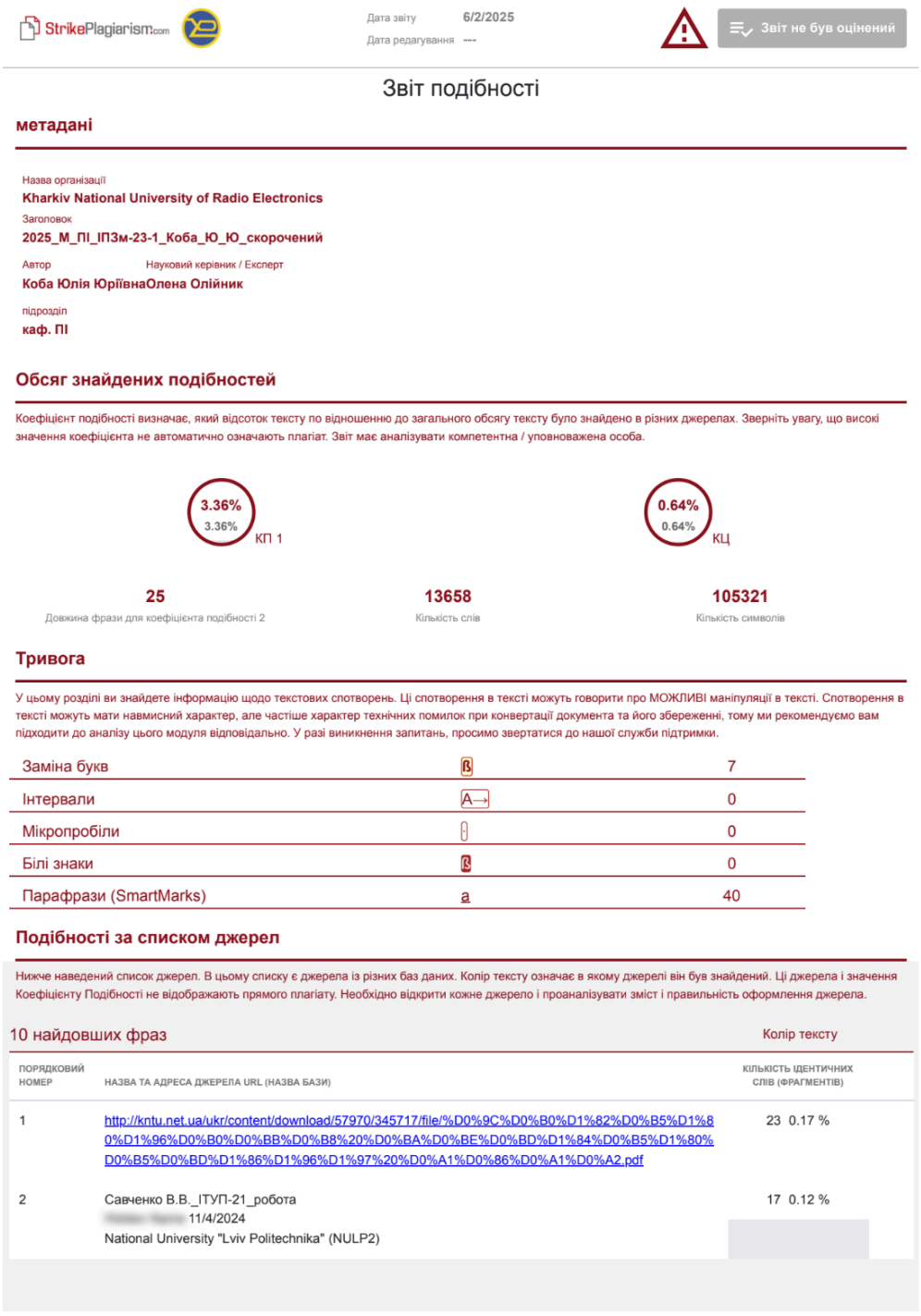
ДОДАТОК А

ЗВІТ РЕЗУЛЬТАТІВ ПЕРЕВІРКИ НА УНІКАЛЬНІСТЬ ТЕКСТУ В БАЗІ ХНУРЕ



StrikePlagiarism.com

Дата звіту 6/2/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_М_ПІ_ІПЗм-23-1_Коба_Ю_Ю_скорочений
Автор Науковий керівник / Експерт
Коба Юлія ЮріївнаОлена Олійник
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

3.36% 3.36% КП 1

0.64% 0.64% КЦ

25 Довжина фрази для коефіцієнта подібності 2
13658 Кількість слів
105321 Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	7
Інтервали	A→	0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)	a	40

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	http://kntu.net.ua/ukr/content/download/57970/345717/file/%D0%9C%D0%B0%D1%82%D0%B5%D1%80%D1%96%D0%B0%D0%BB%D0%B8%20%D0%BA%D0%BE%D0%BD%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D1%96%D1%97%20%D0%A1%D0%86%D0%A1%D0%A2.pdf	23 0.17 %
2	Савченко В.В._ІТУП-21_робота [redacted] 11/4/2024 National University "Lviv Politechnika" (NULP2)	17 0.12 %

Рисунок А.1 – Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

ДОДАТОК Б

КОД СТВОРЕНИЙ У SQL-РЕДАКТОРІ SUPABASE

```sql
-- Create a table for public users
create table users (
  id uuid references auth.users not null primary key,
  email text unique,
  full_name text,
  avatar_url text,

  constraint email_length check (char_length(email) >= 3)
);
-- Set up Row Level Security (RLS)
alter table users
  enable row level security;

create policy "Public users are viewable by everyone." on users
  for select using (true);

create policy "Users can insert their own profile." on users
  for insert with check ((select auth.uid()) = id);

create policy "Users can update own profile." on users
  for update using ((select auth.uid()) = id);

-- This trigger automatically creates a profile entry when a new user signs up via Supabase Auth.
create function public.handle_new_user()
returns trigger
set search_path = ''
as $$
begin
  insert into public.users (id, full_name, avatar_url)
  values (new.id, new.raw_user_meta_data->>'full_name', new.raw_user_meta_data->>'avatar_url');
  return new;
end;
$$ language plpgsql security definer;
create trigger on_auth_user_created
  after insert on auth.users
  for each row execute procedure public.handle_new_user();

-- Set up Storage
insert into storage.buckets (id, name)
  values ('avatars', 'avatars');

-- Set up access controls for storage.
create policy "Avatar images are publicly accessible." on storage.objects
  for select using (bucket_id = 'avatars');

create policy "Anyone can upload an avatar." on storage.objects
  for insert with check (bucket_id = 'avatars');

create policy "Anyone can update their own avatar." on storage.objects
  for update using ((select auth.uid()) = owner) with check (bucket_id = 'avatars');
```

Рисунок Б.1 – Код для роботи з профілями користувачів та зберіганням аватарів

```sql
CREATE OR REPLACE FUNCTION public.update_partner_average_rating()
RETURNS trigger
LANGUAGE plpgsql
AS $function$
BEGIN
    -- Update the average mark for each partner using weighted average
    UPDATE public.partners
    SET average_mark = COALESCE((
        SELECT
            CASE
                WHEN SUM(c.coefficient) > 0 THEN SUM(m.mark * c.coefficient) / SUM(c.coefficient)
                ELSE 0  -- Set to 0 if there are no coefficients
            END
        FROM public.marks m
        JOIN public.criterias c ON m.criteria_id = c.id
        WHERE m.partner_id = partners.id
    ), 0)  -- Use COALESCE to set average_mark to 0 if NULL
    WHERE id IN (SELECT DISTINCT m.partner_id FROM public.marks m);  -- Update only partners with marks

    RETURN NEW;  -- Return the new criteria row
END;
$function$;
```

Рисунок Б.2 – Код для оновлення середньої оцінки партнера

```sql
CREATE TRIGGER update_partner_ratings_after_update_criteria
AFTER UPDATE ON public.criterias
FOR EACH ROW
EXECUTE FUNCTION public.update_partner_average_rating();

CREATE TRIGGER update_partner_ratings_after_delete_criteria
AFTER DELETE ON public.criterias
FOR EACH ROW
EXECUTE FUNCTION public.update_partner_average_rating();

CREATE TRIGGER update_partner_ratings_after_insert_criteria
AFTER INSERT ON public.criterias
FOR EACH ROW
EXECUTE FUNCTION public.update_partner_average_rating();

CREATE TRIGGER update_partner_ratings_after_update_mark
AFTER UPDATE ON public.marks
FOR EACH ROW
EXECUTE FUNCTION public.update_partner_average_rating();

CREATE TRIGGER update_partner_ratings_after_delete_mark
AFTER DELETE ON public.marks
FOR EACH ROW
EXECUTE FUNCTION public.update_partner_average_rating();

CREATE TRIGGER update_partner_ratings_after_insert_mark
AFTER INSERT ON public.marks
FOR EACH ROW
EXECUTE FUNCTION public.update_partner_average_rating();
```

Рисунок Б.3 – Код додавання тригерів

```sql
CREATE OR REPLACE FUNCTION get_partner_marks(
    CUSTOM_CRITERIA_ID INT,
    CUSTOM_USER_ID UUID
)
RETURNS TABLE (
    id bigint,
    name TEXT,
    type TEXT,
    avatar_url TEXT,
    average_mark float,
    mark_id bigint,
    mark bigint,
    criteria_id bigint
)
LANGUAGE plpgsql AS $$
BEGIN
    RETURN QUERY
    SELECT
        p.id,
        p.name,
        p.type,
        p.avatar_url,
        p.average_mark,
        m.id AS mark_id,
        m.mark AS mark,
        m.criteria_id
    FROM public.marks m
    FULL OUTER JOIN public.partners p ON m.partner_id = p.id
    AND m.criteria_id = CUSTOM_CRITERIA_ID
    WHERE p.user_id = CUSTOM_USER_ID;
END;
$$;
```

Рисунок Б.4 – Код для повернення списку партнерів з оцінками

```sql
CREATE OR REPLACE FUNCTION get_marks_and_criteria(
    p_partner_id bigint,
    p_user_id UUID
)
RETURNS TABLE (
    id bigint,
    name TEXT,
    coefficient float,
    mark_id bigint,
    mark bigint,
    partner_id bigint
)
LANGUAGE plpgsql AS $$
BEGIN
    RETURN QUERY
    SELECT
        c.id,
        c.name,
        c.coefficient
        ,
        m.id AS mark_id,
        m.mark,
        m.partner_id
    FROM public.marks m
    FULL OUTER JOIN public.criterias c ON m.criteria_id = c.id
    AND m.partner_id = p_partner_id
    WHERE c.user_id = p_user_id;
END;
$$;
```

Рисунок Б.5 – Код для повернення списку критеріїв з оцінками

```sql
CREATE OR REPLACE FUNCTION get_criteria_correlations(
    p_user_id UUID
)
RETURNS TABLE (
    criteria_id_1 BIGINT,
    criteria_name_1 TEXT,
    criteria_id_2 BIGINT,
    criteria_name_2 TEXT,
    occurrences BIGINT,
    correlation NUMERIC
)
LANGUAGE plpgsql AS $$
BEGIN
    RETURN QUERY
    SELECT
        c1.id AS criteria_id_1,
        c1.name AS criteria_name_1,
        c2.id AS criteria_id_2,
        c2.name AS criteria_name_2,
        COUNT(*) AS occurrences,
        ROUND(CAST(CORR(m1.mark, m2.mark) AS NUMERIC), 2) AS correlation
    FROM public.marks m1
    JOIN public.marks m2 ON m1.partner_id = m2.partner_id AND m1.criteria_id < m2.criteria_id
    JOIN public.criterias c1 ON m1.criteria_id = c1.id
    JOIN public.criterias c2 ON m2.criteria_id = c2.id
    WHERE c1.user_id = p_user_id AND c2.user_id = p_user_id
    GROUP BY c1.id, c2.id
    ORDER BY correlation DESC;
END;
$$;
```

Рисунок Б.6 – Код функції для кореляції між критеріями

ДОДАТОК В

ПРОГРАМНИЙ КОД ДЛЯ ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ

```
    return StoreConnector(
      distinct: true,
      converter: _ViewModel.new,
      ignoreChange: _ViewModel.ignoreChange,
…
class _ViewModel extends TableViewModel<Partner, GeneralTablePointer> {
  _ViewModel(super.store);

  static bool ignoreChange(AppState state) =>
      state.tablesState.getTable<Partner, GeneralTablePointer>().isLoading &&
      state.tablesState.getTable<Partner, GeneralTablePointer>()
          .items.isNotEmpty;
}
```

