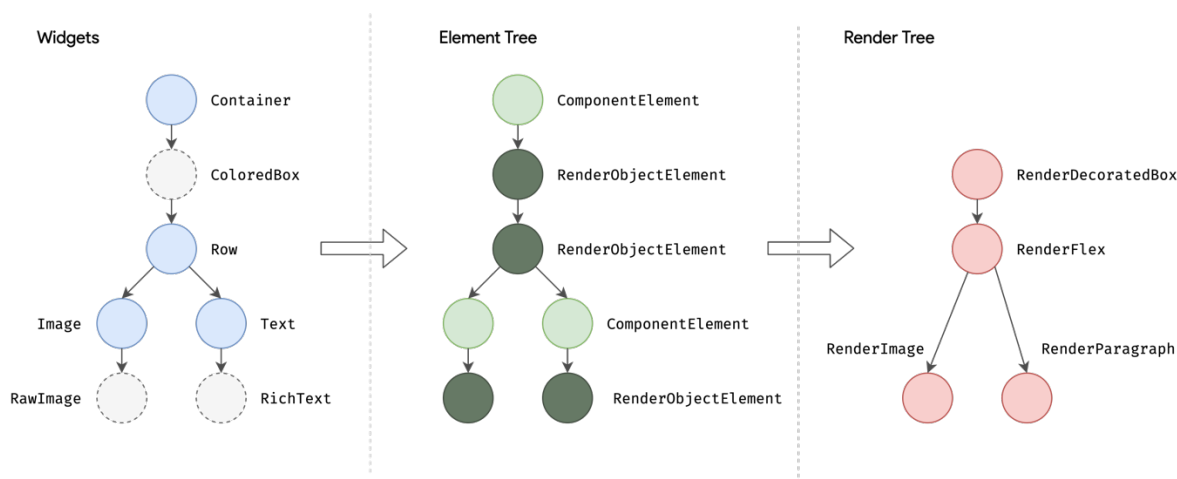


Рис. 2.3 – Детальне зображення етапів компіляції дерева віджетів

Це загальний огляд дерева віджетів в Flutter. Розуміння цієї структури є ключовим для успішного розробництва мобільних додатків у Flutter, де кожен віджет відповідає за свою частину інтерфейсу та може бути налаштований та взаємодіяти з іншими віджетами в дереві.

2.2 Загальні підходи для розробки додатків на базі Flutter

Розробники мобільних додатків на базі Flutter в основному використовують CSR (Client-Side Rendering) підхід. Flutter зазвичай використовується для створення додатків, які виконуються на клієнтському пристрої (мобільний пристрій), і вони надсилають запити до сервера, коли потрібно отримати дані з веб-служб або інших джерел.

SSR (Server-Side Rendering) в основному використовується для веб-застосунків, де інтерфейс створюється на сервері та відправляється до клієнта у вигляді готового HTML-коду. У випадку мобільних додатків, які розробляються

з використанням Flutter, інтерфейс створюється безпосередньо на пристрої користувача за допомогою фреймворку.

Однак можливо виняткові випадки, коли розробники вирішують використовувати SSR для конкретних частин додатків, якщо це необхідно для певних функцій або вимог проекту. Проте, це не типовий сценарій для мобільних додатків на базі Flutter, і переважна більшість розробників використовує CSR для створення крос-платформених мобільних додатків.

В розробці мобільних додатків на Flutter розробники можуть використовувати різні підходи до управління станом. Один із ключових аспектів вибору підходу полягає у рівні складності проекту та потребах у збереженні та оновленні даних.

Локальний стан використовується для простих компонентів, де потрібно зберігати невелику кількість локальних даних. В основному використовує «setState» для зміни стану виджета.

Основні особливості локального стану в Flutter:

- Простота використання: Механізм локального стану в Flutter легкий у розумінні та використанні.
- Обмеженість видимості: Дані, збережені в локальному стані, доступні лише в межах відповідного віджета або компонента.
- Імутабельність: Зазвичай дані оновлюються шляхом створення нового об'єкта, що містить оновлені дані, замість безпосередньої зміни поточних даних.
- Обробка подій: Локальний стан включає обробники подій, які реагують на дії користувача або зміни стану віджета.
- Підтримка анімацій: Локальний стан ідеально підходить для створення анімацій та інтерактивної взаємодії в межах окремого віджета.
- Підтримка тем: Локальний стан дозволяє легко змінювати відображення даних в залежності від активної теми додатка.

Локальний стан допомагає розробникам створювати інтерактивні та анімовані інтерфейси, зберігаючи дані на рівні віджета. Це спрощує управління станом та полегшує розробку додатків.

Глобальний стан дозволяє зберігати стан додатку централізовано, такий як стан авторизації, налаштування тощо. Це може включати використання «InheritedWidget», «Provider» або інших рішень.

Глобальний стан (Global State) в Flutter - це механізм зберігання та управління даними, які доступні для всіх віджетів і компонентів у додатку. Він дозволяє зберігати дані на рівні додатка та робити їх доступними для всіх віджетів без необхідності передачі даних вручну між віджетами. Глобальний стан є корисним у великих додатках або в ситуаціях, коли декілька віджетів потребують доступу до одних і тих самих даних.

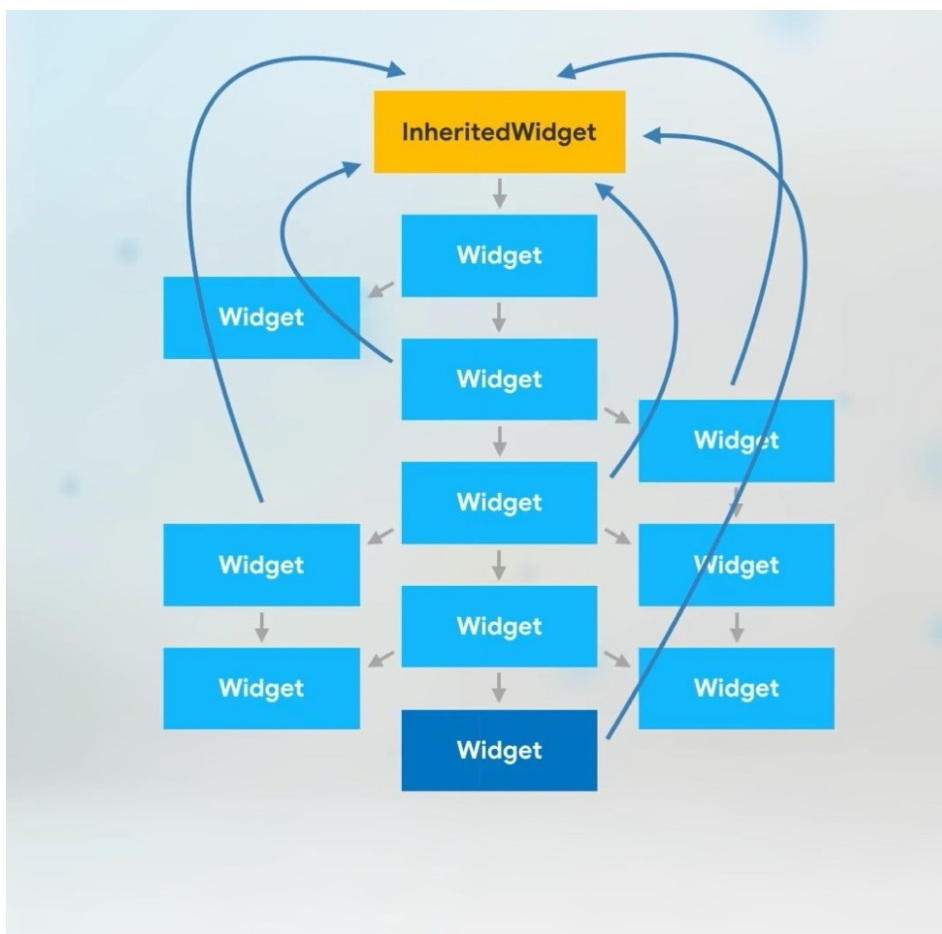


Рис. 2.4 – Використання InheritedWidget для доступу до інформації
рівнем вище

Основні особливості глобального стану в Flutter:

- Глобальний стан зберігається на рівні додатка і є доступним для всіх віджетів безпосередньо.
- Глобальний стан дозволяє централізовано зберігати та оновлювати дані, що спрощує управління станом додатка.
- Хоча дані глобального стану доступні для всіх віджетів, вони залишаються ізольованими від інших компонентів, що допомагає уникнути конфліктів та недорозумінь.
- Глобальний стан особливо корисний у великих додатках зі складним інтерфейсом та великою кількістю даних, які потрібно спільно використовувати.
- Глобальний стан може бути з'єднаним з зовнішніми джерелами даних, такими як бази даних або веб-сервери, для отримання та оновлення інформації.
- Глобальний стан спрощує розробку та підтримку додатків, оскільки він дозволяє використовувати однакові дані та логіку в різних частинах додатка.

