УДК 004.415.25 + 004.42 + 004.514

ДИНАМИЧЕСКАЯ ГЕНЕРАЦИЯ ВИЗУАЛЬНЫХ ИНТЕРФЕЙСОВ FLUTTER-ПРИЛОЖЕНИЙ

А. Ю. Усачев

Высшая школа информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета

usacheow.ar@gmail.com

Аннотация

На этапе разработки проектов команды разработки, помимо прочего, сталкиваются с потребностью в проектировании визуального интерфейса, который удовлетворит все группы пользователей, и распространении новой версии продукта на всех пользователей. В статье предложена концепция создания инструмента для генерации динамических экранов, что позволит оптимизировать процессы адаптации интерфейсов приложений и выпуска обновлений.

Ключевые слова*: flutter, android, ios, dynamic app, мобильные приложения*

ВВЕДЕНИЕ

Необходимость оптимизировать затраты на разработку мобильных приложений приводит к развитию кроссплатформенных решений, которые позволяют реализовывать мобильные программные продукты, покрывающие рынок устройств под управлением операционных систем iOS и Android, с единой кодовой базой. Одним из наиболее активно развивающихся программных инструментов для кроссплатформенной разработки на текущий момент является Flutter [1]. По данным Google Trends ежедневное количество запросов “Flutter” в поиске Google с осени 2019 года превышает запросы аналогичных решений, таких, как React Native и Xamarin [2]. Оптимизация разработки и поддержки нескольких операционных систем, впрочем, не оказывает влияния на трудозатратность таких процессов, как адаптация интерфейсов для конкретного пользователя и обновление приложений.

СУЩЕСТВУЮЩИЕ ПРОБЛЕМЫ

Адаптация интерфейсов

Удовлетворенность конечного пользователя от взаимодействия с интерфейсом напрямую зависит от контекста: индивидуальных характеристик пользователя, устройства и среды [3]. Для решения этой задачи существуют патенты, описывающие способы адаптации верстки экранов под особенности конкретных пользователей [4, 5]. Однако создание нескольких пользовательских интерфейсов для одной и той же функциональности вручную затруднительно, поскольку изменения контекста могут привести к многократному увеличению количества возможных адаптаций [6]. Таким образом, интерфейсы в большинстве случаев остаются универсальными для всех групп пользователей и не учитывают их индивидуальные особенности, что негативно сказывается на степени удовлетворенности пользователя от взаимодействия с приложением.

Обновление приложений

Выпуск очередной версии мобильного приложения может проводиться в штатном режиме и включать в себя новые возможности, расширяющие функционал программы, а может нести исправление критических ошибок и требовать экстренного обновления. Время, за которое обновление достигнет конечного пользователя, сложно прогнозировать, поскольку оно может варьироваться в зависимости от ряда факторов: политики торговых площадок App Store и Google Play, настройки автообновления на устройстве пользователя и др. Это означает, что пользователь приложения может получить обновление с задержкой или не получить его вовсе, что приведет к негативному опыту взаимодействия с системой или отказу от её использования.

КОНЦЕПЦИЯ РЕШЕНИЯ

Для решения проблем, указанных выше, нами предлагается инструмент, который позволяет генерировать пользовательский интерфейс на основе внешних и внутренних факторов для предоставления конечному потребителю индивидуального опыта взаимодействия с сервисом. К внешним факторам относятся данные пользователя, информация об устройстве, текущая дата и другие показатели

контекста. Внутренние факторы – то, что касается непосредственно внутреннего состояния системы. Такое разделение факторов влияния позволит более чётко классифицировать пользователей по группам для наибольшей персонализации.

Реализуется данный инструмент в виде библиотеки, которая подключается к Flutter-проекту, что должно облегчить внедрение и использование данного инструмента, включая получение последующих обновлений. Инструмент можно применять как для генерации приложения целиком, так и для реализации частичной функциональности.

Библиотека предоставляет контракт между серверной и клиентской частями, а также базовые модули для генерации стандартных элементов, экранов и логики. Если есть необходимость иметь кастомные элементы или особую логику работы различных частей приложения, есть возможность дополнить данные модули своими реализациями. Такое подход был принят с целью снизить ограничения, накладываемые на архитектуру проекта, и позволить разработчику гибко настраивать инструмент под свои потребности.

Использование на стороне клиента ограничивается следующими шагами:

1. подключение библиотеки;
2. подмена родительского экрана базовым;
3. дополнение маппера в случае, когда нужны кастомные виджеты.