Рисунок В.1 – Уникнення непотрібного повторного відтворення коду віджетів

```
class PartnersPage extends StatelessWidget {
  const PartnersPage( { super.key });

  @override Widget build(BuildContext context) {
    return Scaffold(
      body: const SafeArea(
        child: Column(
          children: [
            PartnersTableActionBar(),
            Expanded(child: PartnersTable()),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        child: const Icon(Icons.add),
        onPressed: () => StoreProvider.of<AppState>(context).dispatch(
          OpenPageAction(Destination.createPartner),
        ),
      ),
    );
  }
}
class PartnersPage extends StatelessWidget {
  const PartnersPage( { super.key });

  @override Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: Column(
          children: [
            PartnersTableActionBar(),
            Expanded(child: PartnersTable()),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        child: Icon(Icons.add),
        onPressed: () => StoreProvider.of<AppState>(context).dispatch(
          OpenPageAction(Destination.createPartner),
        ),
      ),
    );
  }
}
```

Рисунок В.2 – Код із константними віджетами та без них

```
class PartnersTableActionBar extends StatefulWidget {
  const PartnersTableActionBar( { super.key });

  @override State<PartnersTableActionBar> createState() =>
      _PartnersTableActionBarState();
}

class _PartnersTableActionBarState extends State<PartnersTableActionBar> {
  @override Widget build(BuildContext context) {
    return StoreConnector(
      distinct: true,
      converter: TableViewModel<Partner, GeneralTablePointer>.new,
      builder: (context, viewModel) => Padding(
        padding: const EdgeInsets.all(8.0),
        child: Row(
          children: [
            Expanded(
              child: SearchField(
                hintText: context.strings.searchPartners,
                isLoading: viewModel.isLoading,
                search: viewModel.search,
              ),
            ),
            const SizedBox(width: 16),
            FilterButton(
              isEmpty: viewModel.filter.isFilterByEmpty,
              onPressed: () {
                //TODO: Open filters page
              },
            ),
          ],
        ),
      ),
    );
  }
}

class PartnersTableActionBar extends StatelessWidget {
  const PartnersTableActionBar( { super.key });

  @override Widget build(BuildContext context) {
    return StoreConnector(
      distinct: true,
      converter: TableViewModel<Partner, GeneralTablePointer>.new,
      builder: (context, viewModel) => Padding(
        padding: const EdgeInsets.all(8.0),
        child: Row(
          children: [
            Expanded(
              child: SearchField(
                hintText: context.strings.searchPartners,
                isLoading: viewModel.isLoading,
                search: viewModel.search,
              ),
            ),
            const SizedBox(width: 16),
            FilterButton(
              isEmpty: viewModel.filter.isFilterByEmpty,
              onPressed: () {
                //TODO: Open filters page
              },
            ),
          ],
        ),
      ),
    );
  }
}
```

Рисунок В.3 – Два випадки візуалізації тих самих компонентів за допомогою віджетів Stateful або Stateless

```dart
class PartnersPage extends StatelessWidget {
  const PartnersPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: Column(
          children: [
            Expanded(
              child: StoreConnector(
                distinct: true,
                converter: _ViewModel.new,
                ignoreChange: _ViewModel.ignoreChange,
                builder: (context, viewModel) {
                  return LoadMoreScrollListener(
                    loadMore: viewModel.downloadItems,
                    child: TableRefreshIndicator<Partner, GeneralTablePointer>(
                      builder: (context, iosRefreshIndicator) =>
                          CustomScrollView(
                        slivers: [
                          iosRefreshIndicator,
                          if (viewModel.items.isEmpty)
                            SliverFillRemaining(
                              hasScrollBody: false,
                              child: EmptyTablePlaceholder(
                                isLoading: viewModel.isLoading,
                                title: Text(context.strings.noPartnersFound),
                                subtitle: Text(
                                  viewModel.filter.isEmpty
                                      ? context.strings.addYourFirstPartner
                                      : context.strings.changeYourSearchQuery,
                                ),
                                icon: Icon(
                                   HomePageTabType.partners.activeIconData),
                              ),
                            )
                          else
                            SliverList(
                              delegate: SliverChildBuilderDelegate(
                                (context, index) => PartnerCard(
                                  key: ValueKey(viewModel.items[index].id),
                                  onPressed: () {},
                                  partner: viewModel.items[index],
                                ),
                                childCount: viewModel.items.length,
                              ),
                            ),
                        ],
                      ),
                    ),
                  );
                },
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

```dart
class PartnersPage extends StatelessWidget {
  const PartnersPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: const SafeArea(
        child: Column(
          children: [
            Expanded(child: PartnersTable()),
          ],
        ),
      ),
    );
  }
}
```

Рисунок В.4 – Код з використанням довгих та коротких методів

```
SliverList(
                    delegate: SliverChildBuilderDelegate(
                      (context, index) => PartnerCard(
                        key: ValueKey(viewModel.items[index].id),
                        onPressed: () {},
                        partner: viewModel.items[index],
                      ),
                      childCount: viewModel.items.length,
                    ),
                  )
…
   SliverList(
                    delegate: SliverChildBuilderDelegate(
                      (context, index) => _buildPartner(
                        context,
                        viewModel.items[index],
                      ),
                      childCount: viewModel.items.length,
                    ),
                  ),
              ],
            ),
          ),
        );
},
    );
}

Widget _buildPartner(BuildContext context, Partner partner) {
  return ListTile(
      onTap: () {},
      leading: CircleIconPreview.user(
        imageUrl: partner.avatarUrl,
        radius: 28,
      ),
      title: RichText(
        text: TextSpan(
          text: partner.name,
          style: Theme.of(context).textTheme.bodyLarge,
          children: [
            TextSpan(
              text: ' (${partner.type.getDisplayText(context)})',
              style: Theme.of(context).textTheme.bodySmall,
            ),
          ],
        ),
      ),
      subtitle: RatingView(rate: partner.averageMark ?? 0),
    );
}
```

Рисунок В.5 – Код побудови списку з методами та окремими віджетами

```
SliverList(
                    delegate: SliverChildBuilderDelegate(
                      (context, index) => PartnerCard(
                        key: ValueKey(viewModel.items[index].id),
                        onPressed: () {},
                        partner: viewModel.items[index],
                      ),
                      childCount: viewModel.items.length,
                    ),
                  )
…
SingleChildScrollView(
                    child: Column(
                      children: viewModel.items
                          .map((item) => PartnerCard(
                                key: ValueKey(item.id),
                                onPressed: () {},
                                partner: item,
                              ))
                          .toList(),
                    ),
                  )
```

Рисунок В.6 – Код з використанням SliverList і SingleChildScrollView

```
Future<List<Color>> getAverageColors(List<SimpleImageProvider> images) async {
  final bytes = await Future.wait(images.map((item) => item.toBytes()));

  return compute((bytes) {
    final colors = bytes.map(_getAverageColor).toList();
    return colors;
  }, bytes, );
}
… Future<List<Color>> getAverageColors(List<SimpleImageProvider> images) async {
  final bytes = await Future.wait(images.map((item) => item.toBytes()));

  return bytes.map(_getAverageColor).toList();
}
```

Рисунок В.7 – Код із використанням ізолятів і без них

```
Visibility(
        visible: isLoading,
        replacement: Row(
          mainAxisAlignment: MainAxisAlignment.center,
          mainAxisSize: MainAxisSize.min,
          children: [
            IconTheme(
              data: IconTheme.of(context).copyWith(color: color, size: 32),
              child: icon,
            ),
            const SizedBox(width: 10),
            Flexible(
              child: DefaultTextStyle(
                    style: Theme.of(context).textTheme.bodyLarge!.copyWith(
                          color: color,
                        ),
                    child: title,
                  ),
            ),
          ],
        ),
        child: const StyledLoader.primary(size: 32),
…
    return Container(
      alignment: Alignment.center,
      padding: const EdgeInsets.symmetric(vertical: 16, horizontal: 32),
      child: isLoading
          ? const StyledLoader.primary(size: 32)
          : Row(
              mainAxisAlignment: MainAxisAlignment.center,
              mainAxisSize: MainAxisSize.min,
              children: [
                IconTheme(
                  data: IconTheme.of(context).copyWith(color: color, size: 32),
                  child: icon,
                ),
                const SizedBox(width: 10),
                Flexible(
                  child: DefaultTextStyle(
                    style: Theme.of(context).textTheme.bodyLarge!.copyWith(
                          color: color,
                        ),
                    child: title,
                  )
                ),
              ],
            ),
    );
```

Рисунок В.8 – Код із Opacity та без неї

```
List<Partner> partners = [];

final values =
    result.map((item) => PartnerDto.fromJson(item).toDomain()).toSet();

for (final value in values) {
  partners.add((await getPartnerDetails(value.id)).result!);
}
…
final values =
    result.map((item) => PartnerDto.fromJson(item).toDomain()).toSet();

final partners =
    await Future.wait(values.map((item) => getPartnerDetails(item.id)));
```

Рисунок В.9 – Послідовні та паралельні запити

```
return Image.network(
    widget.imageUrl,
    placeholder: (context, url) => const CircularProgressIndicator(),
    errorWidget: (context, url, error) => const Icon(Icons.error), );