Глобальний стан в Flutter може бути реалізований за допомогою різних підходів, таких як інкапсуляція в батьківському віджеті, використання сторінок (Providers), або використання сторонніх пакетів, таких як Redux або MobX. Вибір підходу залежить від конкретних потреб додатка та уподобань розробника.

Реактивні підходи, такі як MobX та GetX, використовують реактивне програмування та спостерігачі для автоматичного оновлення інтерфейсу користувача при зміні стану. GetX став популярним в спільноті Flutter завдяки своїй продуктивності, простоті використання та багатофункціональності, що робить його прекрасним вибором для розробників, які шукають швидкий та ефективний спосіб створення додатків на Flutter. GetX і MVVM (Model-View-ViewModel) - це два підходи, які часто використовуються в розробці мобільних

додатків на платформі Flutter, і їх можна успішно поєднувати для створення добре структурованих та легко керованих додатків.

MVVM - це архітектурний шаблон, який розділяє додаток на три основні частини: Model (дані), View (представлення) та ViewModel (логіка відображення). ViewModel відокремлює логіку відображення від інтерфейсу користувача, забезпечуючи більшу гнучкість та тестовість.

Комбінування GetX та MVVM може бути дуже корисним в розробці Flutter-додатків:

Програмісти використовують GetX для управління станом додатка, зберігання та оновлення даних на основі подій та реактивних об'єктів. Це допоможе вам створити реактивний інтерфейс.

ViewModel в MVVM може бути використаний для відокремлення логіки відображення від самого інтерфейсу. Ви можете використовувати GetX, щоб пов'язати ViewModel зі своїм представленням (View), а реактивність GetX сприяє оновленню інтерфейсу при зміні даних у ViewModel.

Ця комбінація дозволяє створити додаток з більшою контрольованістю, покращеною тестовістю та реактивним інтерфейсом. Користуючись GetX разом з MVVM, ви можете забезпечити чистоту та розширюваність вашого додатку на платформі Flutter.

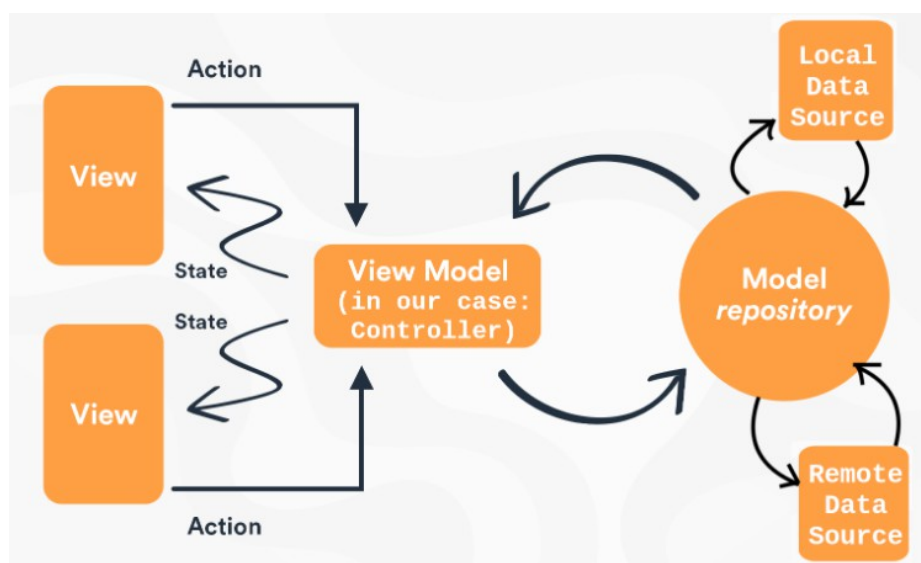


Рис. 2.5 – Один з прикладів використання GetX

В реактивному програмуванні ключовими концепціями є спостережувані дані (observable data), події (events), відповідники (observers), та потоки подій (event streams). Реактивний підхід дозволяє створювати компоненти програми, які взаємодіють один з одним через відправку та прийом подій, реагуючи на зміни в даних безпосередньо.

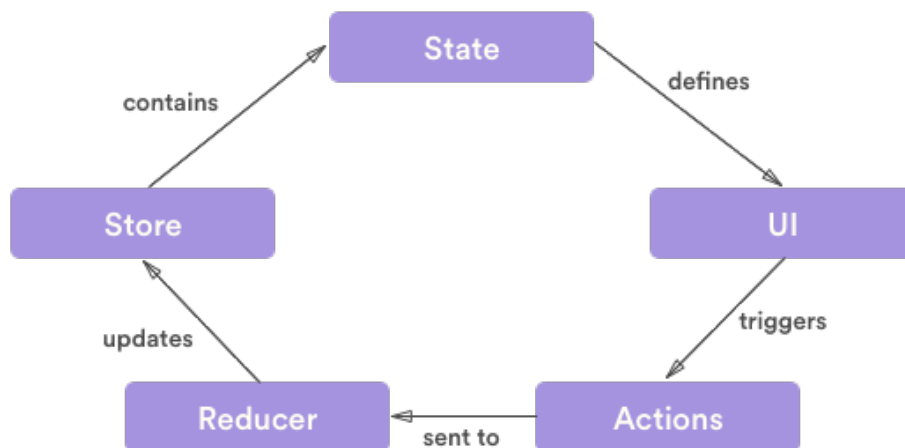


Рис. 2.6 – Схема роботи MobX

Архітектурні патерни, такі як «BLoC», відокремлюють бізнес-логіку від інтерфейсу користувача та використовують потоки для управління станом.

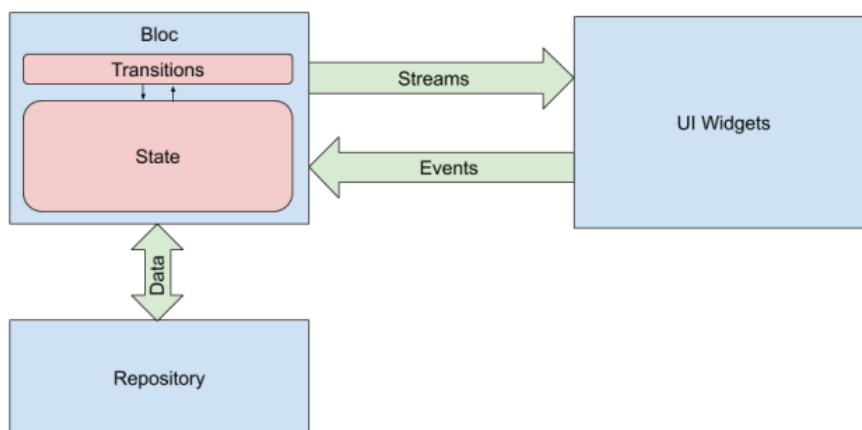


Рис. 2.7 – Схема роботи паттерну BLoC

Абревіатура BLoC розшифровується як business logic component (пер. компонент бізнес-логіки). Це єдине джерело правди для окремої фічі вашого застосунку BLoC реагує на події, відповідаючи на них змінами стану за допомогою `async*` функцій. Паттерн BLoC наслідується від класу Stream, який аналогічно приймає певні входні дані та відповідає на них змінами. Потік

повідомляє про події частині коду яка відповідає за відображення. В випадку чату цей паттерн використовується для завантаження списку користувачів і повідомлень та відправки повідомлень.

Вибір підходу до стейт-менеджменту визначається великою мірою складністю проекту, потребами у збереженні даних, а також особливостями розробки. Багато розробників комбінують декілька з цих методів, щоб досягти оптимального рівня керованості станом та продуктивності.

2.3 Практичні знання по реалізації подібної технології

Першим кроком було ретельне планування впровадження SSR у реєстраційні процеси. Цей етап включав аналіз вимог до реєстрації для різних проектів, ідентифікацію спільних елементів та визначення того, як SSR може сприяти створенню гнучкої системи.

Після зібрання вимог та їх документування ми переходимо до етапу аналізу. Аналіз вимог включає докладний розгляд та розуміння всіх вимог, що були зібрані на попередніх етапах. Головною метою аналізу є розробка стратегії для втілення цих вимог у конкретну бібліотеку для Server-Side Rendering з використанням Flutter. Нижче наведено ключові кроки аналізу вимог:

- Ідентифікація головних функцій: Основний аналіз починається з ідентифікації ключових функціональних можливостей, які мають бути реалізовані бібліотекою. Це включає розуміння того, яким чином бібліотека повинна взаємодіяти з сервером для рендерингу веб-сторінок.
- Декомпозиція вимог: Вимоги можуть бути складними і містити різноманітні сценарії використання. На цьому етапі проводиться декомпозиція вимог на менші компоненти, які можуть бути реалізовані окремо. Це допомагає зрозуміти, які частини бібліотеки потрібно розробити.