Для использования на стороне сервера требуется:

1. сформировать экраны для клиента по шаблону, используя имеющиеся стандартные и кастомные элементы;
2. модифицировать отображаемый контент в зависимости от имеющихся факторов;
3. отправлять экраны в json-формате.

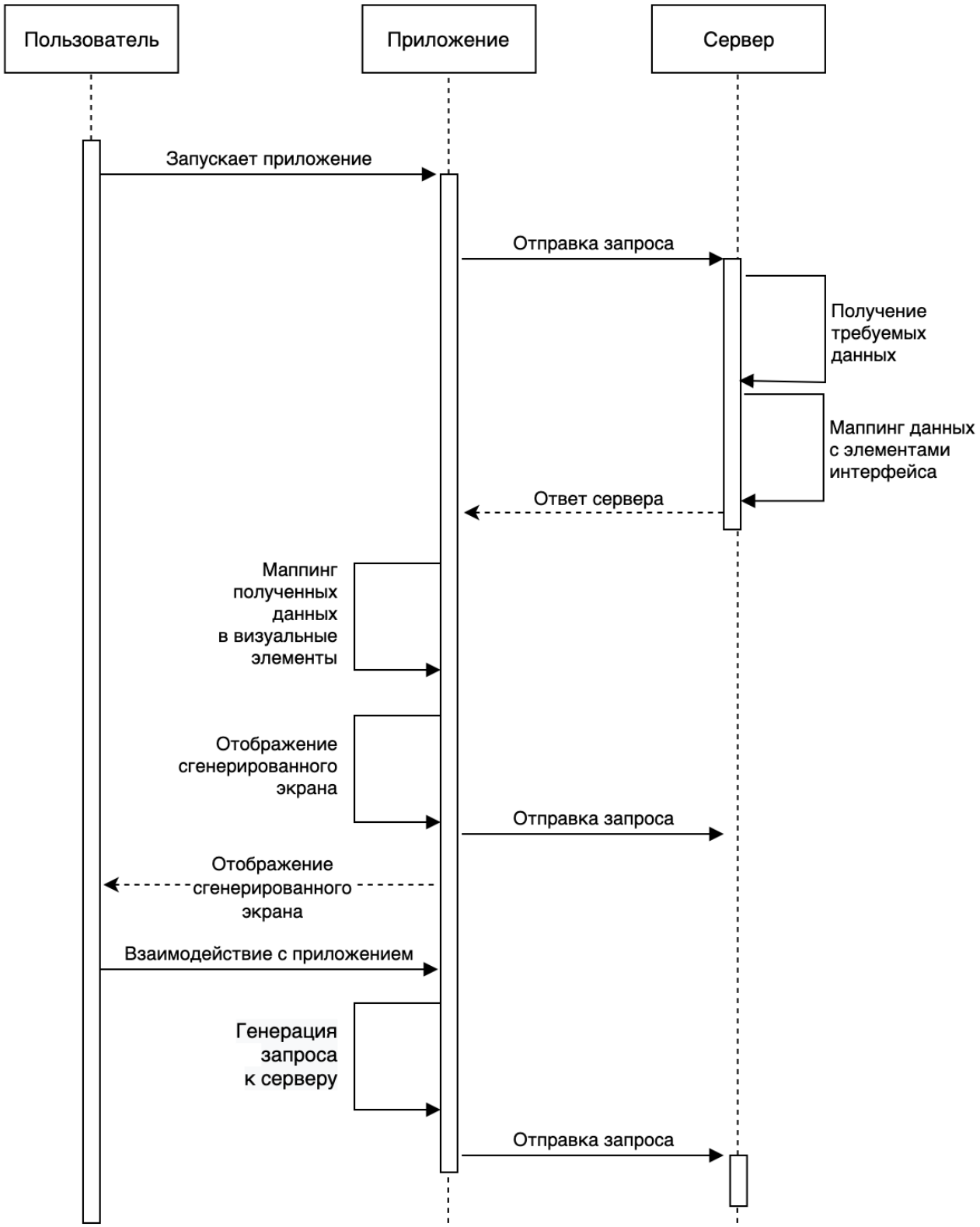

Рисунок 1. Схема взаимодействия сервера и приложения

Как можно заметить на диаграмме последовательности, изображенной на рис. 1, сервер отправляет отображаемые элементы интерфейса, набор которых может изменяться динамически в зависимости от различных факторов, которые определяются бизнес-требованиями. На стороне клиента переданные данные обрабатываются с помощью маппера и отображаются пользователю в виде экрана с требуемой ему информацией. После того, как пользователь производит какое-либо действие, приложение генерирует запрос в зависимости от этого действия по логике, описанной в том же ответе, в котором пришла информация для генерации экрана. Запрос отправляется на сервер, после чего выполнение алгоритма начинается заново.