return CachedNetworkImage(
    imageUrl: widget.imageUrl,
    placeholder: (context, url) => CircularProgressIndicator(),
    errorWidget: (context, url, error) => Icon(Icons.error), );
```

Рисунок В.10 – Звичайний віджет і віджет, що підтримує кешування даних

ДОДАТОК Г
СЛАЙДИ ПРЕЗЕНТАЦІЇ

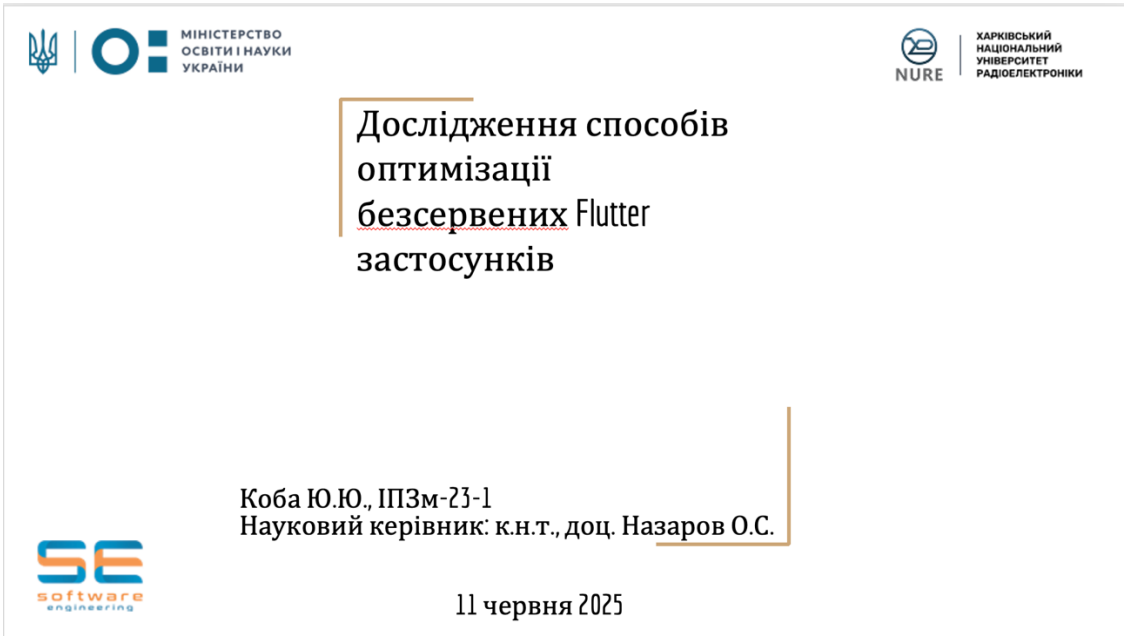


Рисунок Г.1 – Слайд 1 презентації

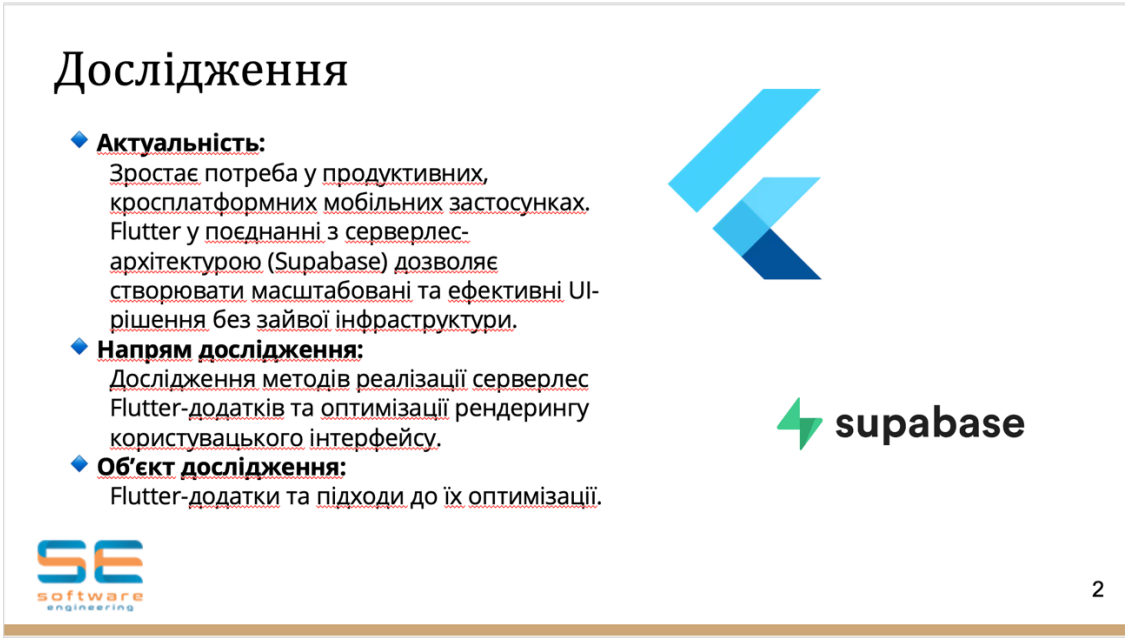


Рисунок Г.2 – Слайд 2 презентації

Огляд літератури (аналогів)

Таблиця 2.1 – Порівняння методів

Автори	Цілі дослідження	Використані методи	Результати та висновки
Luo et al. (2024)	Оптимізація розміщення компонентів	Моделі оптимізації	Підвищення продуктивності серверлес-додатків
Raza et al. (2023)	Конфігурація серверлес-додатків	Статистичне навчання	Зменшення затримок у роботі додатків
Mampage et al. (2023)	Масштабування додатків	Глибоке підкріплювальне навчання	Ефективне використання ресурсів
Nanavati et al. (2024)	Оптимізація Flutter-додатків	Техніки фінального налаштування	Підвищення продуктивності та стабільності

- Висновки з огляду:
  - серверлес-технології як інструмент оптимізації;
  - роль методів машинного навчання;
  - кросплатформні додатки: інноваційні підходи.
- Прогалини:
  - відсутність уніфікованих стандартів для серверлес-платформ;
  - недостатня увага до безпеки;
  - міжплатформна інтеграція.

Рисунок Г.3 – Слайд 3 презентації

Постановка задачі

- **Формулювання проблеми:**
  Розробка Flutter-додатків у серверлес-середовищі супроводжується проблемами низької продуктивності інтерфейсу при масштабуванні. Виникає потреба в ефективній оптимізації рендерингу UI.
- **Мета:**
  Дослідити продуктивність методів реалізації серверлес-додатків на Flutter і запропонувати оптимізації рендерингу користувацького інтерфейсу.
- **Очікувані результати:**
  - Розробка демонстраційного застосунку з використанням Supabase
  - Застосування різних підходів до оптимізації рендерингу
  - Аналіз ефективності цих підходів за допомогою регресійних моделей

Рисунок Г.4 – Слайд 4 презентації

Методологія

🔷 **Використані методи:**

- Порівняльний аналіз хмарних платформ (Supabase, Firebase тощо) з використанням методів прийняття рішень (TOPSIS, WSM)
- Побудова регресійних моделей без залежностей між факторами для оцінки ефективності оптимізацій UI

🔷 **Інструменти та технології:**

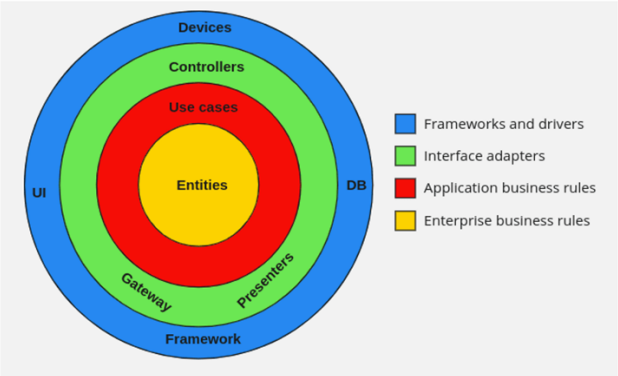
- **Flutter** — кросплатформна розробка мобільних додатків
- **Supabase** — серверлес backend (аутентифікація, база даних, функції)
- **Visual Studio Code** — середовище розробки
- **PostgreSQL** — основа для збереження даних
- **Flutter DevTools** — профілювання рендерингу

SE software engineering

5

Рисунок Г.5 – Слайд 5 презентації

Архітектура система для проведення експериментального дослідження

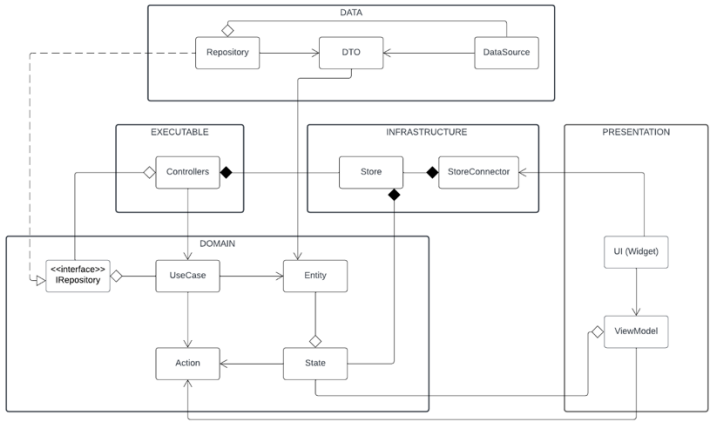


SE software engineering

6

Рисунок Г.6 – Слайд 6 презентації

Архітектура система для проведення експериментального дослідження

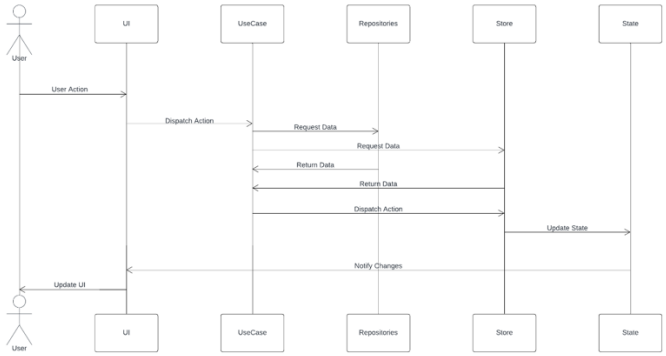


software engineering

7

Рисунок Г.7 – Слайд 7 презентації

Архітектура система для проведення експериментального дослідження



software engineering

8

Рисунок Г.8 – Слайд 8 презентації

Опис програмного забезпечення, що було використано у дослідженні

🔹 **Процес розробки:**
- Побудовано експериментальний Flutter-додаток з інтеграцією Supabase
- Реалізовано різні методи оптимізації рендерингу інтерфейсу
- Проведено заміри продуктивності та порівняльний аналіз
- Застосовано методи аналітики для оцінки ефективності оптимізацій

🔹 **Технологічний стек:**
- **Мова програмування:** Dart
- **Фреймворк:** Flutter
- **Backend-сервіс:** Supabase (PostgreSQL, Auth, Edge Functions)
- **Середовище розробки:** Visual Studio Code
- **Інструменти для тестування та аналізу:** DevTools, логування, регресійний аналіз

SE software engineering

9

Рисунок Г.9 – Слайд 9 презентації

Зміст проведеного експерименту

🔹 **Методи:**
- **TOPSIS** (Technique for Order Preference by Similarity to Ideal Solution)
- **WSM** (Weighted Sum Method)

🔹 **Вхідні дані:**
- Документація до безсерверних сервісів
- Прайсинг
- Відгуки з форумів

🔹 **Критерії:**
- Популярність серед Flutter-розробників
- Кількість безкоштовних запитів
- Вартість запитів після перевищення квоти
- Безкоштовний обсяг бази даних
- Кількість мов програмування

SE software engineering

10

Рисунок Г.10 – Слайд 10 презентації

Результати експерименту

Таблиця 4.1 – Моделювання задачі прийняття рішень для вибору хмарної платформи для безсерверних застосунків

Платформа	Популярність серед Flutter-розробників	Кількість безкоштовних запитів	Вартість запитів після перевищення квоти	Безкоштовний обсяг бази даних	Кількість мов програмування
Firebase	10	1,5	0,39	1	7
AWS Amplify	8	0,5	0,3	5	10
Azure Functions	7	1	0,2	32	30
DynamoDB	6	0,5	0,3	5	10
Supabase	7	10	0	5	7

11

Рисунок Г.11 – Слайд 11 презентації

Результати експерименту

Таблиця 4.3 – Нормалізовані дані таблиці 4.2.

Платформа	Популярність серед Flutter-розробників	Кількість безкоштовних запитів	Вартість запитів після перевищення квоти	Безкоштовний обсяг бази даних	Кількість мов програмування
Firebase	1	0,15	0	0,3	0
AWS Amplify	0,5	0,05	0,23	0,16	0,13
Azure Functions	0,25	0,1	0,49	1	1
DynamoDB	0	0,05	0,23	0,16	0,13
Supabase	0,25	1	1	0,16	0

12

Рисунок Г.12 – Слайд 12 презентації

Результати експерименту

Таблиця 4.4 – Лінійна адитивна розгортка з ваговими коефіцієнтами.

Платформа	Популярність серед Flutter-розробників	Кількість безкоштовних запитів	Вартість запитів після перевищення квоти	Безкоштовний обсяг бази даних	Кількість мов програмування	Z*
Firebase	1	0,15	0	0,3	0	0,115171
AWS Amplify	0,5	0,05	0,23	0,16	0,13	0,086164
Azure Functions	0,25	0,1	0,49	1	1	0,315737
Supabase	0,25	1	1	0,16	0	0,382928
Нормуючі множники	0,5	0,76923077	0,581395	0,617284	0,884956	
Вагові коефіцієнти	0,05	0,3	0,2	0,3	0,05	

13

Рисунок Г.13 – Слайд 13 презентації

Аналіз отриманих результатів

З розрахунку лінійної адитивної розгортки випливає, що Supabase (переважає з найбільшим коефіцієнтом (0,382928)), є найкращим вибором для дослідження оптимізації безсерверних Flutter-застосунків завдяки своїм високим оцінкам за критеріями економічної вигоди (безкоштовні запити, зберігання даних) і популярності серед Flutter-спільноти. Інші сервіси, такі як Azure Functions і Firebase, також мають свої переваги, але не можуть забезпечити таку економічну ефективність, як Supabase. AWS Amplify виявився найменш привабливим варіантом через його високу вартість та обмеження за кількістю безкоштовних запитів і зберігання даних.

14

Рисунок Г.14 – Слайд 14 презентації

Зміст проведеного експерименту

🔹 **Методи:**
- Побудова регресійних моделей без залежностей між факторами для оцінки ефективності оптимізацій UI

🔹 **Вхідні дані:**
- Flutter-застосунок (UI-компоненти, логіка)

🔹 **Критерії:**
- Час рендерингу інтерфейсу

🔹 **Послідовність:**
- Розробка базового Flutter-додатку
- Вимірювання ключових метрик при виконанні однакових задач з використанням і без використання оптимізаційних технік

🔹 **Вимірювання:**
- Використання Timer, вбудованого у Flutter-фреймворк

software engineering

15

Рисунок Г.15 – Слайд 15 презентації

Результати експерименту

Фактор впливу	Опис
x_1	Надмірна переробка віджетів
x_2	Кількість константних віджетів
x_3	Кількість Stateful віджетів
x_4	Обсяг build-методів (у рядках)
x_5	Наявність helper-методів
x_6	Рендеринг усіх віджетів у дереві
x_7	Кількість віджетів з використанням Opacity
x_8	Використання isolates
x_9	Кількість паралельних запитів
x_{10}	Кешування картинок

software engineering

16

Рисунок Г.16 – Слайд 16 презентації

Результати експерименту

Таблиця 4.6 – Верхні та нижні значення факторів впливу.

Фактор впливу	Верхнє значення	Нижнє значення
x_1	on	off
x_2	20	4
x_3	20	2
x_4	120	30
x_5	on	off
x_6	on	off
x_7	20	0
x_8	on	off
x_9	20	0
x_{10}	on	off

17

Рисунок Г.17 – Слайд 17 презентації

Результати експерименту

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0,974760454
R Square	0,950157943
Adjusted R Square	0,932971027
Standard Error	0,547145948
Observations	40

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	10	165,5023718	16,55023718	55,28379493	3,71564E-16
Residual	29	8,681691966	0,299368688		
Total	39	174,1840638			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95,0%*	*Upper 95,0%*
Intercept	5,505615896	0,336548243	16,35906888	3,51331E-16	4,817297453	6,19393434	4,817297453	6,19393434
X Variable 1	0,077319531	0,128494983	0,601731909	0,55202822	-0,185482216	0,340121278	-0,185482216	0,340121278
X Variable 2	0,336867997	0,214011174	1,57406733	0,126319759	-0,100833999	0,774569993	-0,100833999	0,774569993
X Variable 3	0,445928093	0,201418678	2,213936156	0,034852876	0,033980643	0,857875542	0,033980643	0,857875542
X Variable 4	0,269740417	0,17713534	1,52279278	0,138641729	-0,092542031	0,632022865	-0,092542031	0,632022865
X Variable 5	0,273767726	0,13034176	2,10038384	0,044505649	0,007188895	0,540346557	0,007188895	0,540346557
X Variable 6	0,111107668	0,102727241	1,081579409	0,288351747	-0,098993129	0,321208466	-0,098993129	0,321208466
X Variable 7	0,157021809	0,118115601	1,329390927	0,194082819	-0,08455172	0,398595338	-0,08455172	0,398595338
X Variable 8	-1,912738248	0,088774063	-21,54613849	2,16153E-19	-2,094301592	-1,731174903	-2,094301592	-1,731174903
X Variable 9	0,486456715	0,095395103	5,099388726	1,92428E-05	0,291351822	0,681561608	0,291351822	0,681561608
X Variable 10	-0,122151305	0,095078101	-1,284746995	0,209048371	-0,316607855	0,072305246	-0,316607855	0,072305246

18

Рисунок Г.18 – Слайд 18 презентації

Результати експерименту

Таблиця 4.8 – Коефіцієнти рівняння регресії

k_0	5,505615896
k_1	0,077319531
k_2	0,336867997
k_3	0,445928093
k_4	0,269740417
k_5	0,273767726
k_6	0,111107668
k_7	0,157021809
k_8	-1,912738248
k_9	0,486456715
k_{10}	-0,122151305

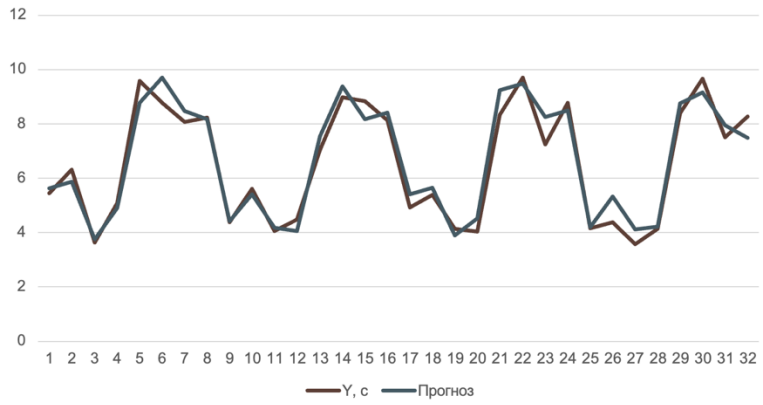
Рівняння регресії:

$$Y = 5{,}505615896 + 0{,}077319531x_1 + 0{,}336867997x_2 + 0{,}445928093x_3 + 0{,}269740417x_4 + 0{,}273767726x_5 + 0{,}111107668x_6 + 0{,}157021809x_7 - 1{,}912738248x_8 + 0{,}486456715x_9 - 0{,}122151305x_{10}$$

19

Рисунок Г.19 – Слайд 19 презентації

Результати експерименту



20

Рисунок Г.20 – Слайд 20 презентації

Аналіз отриманих результатів

Аналіз значущості змінних (P-value):

а) статистично значущі змінні (P-value < 0.05):

1) x_5 (P $=$ 0.0445);

2) x_6 (P $=$ $2.09 * 10^{-5}$):

Ці змінні мають значний вплив на залежну змінну Y.

б) змінні з високими P-value (P-value > 0.05):

1) $x_1, x_2, x_3, x_4, x_6, x_7, x_{10}$.

Їх вплив менш значущий, і їх можна розглянути для виключення з моделі.

в) змінна x_8: (P $= 2.16 * 10^{-19}$) — високий вплив, але коефіцієнт негативний (-1.9127), що свідчить про сильне зниження Y зі збільшенням x_8.

На основі наданих факторів впливу та результатів регресійного аналізу, можна інтерпретувати значення змінних і їхній вплив на залежну змінну Y.

21

Рисунок Г.21 – Слайд 21 презентації

Аналіз отриманих результатів

Ключові фактори впливу:

а) x_5 **(наявність helper-методів):**

1) коефіцієнт: 0.2737, P-value: 0.0445;