- Аналіз додаткових можливостей: Поза основними функціональними вимогами, можуть існувати додаткові можливості, які можуть зробити бібліотеку більш потужною. Наприклад, підтримка кешування для покращення продуктивності. На цьому етапі аналізуються такі можливості та вирішується, чи вони підходять до поточного контексту проекту
- Визначення додаткових вимог: Під час аналізу можуть виявитися додаткові вимоги, які були пропущені на попередніх етапах. Важливо додати ці вимоги до загального списку та розглянути їх в контексті.
- Аналіз можливих ризиків: Вимоги можуть включати можливі ризики, такі як незрозумілі аспекти або суперечливі вимоги. Аналіз ризиків допомагає виявити потенційні проблеми та забезпечити їх вирішення на ранніх етапах.
- Узгодження замовником: Важливо провести замовником узгодження аналізованих вимог. Це може включати обговорення деталей, роз'яснення або внесення корекцій до вимог.

Аналіз вимог є критичним етапом перед початком реалізації бібліотеки. Він допомагає зрозуміти обсяг та складність проекту, а також визначити правильний напрям для розробки, забезпечуючи відповідність вимогам та очікуванням замовника.

2.3 Знання та висновки з інтеграції подібної бібліотеки

Під час практики було реалізовано динамічну форму для створення бланків з можливістю заповнення даних та відправки їх на сервер. Ця бібліотека повинна була забезпечити можливість генерації реєстраційних форм на сервері, враховуючи вимоги кожного проекту. Під час розробки було створено модульну структуру бібліотеки, що дозволяла динамічно змінювати логіку та елементи форми. Це забезпечило високий рівень гнучкості та персоналізації.

Після розробки бібліотеки настала фаза її інтеграції з існуючими реєстраційними процесами. Цей етап включав тестування різних сценаріїв реєстрації для різних проектів з використанням SSR бібліотеки.

Важливо враховувати різні можливі сценарії взаємодії з формою. Наприклад, тестуйте форму з неправильними введеними даними, форму з різними варіантами вибору параметрів тощо.

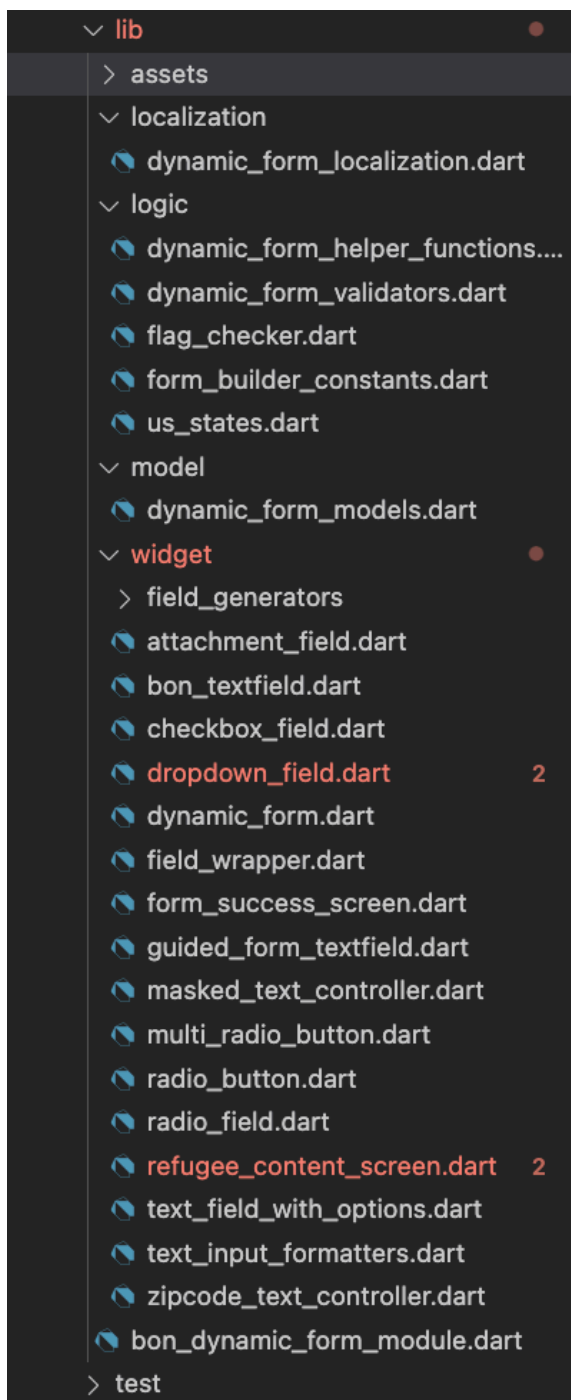


Рис. 2.8 – Дерево файлів створеної бібліотеки

Код для реалізації динамічних форм в Flutter може бути досить різноманітним в залежності від специфіки проекту та потреб користувача. Однак є загальний огляд того, як можна створювати динамічні форми за допомогою Flutter.

Якщо форма включає в себе поле введення тексту то потрібно перевірити як вона взаємодіє з клавіатурою на мобільних пристроях та переконатись що користувач може легко вводити дані та взаємодіяти з формою за допомогою жестів.



Рис. 2.9 – Запит на форму

Тестування допомогло ідентифікувати можливі проблеми та недоліки, які було виправлено в процесі вдосконалення бібліотеки. Це було важливим кроком для забезпечення якості та стабільності реєстраційного процесу.

Перший крок - це аналіз вимог до бібліотеки. Розробники повинні чітко зрозуміти, які функції та можливості має бібліотека, та як вона повинна взаємодіяти з іншими програмними компонентами. Розробники визначають, які аспекти бібліотеки потрібно протестувати. Це може включати в себе тестування

основних функцій, граничних випадків, а також відповідність стандартам та специфікаціям.

Для тестування бібліотеки потрібно створити тестове середовище, яке відтворює умови, в яких бібліотека буде використовуватися в реальних проектах. Це може включати в себе налаштування зовнішніх залежностей та створення фіктивних даних. Розробники створюють тестовий код, який виконує автоматичні перевірки функціональності бібліотеки. Тести можуть бути написані вручну або автоматизовані за допомогою тестових фреймворків.

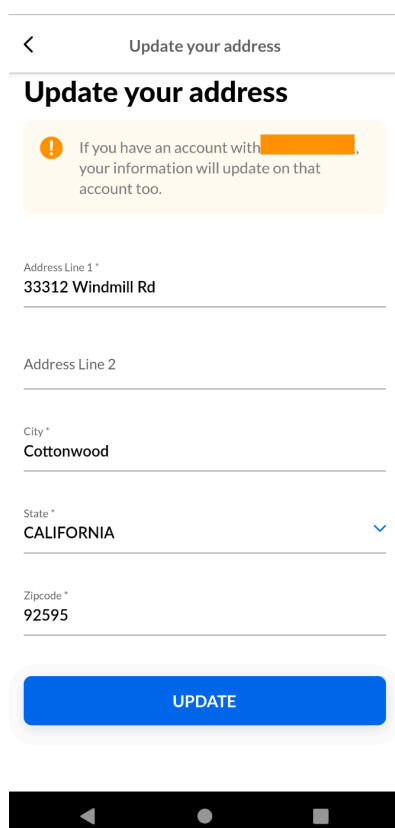


Рис. 2.10 – Результат роботи

Окрема процедура - це валідація тестів, щоб переконатися, що вони коректно відображають функціональність бібліотеки.

Після успішного проходження всіх тестів бібліотеку можна готувати до релізу та поширення серед користувачів.

Після інтеграції і тестування бібліотеки, було виявлено деякі можливості для оптимізації. Були вдосконалені алгоритми генерації форми на сервері та проведено оптимізацію завантаження.

Результати впровадження форми були вражаючими. Час завантаження реєстраційної сторінки скоротився, а якість обслуговування клієнтів підвищилася завдяки гнучкості та персоналізації. Але з тим додалась проблема з додаванням нових компонентів в кодбазу та її загальний розвиток який спричинений складністю написаного коду та фіксованістю на конкретній задачі, а саме рішенням проблеми з гнучкою реєстрацією.

З знань здобутих на практиці було видвинуто нові умови для створення бібліотеки.