АРХИТЕКТУРА ПРОГРАММНОГО РЕШЕНИЯ

Данный инструмент состоит из 3 основных компонентов. Упрощенная схема компонентов программного решения изображена на рисунке 2.

Core – модуль, который состоит из 3 основных пакетов:

- *Base* – пакет с базовыми классами, которые облегчают добавление новых экранов благодаря паттерну “Фасад”. Реализация экранов сводится к переопределению методов, задающих структуру интерфейса;
- *UI* – пакет с кастомными виджетами, которые являются обёртками над нативными виджетами из Flutter. Данные реализации служат для управления состоянием в stateful-виджетах, то есть тех виджетах, которые имеют своё состояние. К их числу относятся TextField, TimePicker, RadioGroup, CheckBox и другие.
- *Resources* – пакет с базовыми ресурсами и стилями приложения и виджетов, каждый атрибут которых может быть изменён соответствующим атрибутом виджета в ответе сервера;

Mappers – модуль с классами, которые переводят ответ сервера в элементы виджетов, отображаемые на экране. Имеет свою иерархию с подразделением мапперов по категории виджетов. Здесь же содержится ResourcesMapper, используемый для внедрения в инструмент кастомных пользовательских виджетов и ресурсов приложения, в число которых входят строковые ресурсы, иконки и текстовые стили;

MainPage – основной экран инструмента, на котором отображаются виджеты, передаваемые с сервера. Использует модуль Mappers для конвертации полученного с сервера ответа в flutter-виджеты.