2) висновок: наявність helper-методів негативно впливає на залежну змінну; вплив значущий, тому не рекомендується використовувати helper-методи для оптимізації.

б) x_8 **(використання isolates):**

1) коефіцієнт: −1.9127, P-value: дуже низьке (<0.001);

2) висновок: використання isolates має дуже сильний позитивний вплив, що дозволяють зменшити час рендерингу;

в) x_9 **(кількість паралельних запитів):**

1) коефіцієнт: 0.48650, P-value: 0.0005.

2) висновок: збільшення кількості паралельних запитів суттєво й позитивно впливає на результат; це може означати, що паралельність оптимізує час виконання задач.

22

Рисунок Г.22 – Слайд 22 презентації

Аналіз отриманих результатів

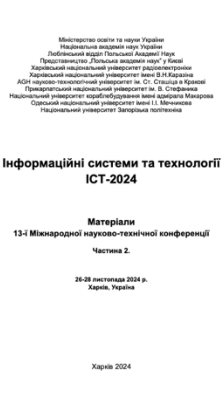
Фактори без значного впливу:

- x_1 (надмірна переробка віджетів): не має статистично значущого впливу (P=0.5520).
- x_2 (кількість константних віджетів): не значущий (P=0.1263);
- x_3 (кількість Stateful віджетів): не значущий (P=0.8579);
- x_4 (обсяг build-методів): не значущий (P=0.1386);
- x_6 (рендеринг усіх віджетів у дереві): не значущий (P=0.3210);
- x_7 (кількість віджетів із використанням Opacity): не значущий (P=0.1941);
- x_{10} (кешування картинок): не значущий (P=0.2090).

Рисунок Г.23 – Слайд 23 презентації

Публікація результатів

Тези на тему «Using Flutter Technology for Developing Cross-platform Applications»



Міністерство освіти та науки України
Національна академія наук України
Люблінський відділ Польської Академії Наук
Представництво „Польська академія наук" у Києві
Харківський національний університет радіоелектроніки
Харківський національний університет імені В.Н.Каразіна
AGH науково-технологічний університет ім. Ст. Сташіца в Кракові
Прикарпатський національний університет ім. В. Стефаника
Національний університет кораблебудування імені адмірала Макарова
Одеський національний університет імені І.І. Мечникова
Національний університет Запорізька політехніка

Інформаційні системи та технології ICT-2024

Матеріали
13-і Міжнародної науково-технічної конференції

Частина 2.

26-28 листопада 2024 р.
Харків, Україна

Харків 2024

Using Flutter Technology for Developing Cross-platform Applications

Yuliia Koba[1,†] and Oleksii Nazarov[1,*,†]

[1] *Kharkiv National University of Radio Electronics, Nauky Ave. 14, 61166 Kharkiv, Ukraine*

Abstract

The article discusses cross-platform mobile app development tools, focusing on Google's Flutter SDK. It explains that Flutter uses the Dart language to build native-feeling apps for multiple platforms, highlighting features like its like graphics engine, ahead-of-time compilation, and platform integration. The text concludes that Flutter is one of the best solutions for cross-platform development.

Keywords

cross-platform development, Dart, Flutter, software development.

1. Introduction

In the past, mobile application development for Android and iOS required the use of different programming languages: Kotlin or Java for Android, and Swift or Objective-C for iOS. Having two separate codebases became a challenge for companies developing mobile apps, leading to the rise of cross-platform development.

Cross-platform app development involves creating mobile applications that can run on multiple platforms. In this type of development, code is written only once, making it compatible with Android, iOS, or Windows. Cross-platform development has become popular due to its features and tools that developers appreciate.

2. The Evolution of Cross-Platform Development Tools

Among popular cross-platform development tools are Xamarin by Microsoft, React Native by Facebook, and PhoneGap by Adobe. Each of these development tools has its own features, advantages, and disadvantages. Developers are always looking for something new and improved to make app development efficient and cost-effective. This constant evolution of technology gave rise to Flutter.

3. Flutter Overview

The Flutter SDK (Software Development Kit) was created by Google to simplify cross-platform app development. Flutter SDK can be used to develop applications that provide a native user interface for Android, iOS, web applications, as well as Linux, Windows, and macOS. To write applications using Flutter, you will need to use the Dart programming language. Dart is a programming language also developed by Google. It is an object-oriented language that can

Information Systems and Technologies (IST-2024), November 26-28, 2024, Kharkiv, Ukraine
* Corresponding author.
† These authors contributed equally.
yuliia.koba@nure.ua (Y. Koba); oleksii.nazarov1@nure.ua (O. Nazarov)
0000-0003-1037-4041 (Y. Koba); 0000-0001-8682-5000 (O. Nazarov)
© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

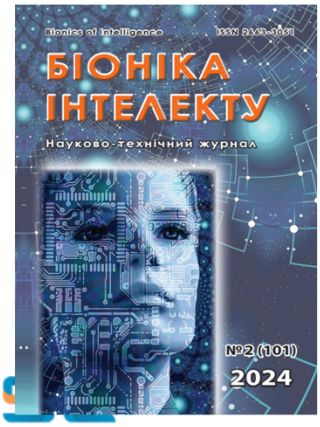
72

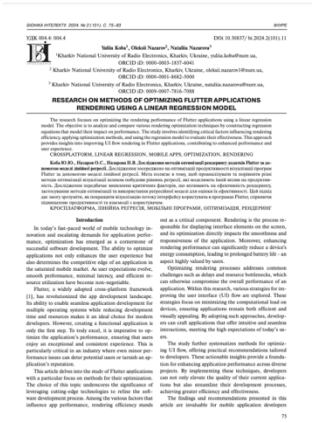
90

Рисунок Г.24 – Слайд 24 презентації

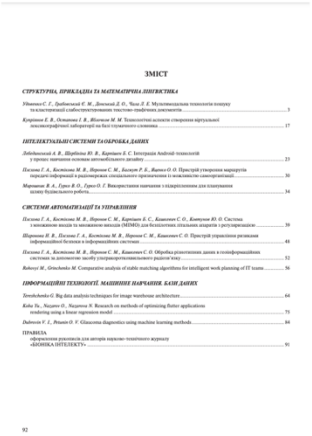
Публікація результатів

Стаття на тему «Research on methods of optimizing Flutter applications rendering using a linear regression model»





RESEARCH ON METHODS OF OPTIMIZING FLUTTER APPLICATIONS RENDERING USING A LINEAR REGRESSION MODEL



ЗМІСТ

software engineering

25

Рисунок Г.25 – Слайд 25 презентації

Підсумки

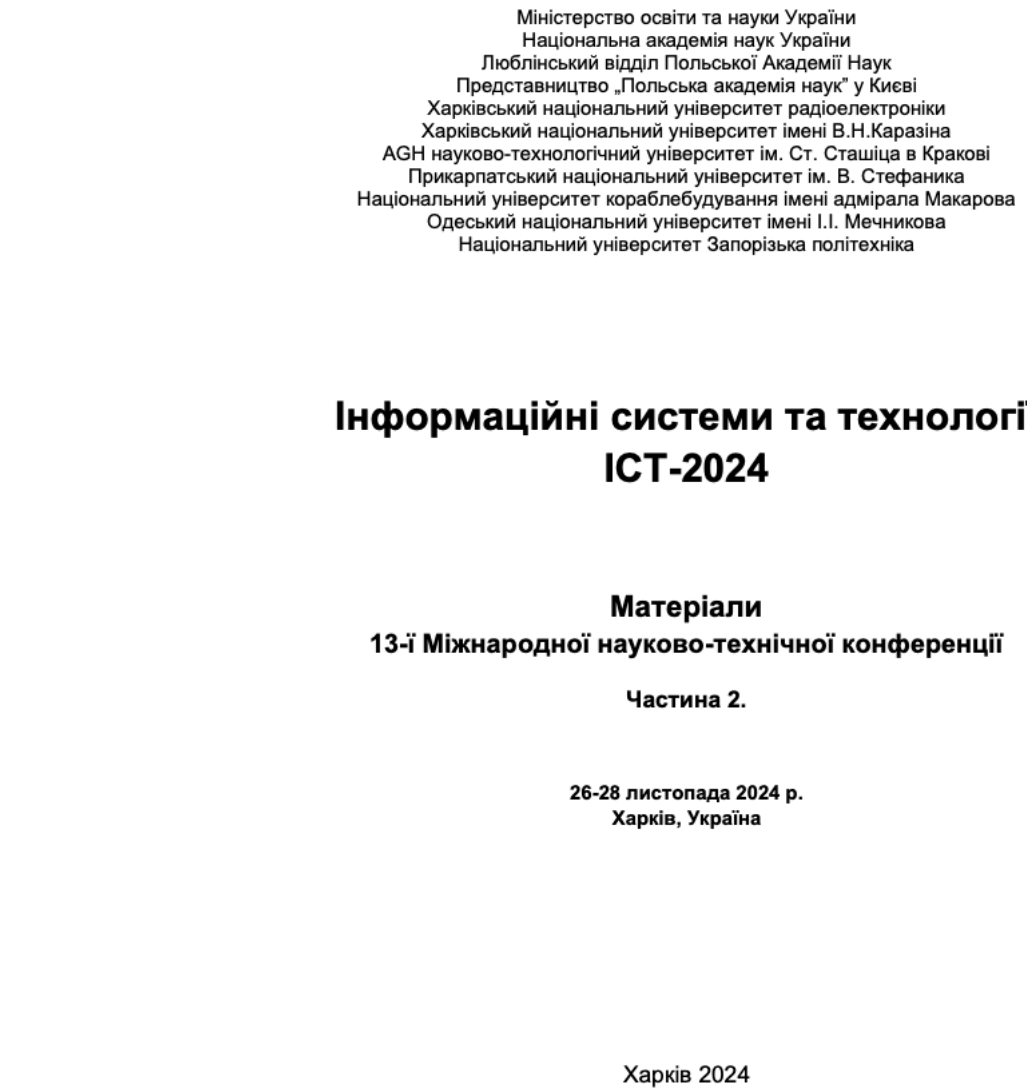
У ході дослідження було проаналізовано безсерверні Flutter-застосунки та методи їх оптимізації з використанням хмарних сервісів. Застосовано багатокритеріальний підхід для вибору оптимального рішення, обрано платформу Supabase. Розроблено архітектуру застосунку, спроектовано базу даних та створено прототип. Експериментальне дослідження включало аналіз продуктивності рендерингу інтерфейсу з використанням лінійної регресійної моделі. Результатом стало створення оптимізованого Flutter-застосунку з підвищеною продуктивністю UI, систематизацією методів оптимізації та практичними рекомендаціями для розробників мобільних додатків.

SE software engineering

26

Рисунок Г.26 – Слайд 26 презентації

ДОДАТОК Д
АПРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ. ТЕЗИ



Міністерство освіти та науки України
Національна академія наук України
Люблінський відділ Польської Академії Наук
Представництво „Польська академія наук" у Києві
Харківський національний університет радіоелектроніки
Харківський національний університет імені В.Н.Каразіна
AGH науково-технологічний університет ім. Ст. Сташіца в Кракові
Прикарпатський національний університет ім. В. Стефаника
Національний університет кораблебудування імені адмірала Макарова
Одеський національний університет імені І.І. Мечникова
Національний університет Запорізька політехніка

Інформаційні системи та технології
ІСТ-2024

Матеріали
13-ї Міжнародної науково-технічної конференції

Частина 2.

26-28 листопада 2024 р.
Харків, Україна

Харків 2024

Рисунок Д.1 – Перша сторінка збірника тез до конференції «Інформаційні системи та технології»

Ministry of education and science of Ukraine
The National Academy of Sciences of Ukraine
Polish Academy of Science, branch in Lublin
Representative office „Polish Academy of Science" in Kyiv
Kharkiv National University of Radio Electronics
V.N. Karazin Kharkiv National University
Science and Technological University AGH after St. Staszic in Krakow
V. Stefanik Precarpathian National University
Admiral Makarov National University of Shipbuilding
Odesa I.I. Mechnikov National University
National University "Zaporizhzhia Polytechnic"

Information Systems and Technologies
IST-2024

Proceedings
of the 13th International Scientific and Technical Conference

Part 2.

November 26 – 28, 2024
Kharkiv, Ukraine

Kharkiv 2024

Рисунок Д.2 – Друга сторінка збірника тез до конференції «Інформаційні системи та технології»

Using Flutter Technology for Developing Cross-platform Applications

Yuliia Koba[1,†] and Oleksii Nazarov[1,*,†]

[1] Kharkiv National University of Radio Electronics, Nauky Ave. 14, 61166 Kharkiv, Ukraine

Abstract

The article discusses cross-platform mobile app development tools, focusing on Google's Flutter SDK. It explains that Flutter uses the Dart language to build native-feeling apps for multiple platforms, highlighting features like its Skia graphics engine, ahead-of-time compilation, and platform integration. The text concludes that Flutter is one of the best solutions for cross-platform development.

Keywords

cross-platform development, Dart, Flutter, software development.

1. Introduction

In the past, mobile application development for Android and iOS required the use of different programming languages: Kotlin or Java for Android, and Swift or Objective-C for iOS. Having two separate codebases became a challenge for companies developing mobile apps, leading to the rise of cross-platform development.

Cross-platform app development involves creating mobile applications that can run on multiple platforms. In this type of development, code is written only once, making it compatible with Android, iOS, or Windows. Cross-platform development has become popular due to its features and tools that developers appreciate.

2. The Evolution of Cross-Platform Development Tools

Among popular cross-platform development tools are Xamarin by Microsoft, React Native by Facebook, and PhoneGap by Adobe. Each of these development tools has its own features, advantages, and disadvantages. Developers are always looking for something new and improved to make app development efficient and cost-effective. This constant evolution of technology gave rise to Flutter.

3. Flutter Overview

The Flutter SDK (Software Development Kit) was created by Google to simplify cross-platform app development. Flutter SDK can be used to develop applications that provide a native user interface for Android, iOS, web applications, as well as Linux, Windows, and macOS. To write applications using Flutter, you will need to use the Dart programming language. Dart is a programming language also developed by Google. It is an object-oriented language that can

---

Information Systems and Technologies (IST-2024), November 26-28, 2024, Kharkiv, Ukraine

* Corresponding author.

† These authors contributed equally.

yuliia.koba@nure.ua (Y. Koba); oleksii.nazarov1@nure.ua (O. Nazarov)

0000-0003-1837-6041 (Y. Koba); 0000-0001-8682-5000 (O. Nazarov)

Рисунок Д.3 – Перша сторінка публікації

optionally compile to JavaScript. Dart supports a wide range of programming constructs, such as interfaces, classes, collections, and generics [1].

Dart uses the Skia C++ graphics engine, which includes all protocols, compositions, and channels. Thanks to the built-in Skia engine, Flutter minimizes interaction with OS components and does not require a bridge that could slow down app performance, as is the case with React Native. Notably, with high-speed C++ at the core, Flutter achieves a high frame rate (animations at 60 or 120 frames per second), making the application feel native [2]. Flutter does not use native components in any form, so developers do not need to create bridges to communicate with them. Instead, like game engines such as Unity or Unreal (which provide highly dynamic user interfaces), Flutter draws the interface independently. Buttons, text, media elements, backgrounds—everything is rendered within the Skia graphics engine in Flutter [2].