2.4 Результат опрацьованого матеріалу

У цьому підрозділі ми розглянемо результати нашої роботи, які стали результатом дослідження та розробки Server-Side Render бібліотеки для фреймворку Flutter. Даний розділ включає в себе опис архітектури бібліотеки, її реалізацію, результати тестування, порівняння з існуючими рішеннями та приклади її використання в реальних проектах.

Впровадження Server-Side Rendering у мобільних додатках, зокрема у фреймворку Flutter, є актуальним завданням, яке вимагає комплексного дослідження та розробки ефективних рішень. У цьому підрозділі було представлено важливі результати роботи, які допоможуть краще зрозуміти важливість та вплив нашої Server-Side Render бібліотеки на сучасний розвиток мобільної розробки.

Цей підрозділ є ключовим для розуміння досягнень та значущості нашого дослідження та розробки та визначення можливих напрямків подальших досліджень у цій області.

Переваги динамічних форм у Flutter:

- Кросплатформенність: Flutter є кросплатформеним фреймворком, що дозволяє створювати динамічні форми, які працюють на різних платформах, таких як Android і iOS, з одним і тим же кодом.

- Широкий вибір віджетів: Flutter має багатий набір вбудованих віджетів і бібліотек, що полегшують створення різних типів динамічних форм та елементів управління.
- Гнучкість дизайну: Flutter дозволяє створювати кастомні дизайни динамічних форм, що відповідають стилю та потребам вашого проекту.
- Гарна продуктивність: Фреймворк Flutter славиться високою продуктивністю та швидкістю рендерингу, що важливо для динамічних форм, які вимагають активної взаємодії з користувачем. Flutter використовує графічний рушій Skia для прискорення процесу відображення. Він може досягати високої продуктивності та рівня FPS (кадрів на секунду), що робить додатки Flutter швидкими та плавними.

Недоліки динамічних форм у Flutter:

- Залежність від мови програмування Dart: Flutter використовує мову програмування Dart, яка може бути менш популярною та менше підтримуватися спільнотою порівняно з іншими мовами програмування.
- Важкість для новачків: Для новачків у розробці Flutter динамічних форм може бути досить складною, оскільки вони повинні знати Dart та специфіку фреймворка.
- Обмежена підтримка веб-платформи: У порівнянні з мобільними платформами, підтримка веб-платформи Flutter ще досить експериментальна і може мати обмеження для створення динамічних форм.
- Збільшене споживання ресурсів: Динамічні форми можуть вимагати більше ресурсів, таких як CPU та пам'ять, на пристрої користувача.
- Залежність від розробників: Розробку та підтримку динамічних форм у Flutter часто виконують розробники, що може бути важким, якщо немає достатньої кількості ресурсів або досвіду.

Загалом, при використанні Flutter для розробки динамічних форм важливо бути готовим до вирішення тих чи інших викликів, які можуть виникнути в процесі розробки.

Впровадження динамічних форм в Flutter вимагає обов'язкового розгортання складної інфраструктури та має свої негативні аспекти.

По-перше, це може призвести до значного збільшення обсягу коду і складності проекту. Динамічні форми часто потребують великої кількості елементів і компонентів, які повинні бути створені та керовані. Це може ускладнити процес розробки і підтримки додатку, особливо в разі потреби в змінах або розширеннях.

По-друге, динамічні форми можуть бути схильні до виникнення помилок і несправностей. Завдання, пов'язані з валідацією введених даних, правильністю взаємодії між компонентами та обробкою подій, можуть стати причиною появи багів. Це вимагає ретельного тестування та відладки.

По-третє, динамічні форми можуть призвести до збільшення витрат пам'яті та обчислювальних ресурсів. Створення та управління багатьма елементами і даними форми може призвести до підвищеного споживання ресурсів пристрою, особливо на старших або менш потужних пристроях.

Нарешті, динамічні форми можуть ускладнити роботу з іншими бібліотеками або сторонніми рішеннями, які використовуються у проекті. Не всі бібліотеки або плагіни можуть підтримувати динамічні форми, і це може вимагати додаткової роботи для їх інтеграції.

Всі ці аспекти важливо враховувати при розгляді можливості впровадження динамічних форм в Flutter та прийняття рішення про їхнє використання в конкретному проекті, особливо якщо ви обрали Server-Side Rendering (SSR) підхід. При цьому, враховуючи різні можливі сценарії взаємодії та валідації даних в динамічних формах, а також забезпечуючи безпеку та надійність, це буде вірний вибір розв'язки задачі для проекту.

Дерево віджетів в Flutter є важливою складовою для створення інтерфейсу користувача. Ця структура дозволяє розробникам не лише створювати і керувати

віджетами для побудови інтерфейсу, але також інтегрувати серверно-орієнтованого рендерінг (SSR) для отримання додаткових даних та вмісту з сервера.

Простота використання дерева віджетів робить процес розробки інтерфейсу користувача більш зручним та ефективним. Розробники можуть легко створювати та налаштовувати віджети для досягнення потрібного результату, а також запитувати та інтегрувати дані з сервера, використовуючи SSR. Тому було видвинуте рішення – розробити бібліотеку для парсингу дерева даних з серверу в компоненти користувацького інтерфейсу.

Легкий доступ до окремих елементів та їх властивостей робить роботу з віджетами дуже зручною, а можливість інтеграції SSR дозволяє отримувати оновлені дані з сервера та оновлювати віджети відповідно до цих даних без зайвого завантаження.

Можливість створювати власні віджети та інтегрувати SSR надає розробникам гнучкість у створенні складних інтерфейсів та отриманні актуальних даних з сервера.

Компонування та вкладення віджетів надає гнучкість у створенні складних інтерфейсів, а реактивний підхід допомагає створювати інтерактивні додатки, які реагують на дії користувачів та оновлення з сервера.

Загалом, дерево віджетів у Flutter разом із можливістю інтеграції SSR є потужним інструментом для розробки інтерфейсу користувача та забезпечення отримання актуальних даних з сервера, що робить додатки більш продуктивними та зручними для користувачів.

Висновки до другого розділу

У даному контексті можна виділити, що дерево віджетів у Flutter, поєднане із можливістю інтеграції серверо-орієнтованого рендерінгу (SSR), створює потужну інструментальну основу для розробки мобільних додатків з інтерфейсом користувача, який не лише відповідає високим стандартам, але й

забезпечує швидкий та ефективний обмін даними з сервером. Розглянуті переваги, такі як простота використання, легкий доступ до елементів, система оновлень та можливість створення власних віджетів, роблять Flutter з SSR ідеальним вибором для розробки додатків, які вимагають високої продуктивності та динамічної зміни вмісту.

Додавши до цього можливість інтеграції SSR, розробники отримують можливість отримувати оновлені дані з сервера та оновлювати віджети додатка в реальному часі, зберігаючи при цьому високу швидкість роботи. Відкриваються нові горизонти для створення інтерактивних та реактивних додатків, що сприяє покращенню досвіду користувачів та забезпеченню більш ефективної обробки даних.

Зазначимо також, що розглянуті можливості Flutter з SSR мають значний потенціал для подальшого розвитку та використання у різних галузях розробки мобільних додатків, що робить цю технологію актуальною та важливою для кваліфікаційної роботи.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З БІБЛІОТЕКОЮ

3.1 Структура бібліотеки та шляхи використання.

Unified Modeling Language (UML) є стандартним мовою моделювання, який використовується для специфікації, візуалізації, побудови та документації об'єктів програмного забезпечення. UML надає набір графічних позначень, які можуть відображати структуру і поведінку програмних систем. UML є важливим у процесі розробки програмного забезпечення, оскільки він дозволяє розробникам і стейкхолдерам легше спілкуватися, обговорювати вимоги та проектувати систему до її фактичного створення.

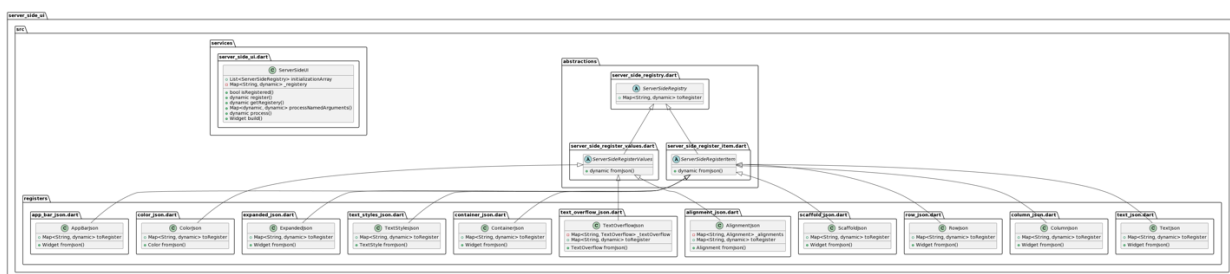


Рис. 3.1 – Діаграма класів

Зображення, є UML діаграмою класів, яка описує бібліотеку для серверного рендерингу (SSR) для Flutter. Ця бібліотека має на меті обробляти JSON запити і генерувати відповідні Flutter компоненти, що дозволяє створювати динамічний користувацький інтерфейс на стороні сервера, який потім може бути доставлений до клієнта.

