Существует два основных подхода к созданию мобильного приложения: нативный и кроссплатформенный.

**Нативные приложения** — это приложения, разработанные в их

собственной операционной среде

**Преимущества нативной разработки:**

∙ стабильная и быстрая производительность;

∙ меньше ограничений в архитектуре и функциях;

∙ интерфейс выполнен в полном соответствии с платформой.

Иногда нативная разработка для обоих устройств может быть очень

затратной, особенно, когда сроки ограничены и для создания приложения

требуется несколько команд. Именно в этот момент кроссплатформенная

разработка проявляет себя.

**Кроссплатформенные приложения** — это приложения, разработанные в «общей операционной среде», такой как Flutter. Это универсальное решение, которое поддерживается несколькими мобильными платформами одновременно.

**Преимущества кроссплатформенной разработки:**

∙ требуется меньше специалистов;

∙ разработка занимает меньше времени и ресурсов;

Кроссплатформенные решения могут иногда работать медленнее, чем

нативные, из-за медленного отклика интерфейса, потому что они обрабатывают больше данных. Также, интерфейс приложения обычно универсален, и реализовать отдельные элементы каждой конкретной ОС не так просто. Кроме того, в кроссплатформенных решениях гораздо сложнее реализовать возможность использования всех функций программного обеспечения смартфона (микрофон, геолокация, плеер и камера, голосовое управление, автономный режим и т. д.).

В рамках данной выпускной квалификационной работы был выбран

кроссплатформенный фреймворк Flutter.

**Flutter** — это мобильный SDK с открытым исходным кодом, который

можно использовать для создания приложений для Android и

iOS из одной базы кода.

Язык, использованный в Flutter называется Dart.

**Dart** — это универсальный объектно-ориентированный язык програм-

мирования с открытым исходным кодом и синтаксисом в стиле C

**Основные преимущества** данного фреймворка: прост в освоении, быстрая компиляция, совместимость в разных версиях ОС, документация и

поддержка на Android Studio и VS Code.

**Основной недостаток** заключается в том, что фреймворк относительно молодой и специалистов, которые хорошо в нем разбираются немного.

**Flutter** состоит из двух важных частей:

– SDK (Software Development Kit)

– Фреймворк (библиотека пользовательского интерфейса, основан-

ная на виджетах)

Более подробно общая архитектура расписана на слайде.

В качестве системы хранения данных был выбран облачный сервис

FireBase от Google.

Firebase предлагает несколько продуктов, которые могут значительно

сократить время написания кода, но нас интересует Firebase Cloud Storage

и Firebase Cloud Firestore.

**Firebase Cloud Storage** предназначена для хранения и обслуживания

пользовательского контента, например фотографии или видео.

**Cloud Firestore —** это база типа document-model. Это означает, что все данные хранятся в объектах, называемых документами, в виде ключ-значение. Где значение может быть чем угодно, от строковой переменной до бинарного файла. Такие документы группируются в коллекции.

Также на текущем слайде показан пример создания коллекции для приложения. В данном случае база используется только для хранения информации по блюдам. В дальнейшем её функциональность можно расширить путем использования таких компонентов как авторизация и обработка заказа.

На основе всех изученных материалов, необходимо было создать для

заведения мобильное доставочное приложение, используя фреймворк Flutter.

Прежде всего, нам необходимо расписать его возможности, которые указаны на слайде.

Далее, необходимо построить блок-схему, чтобы упростить себе зада-

чу и понимать, откуда и куда мы можем прийти в приложении.

Цвет приложения был выбран, основываясь на тематике самого заве-

дения, а также при помощи официального сайта Material Design, где пред-

ставлена интерактивная палитра, для подбора первичных и вторичных от-

тенков.

Центральным элементом приложения на Flutter являются виджеты. Фактически это те визуальные компоненты, из которых состоит графический интерфейс. Рассмотрим экран «Главная» и построим иерархическую структуру виджетов для него.

* Scaffold() – виджет, который создает визуальную структуру
* Column() – виджет, который создает столбец из дочерних виджетов;
* Drawer() – виджет, для создания многоуровневого меню;
* AppBar() – виджет, необходимый для создания верхней информа-

ционной панели;

Построение горизонтальных списков в приложении

для удобства вынесено в отдельный файл, чтобы не нагружать проект

повторяющимся кодом.

В основе разработки логики для приложения лежит паттерн проекти-

рования BLoC(Buisness Logic Component). Данная компонента полностью

реактивна , что позволяет нам в реальном времени получать информацию

от пользователя.Для реализации был использован класс StreamController()

который использует две составляющие: sink и stream.Поток Sink получает

событие на входе, а поток stream "прослушивает"полученные объекты и

реагирует на состояние после проверки валидации. Как раз первый вход в приложение реализован через данную логику и в этом случае sink принимает информацию из текстовых полей экрана авторизации а stream проверяет их на валидацию и выдает ответ: в случае ошибки появится информационное уведомление а в случае правильного заполнения, пользователь перейдет к следующему экрану.

Ответы на возможные вопросы по слайдам:

(Слайд с архитектурой Flutter)

Когда мы пишем приложение Flutter, используя Dart, мы остаемся на уровне *Flutter Framework* (выделено зеленым цветом).

*Flutter Framework* взаимодействует с *Flutter Engine* (синим цветом) через слой абстракции, называемый [Window](https://api.flutter.dev/flutter/dart-ui/Window-class.html). Этот уровень абстракции предоставляет ряд API для косвенного взаимодействия с устройством.

Также через этот уровень абстракции *Flutter Engine* уведомляет *Flutter Framework*, когда:

* событие, представляющее интерес, происходит на уровне устройства (изменение ориентации, изменение настроек, проблема с памятью, состояние работы приложения…)
* какое-то событие происходит на уровне стекла (=жест)
* канал платформы отправляет некоторые данные
* но также и в основном, когда **Flutter Engine готов к рендерингу нового кадра**

Слайд с бизнес-логикой:

**Паттерн проектирования** — это часто встречающееся решение определённой проблемы при проектировании архитектуры программ.