Flutter operates by compiling code. It does not use a virtual machine or interpreted code, so it also does not require an intermediate compilation step. Instead, a Flutter application is compiled once—typically during development—and then runs on any device where it's deployed. This means that Flutter applications perform quickly, and users don't have to wait for them to start up or load [3].

Flutter provides several compatibility mechanisms, whether a developer is adding custom controls to a Flutter app or embedding Flutter into an existing application. Flutter allows developers to add custom code for mobile and desktop apps through the Platform Channel method. This is a straightforward mechanism for communication between the platform-specific code of the host application and Dart code.

Developers can send and receive messages between platform components written in languages like Swift or Kotlin and Dart by creating a shared channel. Data is serialized from Dart into a standard format and then deserialized into an equivalent representation in Kotlin or Swift [4].

4. Conclusion

Therefore, the Flutter SDK is one of the best solutions for writing cross-platform applications. The architecture of applications built with this technology has a simple structure, and the framework itself offers fast compilation and an active GitHub community. The Dart language has a clear C-like syntax, and the large number of libraries supported by other developers makes the coding process faster and simpler. As a result, this technology can be used to create a software application [5] that includes both mobile and web clients.

References

[1] Faiz M. A., Kusumo D. S., Alibasa M. J, Flutter Framework Code Portability Measurement on Multiplatform Applications, in: 2022 1st International Conference on Software Engineering and Information Technology (ICoSEIT), Bandung, 2022, pp. 36-40. doi:10.1109/ICoSEIT55604.2022.10030045.

[2] Boukhary S., Colmenares E, A Clean Approach to Flutter Development through the Flutter Clean Architecture Package, in: 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, 2019, pp. 1115– 1120. doi:10.1109/CSCI49370.2019.00211.

[3] V. Guzzi, Flutter Apprentice: Learn to Build Cross-Platform Apps, Razeware LLC, 2022.

[4] R. Rose, Flutter and Dart Cookbook: Developing Full-Stack Applications for the Cloud, O'Reilly Media, Inc., 2023. 373 p.

Рисунок Д.4 – Друга сторінка публікації

Рисунок Д.5 – Зміст збірника тез до конференції «Інформаційні системи та технології»

ДОДАТОК Е

АПРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ. СТАТТЯ

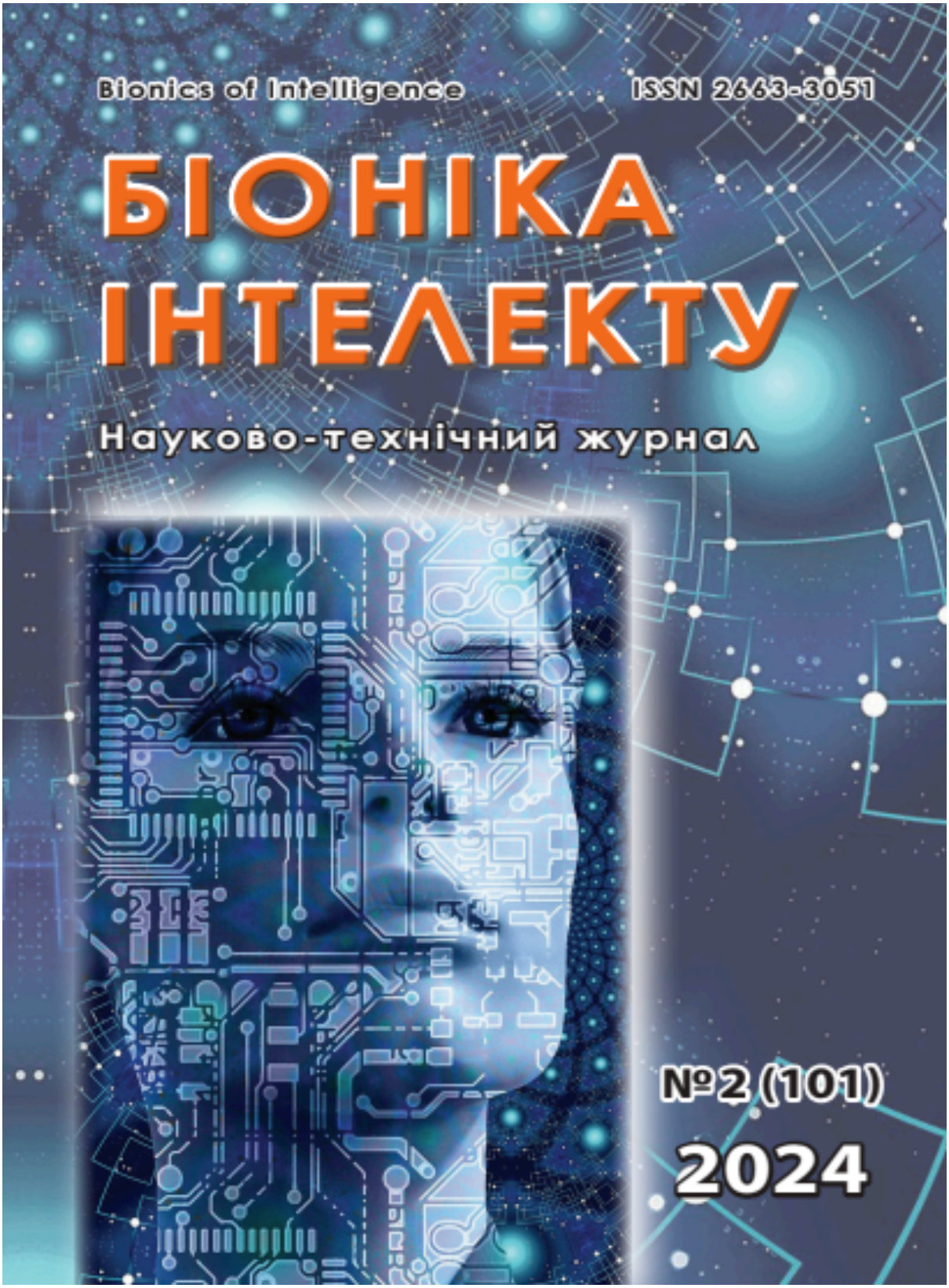


Рисунок Е.1 – Обгортка збірника

ISSN 2663-3051
(ISSN 0555-2656 до 2019 р.)

БІОНІКА ІНТЕЛЕКТУ

ІНФОРМАЦІЯ, МОВА, ІНТЕЛЕКТ

№ 2 (101) 2024

НАУКОВО-ТЕХНІЧНИЙ ЖУРНАЛ

Заснований у жовтні 1967 р.

Засновник та видавець
Харківський національний університет радіоелектроніки

Періодичність видання — *2 рази на рік*

Харків • ХНУРЕ • 2024

Рисунок Е.2 – Перша сторінка збірника

Науково-технічний журнал

«БІОНІКА ІНТЕЛЕКТУ»

ISSN 2663-3051

Заснований Харківським національним університетом
радіоелектроніки у 1967 році

Реферування та індексування:

Google Scholar

BASE

INDEX COPERNICUS
INTERNATIONAL

Національна бібліотека України
імені В. І. Вернадського

DOI prefix: 10.30837
Crossref

Журнал включено до списку наукових спеціалізованих видань України
з технічних та фізико-математичних наук
згідно з наказом Міністерства освіти і науки України № 820 від 11.07.2016
(внесено зміни згідно з наказом МОНУ № 920 від 26.06.2024)

Рисунок Е.3 – Оборот збірника

ЗМІСТ

Рисунок Е.4 – Зміст збірника

БІОНІКА ІНТЕЛЕКТУ. 2024. № 2 (101). С. 75–83 ХНУРЕ

УДК 004.4: 004.4 DOi 10.30837/ bi.2024.2(101).11

Yuliia Koba[1], Oleksii Nazarov[2], Nataliia Nazarova[3]

[1]Kharkiv National University of Radio Electronics, Kharkiv, Ukraine, yuliia.koba@nure.ua,
ORCID iD: 0000-0003-1837-6041

[2] Kharkiv National University of Radio Electronics, Kharkiv, Ukraine, oleksii.nazarov1@nure.ua,
ORCID iD: 0000-0001-8682-5000

[3] Kharkiv National University of Radio Electronics, Kharkiv, Ukraine, nataliia.nazarova@nure.ua,
ORCID iD: 0009-0007-7816-7088

RESEARCH ON METHODS OF OPTIMIZING FLUTTER APPLICATIONS RENDERING USING A LINEAR REGRESSION MODEL

The research focuses on optimizing the rendering performance of Flutter applications using a linear regression model. The objective is to analyze and compare various rendering optimization techniques by constructing regression equations that model their impact on performance. The study involves identifying critical factors influencing rendering efficiency, applying optimization methods, and using the regression model to evaluate their effectiveness. This approach provides insights into improving UI flow rendering in Flutter applications, contributing to enhanced performance and user experience.

CROSSPLATFORM, LINEAR REGRESSION, MOBILE APPS, OPTIMIZATION, RENDERING

Коба Ю.Ю., Назаров О.С., Назарова Н.В. Дослідження методів оптимізації рендерингу додатків Flutter за допомогою моделі лінійної регресії. Дослідження зосереджено на оптимізації продуктивності візуалізації програм Flutter за допомогою моделі лінійної регресії. Мета полягає в тому, щоб проаналізувати та порівняти різні методи оптимізації візуалізації шляхом побудови рівнянь регресії, які моделюють їхній вплив на продуктивність. Дослідження передбачає виявлення критичних факторів, що впливають на ефективність рендерингу, застосування методів оптимізації та використання регресійної моделі для оцінки їх ефективності. Цей підхід дає змогу зрозуміти, як покращити візуалізацію потоку інтерфейсу користувача в програмах Flutter, сприяючи підвищенню продуктивності та взаємодії з користувачем.

КРОСПЛАТФОРМА, ЛІНІЙНА РЕГРЕСІЯ, МОБІЛЬНІ ПРОГРАМИ, ОПТИМІЗАЦІЯ, РЕНДЕРИНГ

Introduction

In today's fast-paced world of mobile technology innovation and escalating demands for application performance, optimization has emerged as a cornerstone of successful software development. The ability to optimize applications not only enhances the user experience but also determines the competitive edge of an application in the saturated mobile market. As user expectations evolve, smooth performance, minimal latency, and efficient resource utilization have become non-negotiable.

Flutter, a widely adopted cross-platform framework [1], has revolutionized the app development landscape. Its ability to enable seamless application development for multiple operating systems while reducing development time and resources makes it an ideal choice for modern developers. However, creating a functional application is only the first step. To truly excel, it is imperative to optimize the application's performance, ensuring that users enjoy an exceptional and consistent experience. This is particularly critical in an industry where even minor performance issues can deter potential users or tarnish an application's reputation.

This article delves into the study of Flutter applications with a particular focus on methods for their optimization. The choice of this topic underscores the significance of leveraging cutting-edge technologies to refine the software development process. Among the various factors that influence app performance, rendering efficiency stands out as a critical component. Rendering is the process responsible for displaying interface elements on the screen, and its optimization directly impacts the smoothness and responsiveness of the application. Moreover, enhancing rendering performance can significantly reduce a device's energy consumption, leading to prolonged battery life - an aspect highly valued by users.

Optimizing rendering processes addresses common challenges such as delays and resource bottlenecks, which can otherwise compromise the overall performance of an application. Within this research, various strategies for improving the user interface (UI) flow are explored. These strategies focus on minimizing the computational load on devices, ensuring applications remain both efficient and visually appealing. By adopting such approaches, developers can craft applications that offer intuitive and seamless interactions, meeting the high expectations of today's users.

The study further systematizes methods for optimizing UI flow, offering practical recommendations tailored to developers. These actionable insights provide a foundation for enhancing application performance across diverse projects. By implementing these techniques, developers can not only elevate the quality of their current applications but also streamline their development processes, achieving greater efficiency and effectiveness.

The findings and recommendations presented in this article are invaluable for mobile application developers

Рисунок Е.5 – Перша сторінка статті

striving to optimize their projects. They highlight the importance of balancing performance with user-centric design, paving the way for software that not only meets but exceeds industry standards. Ultimately, this research contributes to the broader field of mobile software development, presenting new avenues for innovation and excellence.

1. Why is performance important?

Performance in Flutter apps is crucial for several reasons, as it directly impacts user experience, app adoption, and long-term success.

Performance is the backbone of a great user experience, and Flutter apps are no exception. Users today are accustomed to fast and fluid interactions in mobile apps, and any deviation can lead to dissatisfaction [2]:

– instant feedback: when users tap a button or scroll through a list, they expect immediate feedback. A lag of even a few milliseconds can make the app feel unresponsive;

– smooth scrolling and transitions: apps with janky scrolling or choppy animations create a sense of poor quality. Flutter is designed for fluid 60 FPS animations, but without optimization, heavy UI elements or inefficient code can disrupt this;

– perceived quality: high performance is often subconsciously associated with professionalism and trustworthiness. A smooth app feels polished and reliable, while a slow app can erode user confidence.

– Retaining users is just as important as acquiring them, and performance plays a critical role in this [3]:

– avoiding frustration: studies show that even slight performance issues can lead to users abandoning an app. For instance, a 1-second delay in response time can decrease customer satisfaction by up to 16%;

– positive feedback loop: users who enjoy a fast and smooth app are more likely to leave positive reviews, recommend the app to others, and return for repeated usage;

– gamified and real-time features: Flutter apps often include interactive or real-time features like leaderboards, chat systems, or live updates. These require robust performance to maintain engagement.