На діаграмі представлені основні компоненти бібліотеки:

- “src/server_side_ui.dart” - головний клас, який відповідає за ініціалізацію середовища і керування життєвим циклом рендерингу компонентів.
- “services” містить логіку, що використовується сервером для обробки запитів і відповідей. Він включає клас «ServerSideUi», який здається відповідає за ініціалізацію сервісів і можливо зберігає стан компонентів або сесії.

- “registers” - цей пакет містить класи, які використовуються для реєстрації та управління різними типами віджетів Flutter, такими як AppBar, Color, Expanded та інші, на основі даних JSON.
- “abstractions” відділ абстракцій який містить базові класи або інтерфейси, які визначають загальні властивості та методи для різних типів реєстраційних елементів і класів.
- “server_side_registry.dart” і “server_side_register_item.dart” - ці компоненти, ймовірно, відповідають за ведення реєстру всіх доступних віджетів, які можуть бути створені сервером, і включають механізми для додавання нових типів віджетів до системи.

Загалом, ця архітектура дозволяє розробникам використовувати бібліотеку для читання JSON запитів і динамічно створювати віджети Flutter без необхідності писати повторний код на стороні клієнта. Це може бути особливо корисним для створення додатків з широкою варіативністю користувацьких інтерфейсів або коли необхідно швидко оновлювати інтерфейси без розгортання нової версії додатку.

3.2 Документація бібліотеки.

Документація Flutter є всебічним ресурсом, який надає шляхи для початківців та досвідчених розробників для розуміння та освоєння цього фреймворку. Для новачків документація керує процесом налаштування середовища та створення першого додатку на Flutter, пропонуючи подальші кроки для розробників, що приходять з інших платформ, таких як Android, SwiftUI, UIKit, React Native та Xamarin.Forms. Вона включає детальні пояснення як створювати макети, розуміти модель розкладки Flutter та додавати інтерактивність до додатків, інтегруючи станові віджети.

Забезпечення доступності та зрозумілості коду для інших розробників - це важлива складова успішної роботи над Flutter проектами. Важливо створити

докладну документацію, яка допоможе іншим розробникам швидше зрозуміти ваш код і використовувати його в своїх проектах.

Коментарі в коді дозволяють пояснити різні аспекти вашого програмного коду, такі як функції, класи, методи і змінні. Вони повинні містити інформацію про параметри, типи даних, які повертає функція, та опис її призначення. Це робить код більш зрозумілим і сприяє спільній роботі над проектом.

Використання анотацій, таких як «`@deprecated`», допомагає позначити застарілі частини коду, що дозволяє іншим розробникам уникати їх використання.

```

41  /// Checks if a given path is registered in the `_registry`.
42  ///
43  /// The path is split into levels, and each level is checked for ex
44  /// If any level does not exist, it returns `false`, indicating the
45  ///
46  /// @param path The path to check for registration.
47  /// @return `true` if the path is registered, `false` otherwise.
48  bool isRegistered(String path) {

```

Рис. 3.2 – Документація в коді конкретної функції

Анотації «`@param`» і «`@return`» надають інформацію про типи параметрів і повернених значень функцій, що полегшує їх коректне використання.

server_side_ui package

server_side_ui package

LIBRARIES

server_side_ui

Flutter SSR Library

Introduction

Briefly introduce what the Flutter SSR library is and its primary use case. Explain how it integrates with Flutter applications and its benefits for server-side rendering.

Features

List the main features of the library. Include rendering capabilities and optimization features.

Installation

Provide step-by-step instructions on how to install the library in a Flutter project. Include any prerequisites or dependencies that need to be installed first.

Add this to your package's pubspec.yaml file:

dependencies:

server_side_ui: ^latest_version

Usage

From `/example` folder.

MaterialPageRoute(

builder: (ctx) => ServerSideUI()

.build(jsonDecode(_controller.text), context: ctx),

),

Рис. 3.2 – Згенерована перша сторінка документації

Генерація документації за допомогою спеціальних інструментів, таких як “dartdoc”, дозволяє автоматично створювати докладну документацію з коментарів у коді. Ця документація може бути представлена у вигляді HTML-сторінок, які дуже зручно переглядати та шукати необхідну інформацію.

Перша сторінка публікації показує дані з файлу “README.md” які розписані Markdown розміткою. Markdown - це легкий мовний формат для розмітки тексту, який дозволяє створювати структурований контент з простотою та без великої кількості HTML-коду. Якщо натиснути на назву бібліотеки – ми отримаємо детальну документацію компонентів, класів та методів які були задокументовані в коді.

The screenshot shows the DartDoc documentation for the `ServerSideUI` class. At the top, a breadcrumb trail reads "server_side_ui > server_side_ui > ServerSideUI class". Below this, the library name "server_side_ui library" is shown. A list of classes includes `ServerSideRegisterItem`, `ServerSideRegisterValues`, and `ServerSideUI`. The main heading is "ServerSideUI class" with a "Null safety" badge. The description states: "ServerSideUI is a class that manages the server-side user interface components. It contains an initialization array that holds instances of various JSON classes. These classes represent different UI components like Alignment, AppBar, Color, Column, etc. It also maintains a registry map (_registry) that can be used to store and retrieve dynamic data related to these UI components." A "Usage:" section shows the code: `var serverSideUI = ServerSideUI();`. Below are sections for "Constructors" (listing `ServerSideUI()`) and "Properties" (listing `hashCode` and `initializationArray`).

Рис. 3.3 – Опис головного класу бібліотеки

Публікація створеної документації в мережі робить її доступною для інших розробників, що сприяє спільній роботі над проектами та полегшує розуміння коду. Це дозволяє створювати більш якісні та довгострокові проекти з використанням Flutter.

3.3 Приклад використання бібліотеки

У цьому розділі ми розглянемо приклад використання нашої бібліотеки “server_side_ui” у вашому проєкті “example”. Код, наведений нижче, демонструє, як ідея серверного рендерингу може бути впроваджена в мобільний додаток на Flutter. Давайте розглянемо деталі цього прикладу:

У початку нашого додатку ми імпортуємо необхідну бібліотеку “server_side_ui/server_side_ui.dart”. Ця бібліотека надає нам зручний інтерфейс для використання серверного рендерингу у Flutter-додатках.

```
import 'dart:convert';

import 'package:flutter/material.dart';

import 'package:server_side_ui/server_side_ui.dart';

Run | Debug | Profile
void main() => runApp(
  MaterialApp(
    home: AppExample(),
  ), // MaterialApp
);
```

Рис. 3.4 – Ініціалізація додатку та початкового екрану

Ми створюємо основний клас додатку AppExample, який є кореневим віджетом нашого додатку. У цьому віджеті ми будуємо основний інтерфейс додатку, який включає в себе AppBar з назвою додатку.

```
bottomSheet: Padding(
  padding: const EdgeInsets.all(8.0),
  child: ButtonBar(
    alignment: MainAxisAlignment.center,
    children: [
      ElevatedButton(
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (ctx) => ServerSideUI().build(
                jsonDecode(_controller.text),
                context: ctx,
              ),
            ), // MaterialPageRoute
          );
        },
        child: const Text('Open'),
      ), // ElevatedButton
```

Рис. 3.5 – Кнопка яка виконує функціонал бібліотеки

Для введення JSON-даних ми використовуємо `TextFormField`, яке дозволяє користувачеві вводити дані. Поле має можливість вводити багато рядків JSON, оскільки воно підтримує багаторядковий ввід.