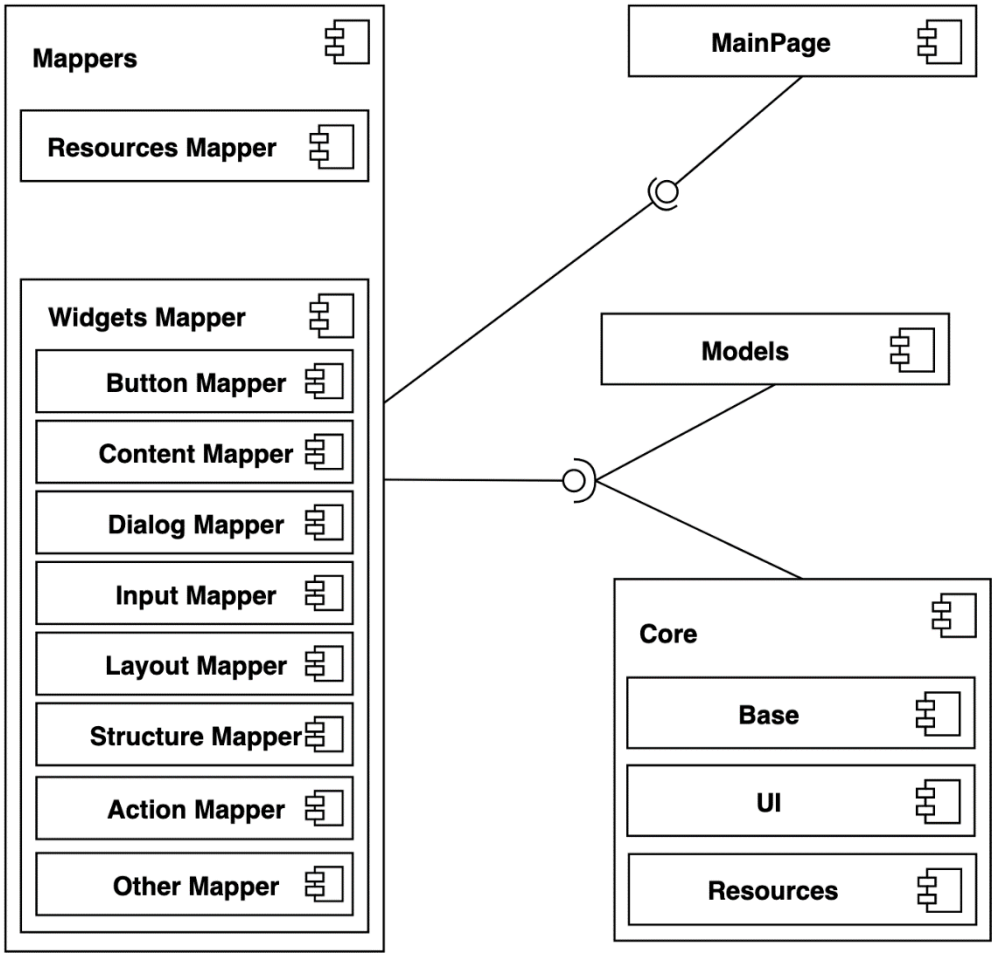

Рисунок 2. Диаграмма компонентов

ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

Для внедрения инструмента в текущий проект достаточно подключить его через файл pubspec.yaml и реализовать переход на экран MainPage. На вход данному экрану передается ссылка, по которой экран сможет получить данные для отображения и объект класса ResourcesMapper.

На стороне сервера потребуется реализовать генерацию виджетов для экрана, который связан с текущим URL, объединив их с теми данными, которые запросил пользователь.

Каждый объект виджета может содержать следующие поля:

- *id* – требуется для идентификации значения, полученного из данного виджета, при передаче данных на сервер;
- *type* – тип данного виджета; в качестве значений допускаются элементы из класса WidgetType или названия кастомных виджетов, которые имеются в проекте;
- *attributes* – набор атрибутов данного виджета; нужны для кастомизации внешнего вида виджета; передаются в формате «ключ: значение»; в качестве ключа допускаются элементы из класса WidgetAttribute, а в качестве значения – элементы из класса WidgetAttributeValue;
- *children* – список дочерних элементов данного виджета; передаются в формате обычного виджета;
- *actions* – список действий пользователя, которые требуется обрабатывать для данного виджета; каждое действие имеет следующие поля:
    - *type* – тип действия; в качестве значений допускаются элементы из класса UserActionType;
    - *effect* – тип эффекта от действия; в качестве значения допускаются элементы из класса EffectType;
    - *deeplink* – диплинк экрана, который требуется открыть в результате данного действия, если effect равен EffectType.openScreen или EffectType.openUrl;
    - *regex* – регулярное выражение, по которому требуется валидировать значение в результате данного действия, если effect равен EffectType.validation;
    - *child* – диалоговый виджет, который требуется отобразить в результате данного действия, если effect равен EffectType.openMessageDialog, EffectType.openActionDialog, EffectType.openPickerDialog, EffectType.openBottomDialog;
- *additionalChild* – опциональное поле для виджета WidgetType.drawer, в котором описывается header данного виджета;
- *values* – опциональное поле для виджетов WidgetType.dropdownButton, WidgetType.popupMenuButton и WidgetType.radioGroup, которое содержит набор строковых значений, отображаемых в данных виджетах.

В качестве примера рассмотрим типовой экран приложения, состоящий из пяти элементов пользовательского интерфейса: appBar, text, два элемента типа textField и button.

Входные данные представлены в формате JSON:

```
{
    “appBar”: {
        “type”: “appBar”,
        “attributes”: {“content”: “Регистрация”}
    },
    “children”: [
        {
            “type”: “padding”,
            ...
            “children”: [
                ...
                {
                    “type”: “text”,
                    “attributes”: {
                    “content”: “Для регистрации придумайте\nлогин и пароль”,
                    “textStyle”: “textDisplay4”
                    }
                },
                ...
                {
                    “type”: “textField”,
                    “attributes”: {“prefixIcon”: “login”, “hint”: “Логин”}
                },
                ...
                {
                    “type”: “textField”,
                    “attributes”: {“prefixIcon”: “password”, “hint”: “Пароль”}
                },
                ...
                {
                    “type”: “raisedButton”,
                    “attributes”: {“content”: “Отправить”, “radius”: “16”}
                },
                ...
            ]
        }
    ]
}
```

Пользовательский интерфейс, сгенерированный по данному входному набору, представлен на рис. 3.

Таким образом, при добавлении новых элементов пользовательского интерфейса на экран, изменении или удалении существующих компонентов на сервере будет соответствующим образом изменяться интерфейс приложения для Flutter.