– The mobile app market is saturated, and competition is fierce. Performance optimization can be a key differentiator [4]:

– standing out from the crowd: with thousands of apps vying for user attention, those that offer superior performance are more likely to be noticed and retained;

– App Store rankings: performance directly impacts app ratings and reviews, which are critical for app store visibility. Apps with poor performance often face negative reviews and lower rankings, making them harder to discover.

Flutter is designed to create apps that work across a wide range of devices and operating systems. Ensuring good performance means your app remains accessible to everyone, regardless of their hardware [5]:

– support for low-end devices: not every user has access to high-performance smartphones. Optimizing performance ensures that users on older or less powerful devices still get a good experience;

– global reach: in many regions, low-end devices dominate the market. A poorly optimized app could alienate a significant portion of potential users.

– High-performance apps often correlate directly with better business outcomes:

– increased conversion rates: for e-commerce apps, performance issues can lead to cart abandonment. A smooth checkout process ensures users complete their transactions;

– improved retention metrics: retained users are more likely to make in-app purchases, subscribe to premium features, or engage with ads, driving higher revenue;

– lower cost of acquisition: satisfied users are more likely to recommend the app, reducing the need for expensive user acquisition campaigns.

Performance isn't just about speed - it's also about efficiency [6]:

– battery life: poorly optimized apps drain battery life, frustrating users. Flutter developers must ensure efficient use of resources like CPU and GPU to preserve device power;

– memory usage: apps that consume excessive memory can slow down the entire device or lead to crashes. Efficient memory management is essential for a smooth user experience;

– data efficiency: apps that minimize unnecessary network requests and efficiently compress or cache data provide a better experience for users with limited data plans.

– As your app grows, its performance needs to scale with it:

– handling more users: apps that perform well under stress - such as during a sudden influx of traffic - are more likely to succeed. Poorly optimized apps may crash or slow down during high usage;

– adding features: a well-optimized codebase makes it easier to add new features without significantly impacting performance. Flutter's modular architecture supports this, but developers must implement best practices to maintain scalability.

Flutter's unique architecture offers many benefits, but it also requires specific attention to performance [7]:

– widget hierarchies: Flutter's declarative approach relies heavily on widgets. Deep or overly complex widget trees can slow down rendering. Developers need to optimize widget structures and use tools like the Flutter DevTools profiler;

– frame budget: Flutter aims for 60 FPS (or 120 FPS on devices with high refresh rates), meaning each frame must be rendered in under 16 milliseconds. Exceeding this budget leads to dropped frames and visible lag;

Рисунок Е.6 – Друга сторінка статті

– Dart performance: Flutter uses Dart, which is fast but requires careful management of asynchronous tasks, memory allocation, and heavy computations to avoid blocking the UI thread.

Building high-performance apps also reduces long-term maintenance and operational costs [8]:

– fewer bugs: optimized apps are often more stable, leading to fewer user complaints and less time spent on bug fixes;

– reduced technical debt: addressing performance early prevents the accumulation of inefficient code that becomes harder to fix later;

– infrastructure costs: efficient apps reduce server load, bandwidth usage, and other infrastructure costs, especially important for apps with large-scale operations.

Lastly, performance isn't just about technical metrics - it's about delighting users.

– micro-interactions: small details like button animations, loading indicators, and page transitions can make an app feel alive. Performance ensures these elements flow seamlessly;

– flow state: apps that perform well create a sense of flow, where users remain engaged without being distracted by lag or glitches.

Performance in Flutter apps isn't just a technical concern — it's a fundamental aspect of delivering value to users, growing your audience, and succeeding in a competitive market. By prioritizing performance, developers can ensure their Flutter apps stand out, delight users, and drive long-term business success.

2. Productivity factors and methods of their optimization

In the context of Flutter applications, performance is primarily focused on two key indicators [9]:

– rendering speed – the speed at which Flutter can generate the pixels that make up the application interface on the screen. Ideally, Flutter should render each frame in approximately 16 milliseconds (ms) to achieve smooth playback at 60 frames per second (FPS). This ensures a seamless and responsive user interaction;

– frames per second (FPS) – FPS indicates the number of times per second the application interface is updated and redrawn on the screen. A higher frame rate leads to a smoother and more fluid user experience. Conversely, a low frame rate can cause jerks, delays, and a sense of sluggishness.

Ensuring optimal rendering speed and high frame rate is critically important for achieving high performance and user satisfaction in Flutter applications.

Several factors can influence the performance of a Flutter application. Here are the main ones:

– widget tree complexity: Flutter builds the application interface using a widget hierarchy. A complex widget tree with many nested elements may require more time to render, which will impact performance;

– widget reconstruction frequency: Flutter rebuilds the entire widget subtree every time there is a change in its state, even if the change affects only a small part of the interface. This can become a performance bottleneck for frequently updated widgets or those deeply nested in the widget tree;

– state management strategy: How the application state is managed can significantly impact performance. Improper state management practices can cause unnecessary widget rebuilds, leading to slowdowns;;

– interface complexity: Visually complex interfaces with rich animations, heavy layouts, or large images may require more computational resources for rendering, potentially affecting performance;;

– device capabilities: Application performance will also depend on the user's device. Devices with low computational power, limited memory, or slow network connections will experience application slowdowns.

Considering these factors, it is important to carefully optimize a Flutter application to ensure the best performance and user experience. For the research, the following optimization methods can be highlighted:

– avoiding unnecessary widget reconstruction;
– using constant constructors;
– minimizing the usage of Stateful widgets;
– minimizing the length of build methods;
– minimizing the usage of helper methods;;
– rendering only widgets that are visible on the current screen;
– minimizing the use of opacity in widgets;
– efficient usage of asynchronous functions and multithreading;
– optimizing network requests;
– data caching.

3. Selection of a linear model for evaluation of optimization methods

To conduct a performance study of optimization methods, a decision was made to build a mathematical experiment model in the form of a linear model without factor dependencies for several reasons [10], substantiated by the specifics of rendering optimization methods research in serverless Flutter applications:

– independence of optimization methods: The primary reason for choosing a linear model is that each UI layer rendering optimization method is applied separately, and their effectiveness does not depend on each other. This allows using a simple linear model where each factor (optimization method) has its own impact on the result (rendering time) without creating interdependencies between them. Thus, changing one method will not directly affect the results of others, which allows building a model without considering complex interactions;

– simplification of experiment complexity: The linear model is one of the simplest mathematical models that effectively evaluates individual factor influences without the

Рисунок Е.7 – Третя сторінка статті

need to complicate the model with interdependent variables. Since the research focuses on comparing the effectiveness of various optimization methods, the simplicity of the linear model maintains analysis transparency and reduces the possibility of errors in result interpretation;

– measurement and comparison capability: For each optimization method, the UI layer rendering time will be measured in two scenarios: with and without optimization techniques. The linear model allows for a clear comparison of these two variants for each method, evaluating which method specifically impacts performance improvement. Each method can be considered as a separate factor, the impact of which is measured independently of other methods;

– convenience for results analysis: Linear regression allows for a clear assessment of each optimization method's contribution to the overall result. This provides an opportunity to evaluate not only the total rendering time but also quantitatively determine how much each method affects the application's performance. This approach yields specific and intuitively understandable results that are convenient for further analysis and decision-making regarding the selection of the most effective methods;

– minimizing the influence of random variables: A linear model without dependencies between factors allows minimizing the impact of random variables and data noise. Since each method is evaluated separately, its effectiveness can be measured more accurately without distorting the results through method interactions, ensuring high experimental reliability.

Given the aforementioned factors, the linear model is an optimal choice for researching rendering optimization method effectiveness, as it provides accuracy, simplicity, and analysis convenience while minimizing experiment complexity.

4. Software development for conducting research

For the research, a page was created that would display a list of items. The study will be conducted in this environment, as lists are one of the most used ways of displaying information in applications. Below is the code for each of the optimization aspects defined above.

The aspect of "avoiding unnecessary widget re-rendering" involves avoiding additional calls to setState methods and ViewModel updates. In this case, we will consider using the ignoreChange method (Fig. 1), which will block re-rendering the page when it is not needed.