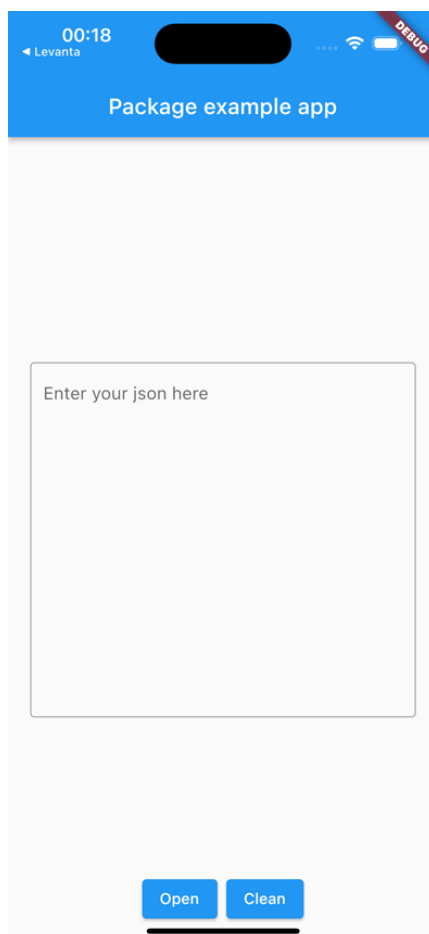


Рис. 3.6 – Результат компіляції проекту - прикладу

Кнопка `Open` відповідає за відправлення JSON-даних для генерації сторінки з використанням серверного рендерингу. Функція цієї кнопки це запит на навігацію з сторінкою яку генерує функція `build` бібліотеки. Клас `MaterialPageRoute` відповідає за анімований перехід на сторінку. Він є стандартним для бібліотеки компонентів `Material`. Функція `jsonDecode` відповідає за перетворення введеного тексту в мапу.

У нижній частині екрану розміщено `AppBar` з двома кнопками `Open` та `Clean`.

Кнопка `Clean` дозволяє очистити поле вводу JSON-даних для подальшого використання. Оскільки в прикладі використовується контролер введення то очистка тексту відбувається всього в один метод.

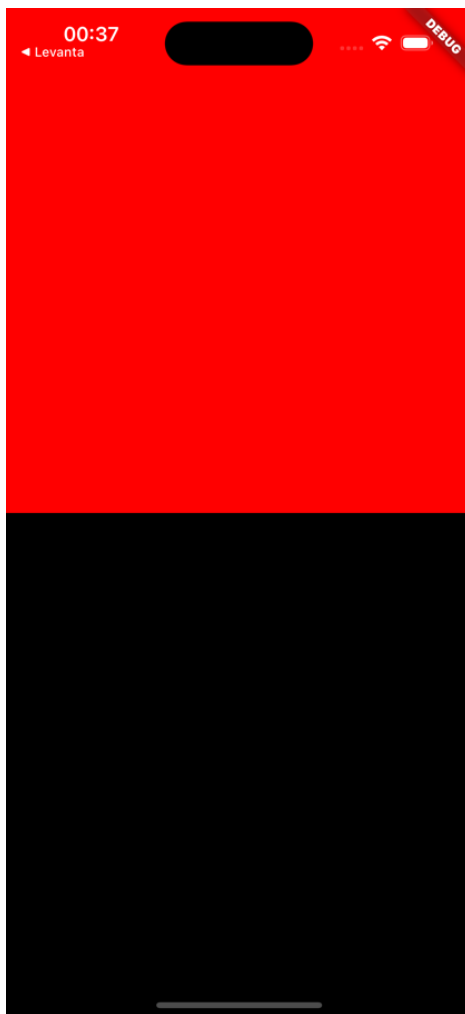


Рис. 3.7 – Виконання наявного прикладу

Цей приклад ілюструє, як бібліотека “server_side_ui” може бути використана в Flutter-додатках для створення інтерфейсу на основі JSON-даних і використання серверного рендерингу для побудови сторінок. Вона робить розробку інтерфейсу більш гнучкою і спрощує процес створення додатків зі складними інтерфейсами. Цей та інші приклади рендеру зберігаються разом з бібліотекою за шляхом “examples/json_examples”.

3.4 Інтеграція в проект

Бібліотеку було інтегровано в існуючий проект. В наведеному прикладі було використано заготовлений запит серверу для демонстрації прикладу який спрацює тільки якщо код зкомпільований для демонстрації. Сам запит зберігається зі всіма файлами які є сталими і підвантажуються з додатку.

```

class QuestionsRepository implements BaseApiRepository {
  @override
  final ApiRepository apiRepository;

  QuestionsRepository({required this.apiRepository});

  Future<dynamic> getQuestions() async {
    if (apiRepository.mockEnvironment) {
      await Future.delayed(const Duration(seconds: 3));
      return jsonDecode(await rootBundle.loadString(
        'packages/question_form/assets/mocked_response/questions.json'));
    }
    return await apiRepository.get('/questions', logError: true);
  }

  Future<void> sendQuestions(List<Map<String, dynamic>> questionsList) async {
    if (apiRepository.mockEnvironment) {
      Future.delayed(const Duration(seconds: 3));
      return;
    }

    await apiRepository.post(
      '/questions',
      data: {'questions': questionsList.toList()},
      logError: true,
    );
  }
}

```

Рис. 3.8 – Завантаження запитань з заготовленого прикладу

Для оновлення екрану використовується BLoC паттерн щоб оновити статус завантаження.

BLoC (Business Logic Component) - це патерн управління станом додатка у фреймворку Flutter. Він відокремлює бізнес-логіку від користувацького інтерфейсу та допомагає забезпечити однонаправлену передачу даних у вашому додатку. BLoC включає в себе вибірку даних (зазвичай через потоки), опрацювання подій та відправку стану інтерфейсу на зміни даних. Цей патерн допомагає покращити модульність, тестируемість і керованість вашого коду у Flutter-додатках.

Він характеризується використанням так званих потоків («Stream») для оновлення інтерфейсу користувача. В потоку є контроллер який відправляє дані та слухач який використовується для сортування, перетворення, з'єднання з іншим потоком даних та слухання потоку на наявні в ньому дані. Цей реактивний підхід досить популярний та широко використовується в програмуванні додатків на фреймворку.

```

class QuestionsScreenBloc {
  final StreamController<QuestionsState> _streamController =
    StreamController<QuestionsState>.broadcast();
  final QuestionsRepository _questionsRepository;

  QuestionsScreenBloc(this._questionsRepository);

  Stream<QuestionsState> get stream => _streamController.stream;

  Future<void> getQuestions() async {
    _streamController.add(
      QuestionsState(type: QuestionsStateType.loading),
    );

    try {
      final response = await _questionsRepository.getQuestions();
      _streamController.add(
        QuestionsState(type: QuestionsStateType.success, data: response),
      );
    } catch (e) {
      _streamController.add(
        QuestionsState(type: QuestionsStateType.error, data: e),
      );
    }
  }
}

```

Рис. 3.9 – Демонстрація контролера потоку та функції яка відповідає за надсилання в потік нових даних

StatefulWidget - це клас у фреймворку Flutter, який представляє собою віджет зі станом. У простому вигляді це клас, який може змінювати свій внутрішній стан та перерендерювати себе на основі цього стану. Віджети StatefulWidget використовуються для відображення інтерактивного контенту або динамічної інформації, яка може змінюватися під час виконання програми. Коли стан віджета змінюється, віджет перерендерюється, що дозволяє оновлювати вміст інтерфейсу під час взаємодії користувача або зміни даних.

На даному рисунку показана ініціалізація класу та реєстрація нових компонентів для рендерингу. Оскільки для рендеру таких складних компонентів займається сервер було прийнято рішення використати компонентний підхід та зареєструвати компоненти для використання що мінімізувало роботу серверу. Клас State в Flutter є ключовою частиною для зберігання та управління станом віджетів типу StatefulWidget. Він містить дані, які можуть змінюватися під час

виконання програми і відповідає за оновлення та перерендерення віджета при зміні цього стану.

```
class _QuestionsScreenState extends State<QuestionsScreen> {
  final List _registers = [
    CompleteScreenJson(),
    QuestionBoxColorJson(),
    QuestionsFlowScreenJson(),
    QuestionImageTypeJson(),
    SmilesScreenJson(),
    StepScreenJson(),
  ];

  late final QuestionsScreenBloc _questionsScreenBloc;
  final ServerSideUI _serverSideUI = ServerSideUI();

  @override
  void initState() {
    for (var element in _registers) {
      _serverSideUI.register(element);
    }

    _questionsScreenBloc = QuestionsScreenBloc(
      QuestionsRepository(apiRepository: context.read<ApiRep
    _questionsScreenBloc.getQuestions();

    _questionsScreenBloc.stream.listen((event) {
      if (event.type == QuestionsStateType.done) {
        Navigator.of(context).pop();
      }
    });

    super.initState();
  }
}
```

Рис. 3.10 – Ініціалізація компоненту SSR

State також має свій власний життєвий цикл і методи для керування станом та взаємодії з користувачем. Цей клас допомагає створювати інтерактивні та змінювані за реакцією на події віджети в Flutter.

```
@override
Widget build(BuildContext context) => Provider(
  create: (ctx) => _questionsScreenBloc,
  child: StreamBuilder<QuestionsState>(
    initialData: QuestionsState(type: QuestionsStateType.loading),
    stream: _questionsScreenBloc.stream,
    builder: (context, snapshot) {
      if (snapshot.data != null) {
        if (snapshot.data!.type == QuestionsStateType.loading) {
          return const ScreenWrapper(
            child: Center(
              child: CircularProgressIndicator(),
            ), // Center
          ); // ScreenWrapper
        }
        if (snapshot.data!.type == QuestionsStateType.success) {
          return _serverSideUI.build(snapshot.data?.data['screen']);
        }
      }
      return Container();
    },
  ), // StreamBuilder
); // Provider
}
```