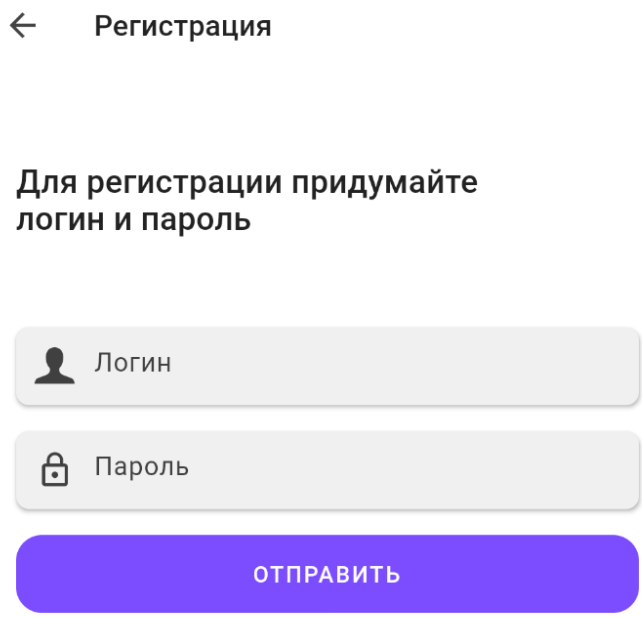

Рисунок 3. Пример экрана регистрации

ЗАКЛЮЧЕНИЕ

Предложенный инструмент может быть использован на любом этапе разработки программного решения, что расширяет область его применения. При использовании данного инструмента пользователь получает возможность гибкой адаптации интерфейсов приложения под потребности определенных групп пользователей и мгновенную поставку обновлений на устройства всех пользователей, что, помимо прочего, дает возможность минимизировать риски от различных дефектов, выявленных после выпуска очередной версии.

СПИСОК ЛИТЕРАТУРЫ

1. *Showcase* // Flutter Developers. URL: https://flutter.dev/showcase.
2. *Flutter, React Native, Xamarin* // Google Trends. URL: https://trends.google.com/trends/explore?q=Flutter,React%20Native,Xamarin
3. *Yigitbas E., Jovanovikj I., Biermeier K. et al.* Integrated model-driven development of self-adaptive user interfaces // Software and Systems Modeling. 2020. URL: https://link.springer.com/content/pdf/10.1007/ s10270-020-00777-7.pdf.
4. *Displaying dynamic user interface elements in a social networking system* // Google Patents. URL: https://patents.google.com/patent/ US10296159B2/en.

5. *Methods and systems for partial personalization during mobile application update* // Google Patents. URL: https://patentimages.storage.googleapis.com/7f/0e/06/aef189685fa6b7/US9582267.pdf
6. *Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu.* Adaptive Model-Driven User Interface Development Systems // ACM Computing Surveys. V.47, No.1, Article 9. 2014. 33 p.

DYNAMICALLY GENERATED UI ON FLUTTER

A. Y. Usachev

Higher School of Information Technologies and Intelligent Systems, Kazan Federal University

usacheow.ar@gmail.com

Abstract

The development team develops visual interfaces that satisfy the needs of all user groups and then distribute the new version of the product to all users. This article is aimed to propose the concept of a tool for generating dynamic screens, which can help to solve the following problems: adaptation of the application interface and quick release of updates.

Keywords*: flutter, android, ios, dynamic app*

REFERENCES

1. *Showcase* // Flutter Developers. URL: https://flutter.dev/showcase.
2. *Flutter, React Native, Xamarin* // Google Trends. URL: https://trends.google.com/trends/explore?q=Flutter,React%20Native,Xamarin
3. *Yigitbas E., Jovanovikj I., Biermeier K. et al.* Integrated model-driven development of self-adaptive user interfaces // Software and Systems Modeling. 2020. URL: https://link.springer.com/content/pdf/10.1007/ s10270-020-00777-7.pdf.
4. *Displaying dynamic user interface elements in a social networking system* // Google Patents. URL: https://patents.google.com/patent/US10296159B2/en.

5. *Methods and systems for partial personalization during mobile application update* // Google Patents. URL: https://patentimages.storage.googleapis.com/7f/0e/06/aef189685fa6b7/US9582267.pdf
6. *Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu.* Adaptive Model-Driven User Interface Development Systems // ACM Computing Surveys. V.47, No.1, Article 9. 2014. 33 p.

СВЕДЕНИЯ ОБ АВТОРЕ

УСАЧЕВ Артемий Юрьевич – студент 2 курса магистратуры Высшей Школы ИТИС Казанского (Приволжского) Федерального университета.

Artemy Yurievich USACHEV – postgraduate student of the Higher School of ITIS Kazan (Volga region) Federal University.

email: usacheow.ar@gmail.com

Материал поступил в редакцию 3 апреля 2020 года