```
    return StoreConnector(
      distinct: true,
      converter: _ViewModel.new,
      ignoreChange: _ViewModel.ignoreChange,
_
class _ViewModel extends TableViewModel<Partner, GeneralTablePointer> {
  _ViewModel(super.store);

  static bool ignoreChange(AppState state) =>
      state.tablesState.getTable<Partner, GeneralTablePointer>().isLoading &&
      state.tablesState.getTable<Partner, GeneralTablePointer>()
          .items.isNotEmpty;
}
```

Fig. 1. Avoiding unnecessary widget re-rendering code

Constant constructors allow you to create immutable widgets. This allows Flutter to reuse them in memory more efficiently, which reduces the cost of rendering and object creation. The result is a reduced memory footprint and improved performance. Below is the code (Fig. 2) with and without constant widgets.

```
class PartnersPage extends StatelessWidget {
  const PartnersPage( { super.key });

  @override Widget build(BuildContext context) {
    return Scaffold(
      body: const SafeArea(
        child: Column(
          children: [
            PartnersTableActionBar(),
            Expanded(child: PartnersTable()),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        child: const Icon(Icons.add),
        onPressed: () => StoreProvider.of<AppState>(context).dispatch(
          OpenPageAction(Destination.createPartner),
        ),
      ),
    );
  }
}
class PartnersPage extends StatelessWidget {
  const PartnersPage( { super.key });

  @override Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: Column(
          children: [
            PartnersTableActionBar(),
            Expanded(child: PartnersTable()),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        child: Icon(Icons.add),
        onPressed: () => StoreProvider.of<AppState>(context).dispatch(
          OpenPageAction(Destination.createPartner),
        ),
      ),
    );
  }
}
```

Fig. 2. Code with and without constant widgets

Stateful widgets are more expensive than Stateless widgets because they have state that needs to be stored and updated. Too many of these widgets can slow down your application. Therefore, below (Fig. 3) are two cases of rendering the same components using Stateful or Stateless widgets.

The build methods are executed every time the widget is redrawn. If the method is large and complex, it can cause delays in the interface. The following is code using the long and short build methods (Fig. 4).

Helper methods inside build often create new objects on each call, which impacts performance. Below is the code using list building with methods and individual widgets (Fig. 5).

Rendering elements that are not visible to the user consumes device resources without any benefit. Therefore, it is better to use ListView (or SliverList) than SingleChildScrollView. The code for using both is given below (Fig. 6).

Рисунок Е.8 – Четверта сторінка статті

```
class PartnersTableActionBar extends StatefulWidget {
  const PartnersTableActionBar( { super.key });

  @override StatefulPartnersTableActionBarState createState() =>
      _PartnersTableActionBarState();
}

class _PartnersTableActionBarState extends State<PartnersTableActionBar> {
  @override Widget build(BuildContext context) {
    return StoreConnector(
      distinct: true,
      converter: TableViewModel<Partner, GeneralTablePointer>.new,
      builder: (context, viewModel) => Padding(
        padding: const EdgeInsets.all(8.0),
        child: Row(
          children: [
            Expanded(
              child: SearchField(
                hintText: context.strings.searchPartners,
                isLoading: viewModel.isLoading,
                search: viewModel.search,
              ),
            ),
            const SizedBox(width: 16),
            FilterButton(
              isEmpty: viewModel.filter.isFilterByEmpty,
              onPressed: () {
                //TODO: Open filters page
              },
            ),
          ],
        ),
      ),
    );
  }
}

class PartnersTableActionBar extends StatelessWidget {
  const PartnersTableActionBar( { super.key });

  @override Widget build(BuildContext context) {
    return StoreConnector(
      distinct: true,
      converter: TableViewModel<Partner, GeneralTablePointer>.new,
      builder: (context, viewModel) => Padding(
        padding: const EdgeInsets.all(8.0),
        child: Row(
          children: [
            Expanded(
              child: SearchField(
                hintText: context.strings.searchPartners,
                isLoading: viewModel.isLoading,
                search: viewModel.search,
              ),
            ),
            const SizedBox(width: 16),
            FilterButton(
              isEmpty: viewModel.filter.isFilterByEmpty,
              onPressed: () {
                //TODO: Open filters page
              },
            ),
          ],
        ),
      ),
    );
  }
}
```

Fig. 3. Two cases of rendering the same components using Stateful or Stateless widgets

The Opacity widget adds a rendering layer, which increases the load on the GPU. Using alternatives like Colors.transparent or style management is more efficient. The Visibility widget also relies on the Opacity widget. Below is the code using this widget and an alternative without it (Fig. 7).

Asynchrony and isolates allow you to perform resource-intensive tasks (data loading, calculations) outside the main thread responsible for rendering the UI. Below is a class that defines the dominant color of an image list with and without the use of isolates (Fig. 8).

Improper request handling, such as redundant or frequent calls, can overload the network and slow down the application, and some independent calls can be made simultaneously. Below is how to use sequential and parallel requests to the server (Fig. 9).

Caching reduces the number of recalculations and data downloads from the network or database. Below is an implementation of displaying avatars using a regular widget and a widget that supports data caching (Fig. 10).

```
class PartnersPage extends StatelessWidget {
  const PartnersPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: Column(
          children: [
            Expanded(
              child: StoreConnector(
                distinct: true,
                converter: _ViewModel.new,
                ignoreChange: _ViewModel.ignoreChange,
                builder: (context, viewModel) {
                  return LoadMoreScrollListener(
                    loadMore: viewModel.downloadItems,
                    child: TableRefreshIndicator<Partner, GeneralTablePointer>(
                      builder: (context, iosRefreshIndicator) =>
                          CustomScrollView(
                        slivers: [
                          iosRefreshIndicator,
                          if (viewModel.items.isEmpty)
                            SliverFillRemaining(
                              hasScrollBody: false,
                              child: EmptyTablePlaceholder(
                                isLoading: viewModel.isLoading,
                                title: Text(context.strings.noPartnersFound),
                                subtitle: Text(
                                  viewModel.filter.isEmpty
                                      ? context.strings.addYourFirstPartner
                                      : context.strings.changeYourSearchQuery,
                                ),
                                icon: Icon(
                                    HomePageTabType.partners.activeIconData),
                              ),
                            )
                          else
                            SliverList(
                              delegate: SliverChildBuilderDelegate(
                                (context, index) => PartnerCard(
                                  key: ValueKey(viewModel.items[index].id),
                                  onPressed: () {},
                                  partner: viewModel.items[index],
                                ),
                                childCount: viewModel.items.length,
                              ),
                            ),
                        ],
                      ),
                    ),
                  );
                }
              ),
            ),
          ],
        ),
      ),
    );
  }
}

class PartnersPage extends StatelessWidget {
  const PartnersPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: const SafeArea(
        child: Column(
          children: [
            Expanded(child: PartnersTable()),
          ],
        ),
      ),
    );
  }
}
```

Fig. 4. Code using the long and short build methods

Рисунок Е.9 – П’ята сторінка статті

```
SliverList(
                    delegate: SliverChildBuilderDelegate(
                      (context, index) => PartnerCard(
                        key: ValueKey(viewModel.items[index].id),
                        onPressed: () {},
                        partner: viewModel.items[index],
                      ),
                      childCount: viewModel.items.length,
                    ),
                  )
-
  SliverList(
                    delegate: SliverChildBuilderDelegate(
                      (context, index) => _buildPartner(
                        context,
                        viewModel.items[index],
                      ),
                      childCount: viewModel.items.length,
                    ),
                  ),
              ],
            ),
          ),
        );
},
    );
}

Widget _buildPartner(BuildContext context, Partner partner) {
  return ListTile(
      onTap: () {},
      leading: CircleIconPreview.user(
        imageUrl: partner.avatarUrl,
        radius: 28,
      ),
      title: RichText(
        text: TextSpan(
          text: partner.name,
          style: Theme.of(context).textTheme.bodyLarge,
          children: [
            TextSpan(
              text: ' (${partner.type.getDisplayText(context)})',
              style: Theme.of(context).textTheme.bodySmall,
            ),
          ],
        ),
      ),
      subtitle: RatingView(rate: partner.averageMark ?? 0),
    );
}
```

Fig. 5. Code using list building with methods and individual widgets

```
SliverList(
                    delegate: SliverChildBuilderDelegate(
                      (context, index) => PartnerCard(
                        key: ValueKey(viewModel.items[index].id),
                        onPressed: () {},
                        partner: viewModel.items[index],
                      ),
                      childCount: viewModel.items.length,
                    ),
                  )
-
SingleChildScrollView(
                  child: Column(
                    children: viewModel.items
                        .map((item) => PartnerCard(
                              key: ValueKey(item.id),
                              onPressed: () {},
                              partner: item,
                            ))
                        .toList(),
                  ),
                )
```

Fig. 6. Code using SliverList and SingleChildScrollView

Following these recommendations allows you to create fast, stable, and energy-efficient applications that provide a positive user experience.