Рис. 3.11 – Застосування функції рендеру

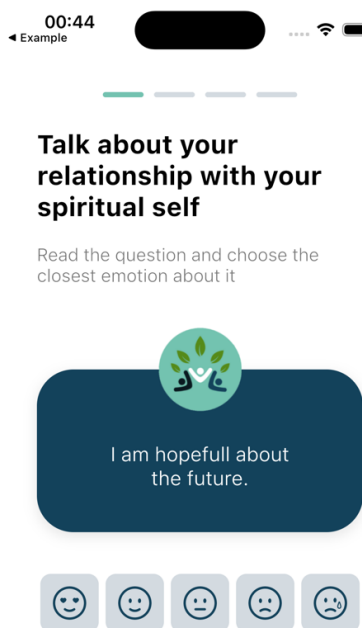


Рис. 3.12 – Результат виконання функції рендеру

Функція рендеру відпрацьовує тоді і тільки тоді коли запит був успішним. Результатом виконання функції є екрани запитань, екран завершення опитування та початковий екран. Наведений компонент програмного комплексу доступний для перегляду в репозиторію проекту.

3.5 Тестування компонентів бібліотеки

Оскільки тема роботи – бібліотека, то тестування бібліотеки використовує інші підходи такі як `widget test`, `unit test`, `e2e test`. Для тестування даної бібліотеки було вибрано `unit` тести оскільки вона не має надскладної логіки по відмальовуванню інтерфейсів і в той час для тестування компонентів проекту де використовується дана бібліотека можна використати заготовлені запити що спрощує створення тест-кейсів.

Середовище мови надає комфортні програмні інтерфейси для написання тестів що спрощує створення тест кейсів.

```

Run | Debug
group('ServerSideUI', () {
  late ServerSideUI serverSideUI;

  setUp(() {
    serverSideUI = ServerSideUI();
    // Setup mock data or dependencies if needed.
  });

  Run | Debug
  test('isRegistered returns true for registered path', () {
    bool result = serverSideUI.isRegistered('Row');
    expect(result, true);
  });

  Run | Debug
  test('isRegistered returns false for non-registered path', () {
    bool result = serverSideUI.isRegistered('nonexistent.path');
    expect(result, false);
  });
});

```

Рис. 3.13 – Приклад кейсу з функцією перевірки

Також було написано віджет тести для перевірки одного з компонентів початкового реєстру на те чи відбудеться його рендер.

На рисунку зображено результат тестування компонентів бібліотеки та їх позитивний результат. Було перевірено роботу початкового реєстру, функцію додавання нового компоненту в реєстр, булеву функцію яка відповідає за наявність компоненту в реєстрі, функцію парсингу аргументів компоненту та віджет тест на перевірку функції рендеру компонента.

```

✓ packages/common_ui/server_side_ui/test/unit/unit_test.dart 5/5 passed: 68ms
✓ ServerSideUI Unit Test 5/5 passed: 68ms
  ✓ isRegistered returns true for registered path 45ms
  ✓ isRegistered returns false for non-registered path 6.0ms
  ✓ register adds items to registry 4.0ms
  ✓ build returns a widget for a valid tree 9.0ms
  ✓ processNamedArguments processes named arguments correctly 4.0ms
✓ packages/common_ui/server_side_ui/test/widgets/widget_ui_test.dart 1/1 passed: 1.9s
  ✓ ServerSideUI build method creates a Text widget 1.9s

```

Рис. 3.14 – Результат тестування

Тестування потрібне для того щоб при внесені змін в бібліотеку не зіпсувати основну логіку що економить час в критичних випадках та дає гарантію що компонент відпрацює вірно.

Висновки до третього розділу

У розділі, присвяченому розробці серверно-сайд рендерінгу (SSR) на платформі Flutter, була проведена значуща робота з бібліотекою, документацією та тестуванням, що відіграють важливу роль у розробці та успішному впровадженні цієї технології.

По-перше, робота над бібліотекою була виконана з великою уважністю і педантичністю. Результатом є функціональна бібліотека, яка дозволяє легко інтегрувати SSR в мобільні додатки на Flutter. Вона має інтуїтивний інтерфейс та надійну роботу, що робить її важливим інструментом для розробників.

Друга складова — це документація, яка була створена з ретельною увагою до деталей. Ця документація є ключовою для спільноти розробників, оскільки вона дозволяє швидко ознайомитися з можливостями та функціональністю бібліотеки. Доступ до відкритої документації сприяє обміну знаннями та можливості спільної роботи над проектом.

Третя складова — тестування. Важливо відзначити, що тести виконані успішно і підтвердили коректну роботу бібліотеки. Це забезпечує високу надійність та стабільність роботи мобільних додатків, які використовують SSR.

Отже, взаємодія цих компонентів створила надійну та ефективну реалізацію SSR на платформі Flutter. Результати цієї роботи відкривають нові можливості для подальших досліджень та розробок у цьому напрямку і можуть служити основою для майбутніх проектів.

ВИСНОВОК

Магістерська робота є результатом важкої та наполегливої роботи, яка відображає високий рівень професійності та знань у сфері розробки мобільних додатків на основі Flutter та серверно-сайд рендерінгу (SSR). Дана робота розкриває великий потенціал цих технологій та їх вплив на сучасний ринок мобільних додатків.

У ході розгляду трьох розділів кваліфікаційної роботи були проаналізовані різні аспекти використання Server-Side Rendering (SSR) та фреймворку Flutter для реалізації веб-додатків та мобільних додатків.

У першому розділі "Аналіз напрямків використання SSR та фреймворку для реалізації" були розглянуті різні аспекти використання SSR рендерінгу на сучасних веб-сайтах та визначені переваги цього підходу. Також було проведено аналогію між SSR та фреймворком Flutter, щоб виділити основні схожості та відмінності.

Другий розділ "Розробка бібліотеки Server-Side Render та її інтеграція в проект" описує процес створення бібліотеки для SSR рендерінгу та її подальшу інтеграцію в реальний проект. У цьому розділі розглядаються деталі реалізації та способи використання цієї бібліотеки.

Третій розділ "Інтерфейс та порядок роботи з бібліотекою SSR рендеру" надає інформацію про взаємодію з бібліотекою SSR рендерінгу, описує порядок роботи з її інтерфейсом та можливостями.

У цілому, дослідження та практична реалізація SSR рендерінгу та фреймворку Flutter дозволили розглянути різні аспекти роботи з цими технологіями та визначити їхні переваги та можливості в сучасному програмуванні.

Завдяки детальному аналізу та дослідженню можливостей Flutter та SSR, робота розкриває важливі переваги цих технологій, такі як простота використання, швидкість розробки, динамічність та висока продуктивність. Вивчення та дослідження практичних аспектів інтеграції SSR в мобільних

додатках робить цю роботу важливим внеском у розвиток сфери мобільного програмування.

Маністерська робота покликана сприяти розумінню та використанню сучасних технологій у створенні високоефективних та інтерактивних мобільних додатків. Результати дослідження, аналізу та практичного впровадження цих технологій є важливим внеском у розвиток індустрії мобільних додатків та можуть бути використані як основа для подальших наукових досліджень та розробок.