```
Visibility(
      visible: isLoading,
      replacement: Row(
        mainAxisAlignment: MainAxisAlignment.center,
        mainAxisSize: MainAxisSize.min,
        children: [
          IconTheme(
            data: IconTheme.of(context).copyWith(color: color, size: 32),
            child: icon,
          ),
          const SizedBox(width: 10),
          Flexible(
            child: DefaultTextStyle(
                  style: Theme.of(context).textTheme.bodyLarge!.copyWith(
                        color: color,
                      ),
                  child: title,
                ),
          ),
        ],
      ),
      child: const StyledLoader.primary(size: 32),
-
  return Container(
    alignment: Alignment.center,
    padding: const EdgeInsets.symmetric(vertical: 16, horizontal: 32),
    child: isLoading
        ? const StyledLoader.primary(size: 32)
        : Row(
            mainAxisAlignment: MainAxisAlignment.center,
            mainAxisSize: MainAxisSize.min,
            children: [
              IconTheme(
                data: IconTheme.of(context).copyWith(color: color, size: 32),
                child: icon,
              ),
              const SizedBox(width: 10),
              Flexible(
                child: DefaultTextStyle(
                  style: Theme.of(context).textTheme.bodyLarge!.copyWith(
                        color: color,
                      ),
                  child: title,
                )
              ),
            ],
          ),
  );
```

Fig. 7. Code with and without Opacity

```
Future<List<Color>> getAverageColors(List<SimpleImageProvider> images) async {
  final bytes = await Future.wait(images.map((item) => item.toBytes()));

  return compute((bytes) {
    final colors = bytes.map(_getAverageColor).toList();
    return colors;
  }, bytes, );
}
- Future<List<Color>> getAverageColors(List<SimpleImageProvider> images) async {
  final bytes = await Future.wait(images.map((item) => item.toBytes()));

  return bytes.map(_getAverageColor).toList();
}
```

Fig. 8. Code with and without isolates usage

```
List<Partner> partners = [];

final values =
    result.map((item) => PartnerDto.fromJson(item).toDomain()).toSet();

for (final value in values) {
  partners.add((await getPartnerDetails(value.id)).result!);
}
-
final values =
    result.map((item) => PartnerDto.fromJson(item).toDomain()).toSet();

final partners =
    await Future.wait(values.map((item) => getPartnerDetails(item.id)));
```

Fig. 9. Sequential and parallel requests

```
return Image.network(
    widget.imageUrl,
    placeholder: (context, url) => const CircularProgressIndicator(),
    errorWidget: (context, url, error) => const Icon(Icons.error), );

return CachedNetworkImage(
    imageUrl: widget.imageUrl,
    placeholder: (context, url) => CircularProgressIndicator(),
    errorWidget: (context, url, error) => Icon(Icons.error), );
```

Fig. 10. Regular widget and a widget that supports data caching

Рисунок Е.10 – Шоста сторінка статті

5. Conducting an experimental study of selected rendering optimization methods

5.1. Stage 1. A priori information analysis.

Let the following become known during the analysis of a priori information about the software:

– the software is a mobile application written in the Dart programming language using the Flutter framework;

– the software runs on a mobile device running the Android or iOS operating system;

– the software uses Supabase as a server..

We will use the mathematical model of the experiment in the form of a linear model without dependencies between factors. The task is reduced to finding the values of the coefficients ki of the regression equation.

$$y = k_0 + k_1 x_1 + \ldots + k_n x_n$$

The factors with the largest values will have the greatest influence on the output characteristic.

5.2. Stage 2. Selection of influencing factors.

The impact factors are selected as a result of the analysis of a priori information about the software (Table 1)

Table 1

Factors of influence

Factor	Description
x_1	Excessive widget processing
x_2	Number of constant widgets
x_3	Number of Stateful widgets
x_4	Amount of build methods (in rows)
x_5	Availability of helper methods
x_6	Rendering of all widgets in the tree
x_7	Number of widgets using Opacity
x_8	Using isolates
x_9	Number of parallel requests
x_{10}	Caching pictures

5.3. Stage 3. Selection of upper and lower levels for factors.

We select the upper and lower levels for each factor (Table 2).

Table 2

Upper and lower values of influence factors.

Factor	Upper value	Lower value
x_1	on	off
x_2	20	4
x_3	20	2
x_4	120	30
x_5	on	off
x_6	on	off
x_7	20	0
x_8	on	off
x_9	20	0
x_{10}	on	off

5.4. Stage 4. Compilation of the matrix of planning and conducting experiments.

Y is the time of loading and rendering of the table. The matrix and the results of the experiment are shown in Fig 11.

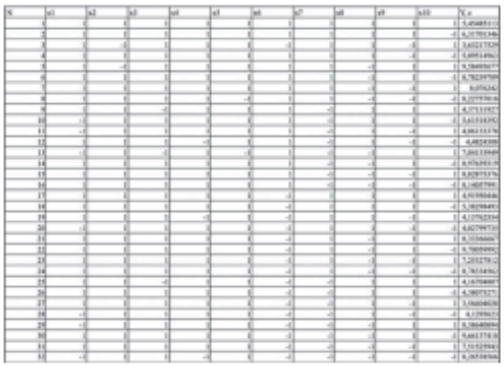


Fig. 11. Matrix of experiments and results

5.5. Stage 5. Analysis of results.

Using data analysis in Microsoft Excel, we will perform a regression analysis (Fig. 12).

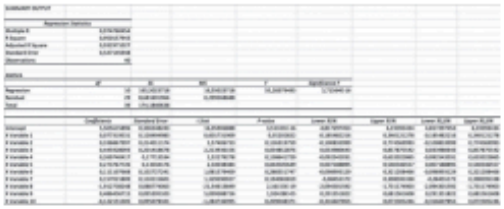


Fig. 12. Regression analysis of research results

The main conclusions:

– Multiple R: 0.9748 — strong correlation between independent variables and dependent variable;

– R Square: 0.9502 — model explains 95.02% of the variation of the dependent variable;

– Adjusted R Square: 0.9329 — the adjusted coefficient takes into account the number of variables and the sample size;

– Significance F: $3.715*10^{-16}$ (very low value) — the model is statistically significant.

The coefficients of the regression equation can be found as:

Рисунок Е.11 – Сьома сторінка статті

$$k_j = \frac{\sum_{i=1}^{N} x_{ij} y_i}{N}, j = 1k$$

Table 3

Coefficients of the regression equation

k_0	5,505615896
k_1	0,077319531
k_2	0,336867997
k_3	0,445928093
k_4	0,269740417
k_5	0,273767726
k_6	0,111107668
k_7	0,157021809
k_8	-1,912738248
k_9	0,486456715
k_{10}	-0,122151305

Regression equation:

$$Y = 5{,}505615896 + 0{,}077319531x_1 + 0{,}336867997x_2 + 0{,}445928093x_3 + 0{,}269740417x_4 + 0{,}273767726x_5 + 0{,}111107668x_6 + 0{,}157021809x_7 - 1{,}912738248x_8 + 0{,}486456715x_9 - 0{,}122151305x_{10}$$

Visualization of forecasting results is shown in Fig. 13.

Fig. 13. Prediction results

Analysis of the significance of variables (P-value):

а) statistically significant variables (P-value < 0.05):

1) x_5 ($P = 0.0445$);

2) x_8 ($P = 2.09*10^{-5}$):

These variables have a significant effect on the dependent variable Y.

б) ariables with high P-value (P-value > 0.05):

1) $x_1, x_2, x_3, x_4, x_6, x_7, x_{10}$.

Their influence is less significant and can be considered for exclusion from the model.

в) variable x_8: ($P = 2.16*10^{-19}$) — a high influence, but the coefficient is negative (-1.9127), which indicates a strong decrease in Y with an increase in x_8.

Based on the given influencing factors and the results of the regression analysis, it is possible to interpret the values of the variables and their effect on the dependent variable Y. Below is a more detailed analysis.

Key influencing factors:

а) x_5 (availability of helper methods):

1) coefficient: 0.2737, P-value: 0.0445;

2) conclusion: the presence of helper methods negatively affects the dependent variable; the impact is significant, so it is not recommended to use helper methods for optimization.

б) x_8 (using isolates):

1) coefficient: −1.9127, P-value: very low (<0.001);

2) conclusion: the use of isolates has a very strong positive effect, allowing to reduce rendering time;

в) x_9 (number of parallel requests):

1) coefficient: 0.48650, P-value: 0.0005.

2) conclusion: an increase in the number of parallel requests significantly and positively affects the result; this may mean that parallelism optimizes the execution time of tasks.

Factors without significant influence:

x_1 (excessive widget recycling): no statistically significant effect (P=0.5520);

x_2 (number of constant widgets): not significant (P=0.1263);

x_3 (number of Stateful widgets): not significant (P=0.8579);

x_4 (length of build methods): not significant (P=0.1386);

x_6 (rendering of all widgets in a tree): not significant (P=0.3210);

x_7 (number of widgets using Opacity): not significant (P=0.1941);

x_{10} (image caching): not significant (P=0.2090).

Conclusion

In the competitive landscape of modern mobile technology, the optimization of Flutter applications is essential for delivering an exceptional user experience. This study underscores the importance of rendering optimization, highlighting its critical role in achieving smooth and responsive user interfaces while minimizing energy consumption. By systematically exploring various optimization strategies, the research provides actionable insights for developers aiming to enhance the performance of their applications.

The findings reveal that certain practices significantly influence the efficiency of Flutter applications. Notably, the use of helper methods negatively impacts performance and is not recommended as an optimization strategy due to its adverse effects on rendering time. On the other hand, employing isolates demonstrates a remarkably strong positive effect, enabling a significant reduction in rendering time by offloading computationally intensive tasks to separate threads. Additionally, increasing the number of parallel requests emerges as a powerful technique, as it enhances task execution efficiency and optimizes overall application performance.

These insights emphasize the importance of thoughtful and informed optimization practices. Developers are

Рисунок Е.12 – Восьма сторінка статті

encouraged to prioritize strategies like isolates and parallelism while avoiding techniques that may inadvertently hinder performance. By implementing these recommendations, developers can create Flutter applications that are not only visually appealing but also efficient and resource-friendly, catering to the high expectations of modern users.

In conclusion, this research provides a robust framework for optimizing Flutter applications, paving the way for superior software development. By adopting these practices, developers can improve both the performance and the user experience of their applications, ensuring long-term success in the dynamic mobile technology market.

References

[1] *Bhagat, S.* (2022). Review on Mobile Application Development Based on Flutter Platform. *International Journal for Research in Applied Science and Engineering Technology*. https://doi.org/10.22214/ijraset.2022.39920.

[2] *Bia*ł*kowski, D., & Smolka, J.* (2022). Evaluation of Flutter framework time efficiency in context of user interface tasks. *Journal of Computer Sciences Institute*. https://doi.org/10.35784/jcsi.3007.

[3] *Zuniga, A., Flores, H., Lagerspetz, E., Nurmi, P., Tarkoma, S., Hui, P., & Manner, J.* (2019). Tortoise or Hare? Quantifying the Effects of Performance on Mobile App Retention. *The World Wide Web Conference*. https://doi.org/10.1145/3308558.3313428.

[4] *Hort, M., Kechagia, M., Sarro, F., & Harman, M.* (2021). A Survey of Performance Optimization for Mobile Applications. *IEEE Transactions on Software Engineering*, 48, 2879-2904. https://doi.org/10.1109/TSE.2021.3071193.

[5] *Biørn-Hansen, A., Grønli, T., & Ghinea, G.* (2019). Animations in Cross-Platform Mobile Applications: An Evaluation of Tools, Metrics and Performance. *Sensors (Basel, Switzerland)*, 19. https://doi.org/10.3390/s19092081.

[6] *Nanavati, J., Patel, S., Patel, U., & Patel, A.* (2024). Critical Review and Fine-Tuning Performance of Flutter Applications. *2024 5th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI)*, 838-841. https://doi.org/10.1109/ICMCSI61536.2024.00131.

[7] *McKelvie, K., & Kanapesky, A.* (2022). Architectural improvements to increase reverberation and reduce flutter echo in two music rehearsal rooms. *The Journal of the Acoustical Society of America*. https://doi.org/10.1121/10.0015412.

[8] *Lovrić, L., Fischer, M., Röderer, N., & Wünsch, A.* (2023). Evaluation of the Cross-Platform Framework Flutter Using the Example of a Cancer Counselling App. , 135-142. https://doi.org/10.5220/0011824500003476.

[9] *Piskor, J., & Badurowicz, M.* (2023). Performance comparison of Flutter platform GUI in web and native environments. *Journal of Computer Sciences Institute*. https://doi.org/10.35784/jcsi.3677.

[10] *Williams, B.* (2020). Identification of the linear factor model. *Econometric Reviews*, 39, 109 - 92. https://doi.org/10.1080/07474938.2018.1550042.

The article was delivered to editorial stuff on the 14.11.2024

Рисунок Е.13 – Дев’ята сторінка статті

ДОДАТОК Ж

ЕКСПЕРНИЙ ВИСНОВОК РЕЗУЛЬТАТІВ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА ВІДПОВІДНІСТЬ ОФОРМЛЕННЯ ВИМОГАМ ДСТУ 3008:2015

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-23-1 (група)

Коба Юлія Юріївна
(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.4 Нумерація розділів, підрозділів, пунктів, підпунктів	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні позначки	
	7.15 Додатки	

зауважень немає

Експерт _____ (підпис) Олена ОЛІЙНИК (прізвище, ініціали)

02.06.2025

Рисунок Ж.1 – Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015