Отже, ця маністерська робота не лише підкреслює важливість використання Flutter та SSR в сучасній розробці мобільних додатків, але і надає конкретні практичні рекомендації щодо їх впровадження та оптимізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Уїттакер Д. А, Арбон Д. В. How Google Tests Software / Д. А. Уїттакер, Д. В. Арбон. – СПб.: Addison-Wesley Professional, 2014. – 385 с.
2. Шалоуей А. Шаблони проектування. Новий підхід до об'єктно-орієнтованого аналізу та проектування / Алан Шаллоуей. - М.: "Вільямс", 2002. – 288 с.
3. Фрімен Е. Паттерни проектування / Е. Фрімен, Е. Фрімен. – СПб: Фабула, 2011. – 656 с.
4. Beginning Flutter: A Hands On Guide to App Development / Marco L. Napoli, 2019. – 528 с. 9. Flutter in Action / Eric Windmill, 2019. – 368 с.
5. The Dart Programming Language / Gilad Bracha, 2015. – 224 с.
6. Alessandria S. Programming Flutter: Native, Cross-Platform Apps the Easy Way / S. Alessandria, B. Kayfitz., 2021. – 648 с.
7. Aharony N. Library and Information Science research areas: A content analysis of articles from the top 10 journals 2007-8 / N. Aharony // Journal of Librarianship and Information Science. – 2012. – 44(1). – P. 27-35.
8. Milojevic S. The cognitive structure of Library and Information Science: Analysis of article title words / S. Milojevic, C. R. Sugimoto, E. Yan, Y. Ding // Journal of the American Society for Information Science and Technology. – 2011. – 62(10). – P. 1933-1953.
9. Nixon J. M. Core Journals in Library and Information Science: Developing a Methodology for Ranking LIS Journals / J. M. Nixon // College and Research Libraries. – 2014. – 75(1). – P. 66-90.
10. Sin S.-C. J. Longitudinal Trends in Internationalisation, Collaboration Types, and Citation Impact: A Bibliometric Analysis of Seven LIS Journals (1998–2008) / S.- C. J. Sin // Journal of Library and Information Studies. – 2011. – 9. – P. 27–49.
11. Литвинова С.Г. Компонентна модель хмаро орієнтованого навчального середовища загальноосвітнього навчального закладу / Світлана

- Григорівна Литвинова // Науковий вісник. – Випуск 35. – Серія: Педагогіка. Соціальна робота. – Ужгород: УЖНУ, 2015. – С. 99-107.
12. Фаулер М, Скотт К. UML у короткому викладі. Застосування стандартної мови об'єктного моделювання / М. Фаулер, К. Скотт. – М: «Світ», 1999 – 191 с.
 13. Joshi D. Building Cross-Platform Apps with Flutter and Dart: Build scalable apps for Android, iOS, and web from a single codebase / Deven Joshi., 2023. – 378 с.
 14. Walrath K. Dart: Up and Running / K. Walrath, S. Ladd., 2012. – 158 с.
 15. Leung T. Beginning Power Apps: The Non-Developer's Guide to Building Business Applications / Tim Leung., 2021. – 943 с. – (2).
 16. Feipeng L. Android Native Development Kit Cookbook / Liu Feipeng., 2013. – 346 с.
 17. Vaish G. High Performance iOS Apps: Optimize Your Code for Better Apps / Gaurav Vaish., 2016. – 464 с.
 18. Коренівська О.Л., Коротун О.В., Франчук В.О. “Гнучка система реєстрації в мобільному банкінгу за допомогою server-side rendering (SSR)” Тези VI Всеукраїнської науково-технічної конференції здобувачів вищої освіти і молодих учених «Комп’ютерні технології: інновації, проблеми, рішення», 29-30 листопада 2023 року. Житомир: "Житомирська політехніка", 2023. – 35-36 с.
 19. Вакалюк Т.А., Франчук В.О. “Організація state-менеджмента та асинхронності на сьогоdnішній лад” Тези XVI Всеукраїнська науково-практична web конференція аспірантів, студентів та молодих вчених «Комп’ютерні інтелектуальні системи та мережі», 21-23 березня 2023 року. Кривий ріг: " Криворізький національний університет", 2023. – 65-67 с.
 20. Вакалюк Т.А., Франчук В.О. “Сучасний стан і перспективи використання цифрових технологій в освіті та інших галузях” Тези XVI Всеукраїнська науково-практична web конференція аспірантів, студентів та молодих вчених «Комп’ютерні інтелектуальні системи та мережі», 21-23 березня

- 2023 року. Кривий ріг: " Криворізький національний університет", 2023. – 207-209 с.
- 21.Вакалюк.Т.А., Франчук В.О. “Інформаційні системи та технології” Тези XVI Всеукраїнська науково-практична web конференція аспірантів, студентів та молодих вчених «Комп’ютерні інтелектуальні системи та мережі», 21-23 березня 2023 року. Кривий ріг: " Криворізький національний університет", 2023. – 209-211 с.
- 22.Офіційна документація Flutter [Електронний ресурс] // Flutter – Режим доступу до ресурсу: <https://flutter.dev/docs>.
- 23.Офіційна документація мови програмування Dart [Електронний ресурс] – Режим доступу до ресурсу: <https://dart.dev/guides>.
- 24.Плюси і мінуси об’єктно-орієнтованого програмування [Електронний ресурс] - 2015. Режим доступу: <http://damp.biz/plyusi-i-minusi-obyektno-oriyentovanogo-programuvannya/>.
- 25.Документація до пакету flutter_bloc [Електронний ресурс] – Режим доступу до ресурсу: <https://bloclibrary.dev/>.
26. Orozco J. V. RESTful Services: what they are, and what they aren't [Електронний ресурс] / Juan Villalobos Orozco. – 2019. – Режим доступу до ресурсу: <https://www.brainstobytes.com/restful-services-what-they-are-and-what-they-are-not/>.
- 27.Flutter – Вікіпедія [Електронний ресурс] / Режим доступу: <https://uk.wikipedia.org/wiki/Flutter>
- 28.Чому Flutter ідеально підходить для кросплатформенної розробки | DOU [Електронний ресурс] / Режим доступу: <https://dou.ua/forums/topic/46244/>

ДОДАТКИ

Вхідні дані для робочого прикладу

```

{
  "Scaffold": {
    "body": {
      "Column": {
        "children": [
          {
            "Expanded": {
              "child": {
                "Container": {
                  "color": {
                    "Color": "#FF0000"
                  }
                }
              }
            }
          },
          {
            "Expanded": {
              "child": {
                "Container": {
                  "color": {
                    "Color": "#000000"
                  }
                }
              }
            }
          }
        ]
      }
    }
  }
}

```

Приклад запиту з робочого додатку

```

{
  "screen": {
    "QuestionsFlowScreen": {
      "screens": [
        {
          "StepScreen": {
            "text": "<h1>Your new Relationship Scorecard</h1>"
          }
        },
        {
          "SmilesScreen": {
            "title": "<h5>Talk about your relationship with your spiritual self</h5>\n\n<s1>Read the question
and choose the closest emotion about it</s1>",
            "image": "QuestionImageType.plant",
            "questionBoxColor": "QuestionBoxColor.primary",
            "finalText": "<h5>Your relationship with your spiritual self</h5>",
            "questionsList": [
              "I am hopefull about the future.",
              "I belong to a faith community who accepts me and cares about me.",
              "I belong to a community in which I can be honest and who I can confide in.",
              "I am happy with my current involvement in my faith community."
            ]
          }
        },
        {
          "SmilesScreen": {
            "title": "<h5>Talk about your relationship with yourself</h5>\n\n<s1>Read the question and choose
the closest emotion about it</s1>",
            "image": "QuestionImageType.heard",
            "questionBoxColor": "QuestionBoxColor.secondary",
            "finalText": "<h5>Your relationship with yourself</h5>",
            "questionsList": [
              "I have places where I belong, am valued, and bring value to others.",
              "I can recognize my feelings, can name them, and know their source.",
              "I have specific goals around learning new things for my personal growth.",
              "I know healthy ways to cope with my feelings and stress.",
              "I am happy with my physical health."
            ]
          }
        },
        {
          "SmilesScreen": {
            "title": "<h5>Talk about your relationship with others</h5>\n\n<s1>Read the question and choose the
closest emotion about it</s1>",
            "image": "QuestionImageType.peoples",
            "questionBoxColor": "QuestionBoxColor.primary",
            "finalText": "<h5>Your relationship with others</h5>",
            "questionsList": [
              "I have meaningful, supportive relationships with people in my family.",

```

```

        "I have a good working relationship with most of the people I work with.",
        "I have meaningful, supportive relationships with people outside of my family.",
        "My job provides benefits like health insurance, retirement, paid time off, etc.",
        "I have the support I need to get to work. (i.e., transportation, child care, etc.)",
        "I know how to ask for what I need, know who to ask and when.",
        "I see the chance to grow my income and skills at my job."
    ]
}
},
{
    "SmilesScreen": {
        "title": "<h5>Talk about your relationship with the physical world</h5>\n\n<s1>Read the question
and choose the closest emotion about it</s1>",
        "image": "QuestionImageType.city",
        "questionBoxColor": "QuestionBoxColor.secondary",
        "finalText": "<h5>Your relationship with physical world</h5>",
        "questionsList": [
            "I have safe, stable housing that I can afford.",
            "My income is stable and meets my expenses.",
            "I know where all my money goes.",
            "I am able to protect my assets and income. (i.e., insurance, etc.)",
            "I am investing in my future financial health.",
            "I have an emergency savings account.",
            "I am acquiring assets."
        ]
    }
},
{
    "CompleteScreen": {
        "title": "<h5>Thank you!</h5>\n\n<s1>Talk over your latest results with your coach and celebrate
your progress!</s1>"
    }
}
}
}
}

```