

Firebase – это не только PUSH-сообщения

Андрей Немцев
Руководитель мобильной практики

2020 г.

Платформа Firebase



Содержание

- Анализ проблем при помощи Firebase Performance
- Перенос базы данных в облако Firebase
- Авторизация пользователей без бэкенда
- Для каких проектов стоит применять







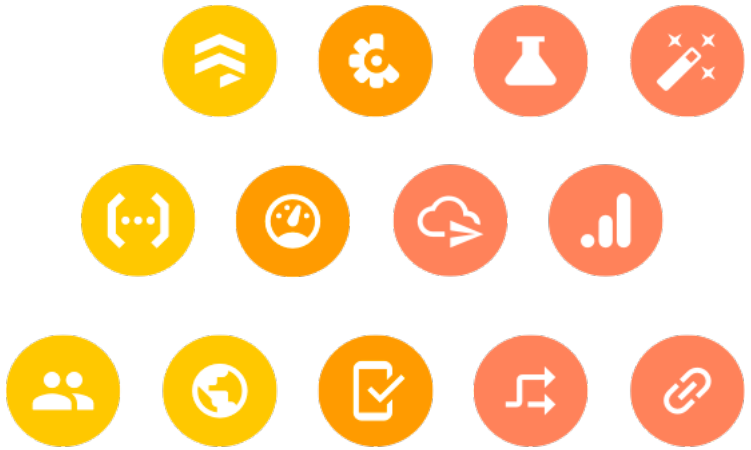
Платформа Firebase

Firebase

Google

Наиболее известные сервисы

- Cloud Messaging
- Analytics
- Crashlytics
- Remote Config



Платформа Firebase

Firebase

Google

Наиболее известные сервисы

- Cloud Messaging
- Analytics
- Crashlytics
- Remote Config





USETECH

Мобильное решение для сервиса велопроката

Приложение

- Отображение данных о наполненности велопарковок
- Информация о текущей поездке
- История поездок клиент
- Оплата дост

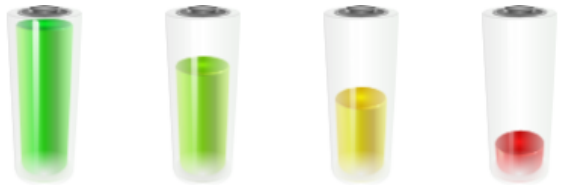


Проблемы первых версий

- наличие ошибок выполнения клиентских запросов



- жалобы клиентов на большой объем трафика



- повышенный расход заряда батареи

Поиск причины проблем



Возможные причины

- Таймауты выполнения запросов
- Ошибки выполнения при большой нагрузке

- Слишком частый опрос сервера
- Неоптимальное представление ответа в API – «много лишних данных»

Анализ проблем при помощи Firebase Performance Monitoring

- Анализ сетевых проблем
- Производительность UI



Анализ проблем при помощи Firebase Performance Monitoring



apply plugin: 'com.google.firebase.firebase-perf’

classpath 'com.google.firebase:perf-plugin:1.3.1'



pod 'Firebase/Performance’

pod install

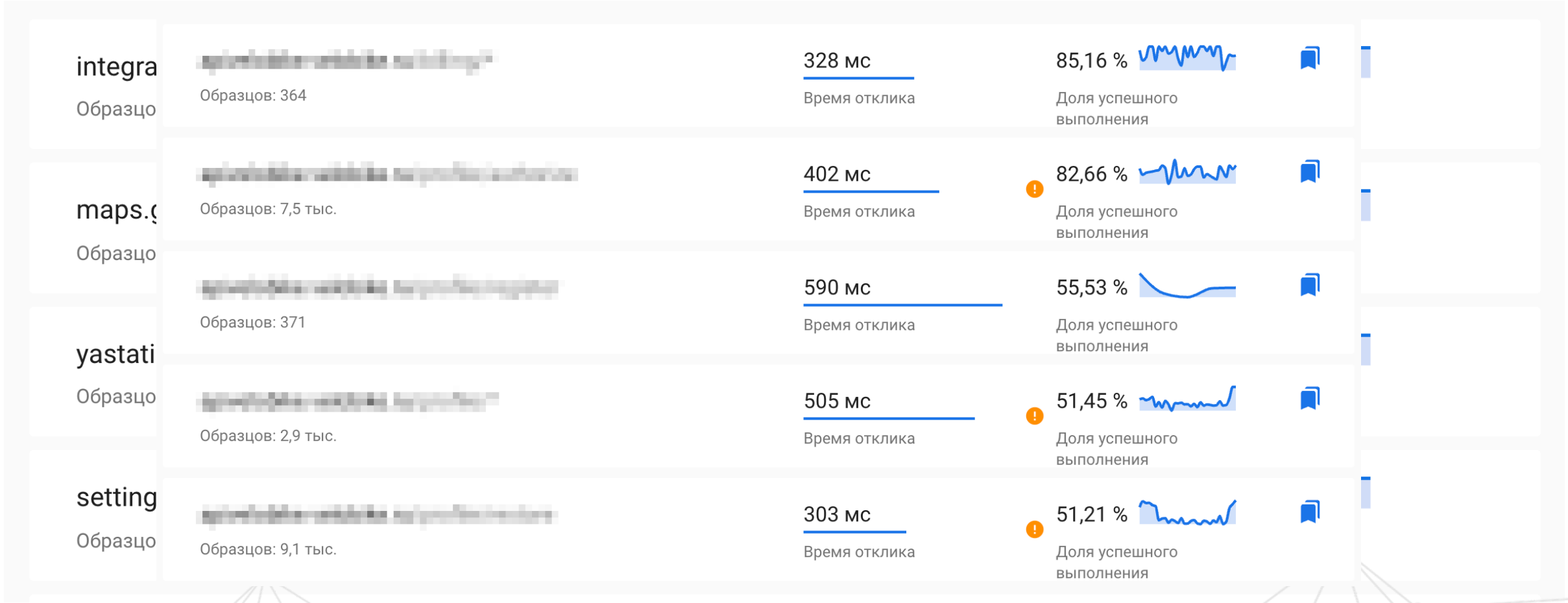
import Firebase

FirebaseApp.configure()

Выявление проблем сетевого уровня

integrationapi.net/push/sdk/messages/events Образцов: 196 тыс.	216 мс Время отклика	100 % Доля успешного выполнения
maps.googleapis.com/maps/api/staticmap Образцов: 8,6 тыс.	333 мс Время отклика	100 % Доля успешного выполнения
yastatic.net/weather/i/icons/islands/*/*.png Образцов: 33 тыс.	270 мс Время отклика	100 % Доля успешного выполнения
settings.crashlytics.com/spi/v2/platforms/android/apps… Образцов: 88 тыс.	339 мс Время отклика	100 % Доля успешного выполнения

Выявление проблем сетевого уровня



Firebase Performance. Выявление проблем сетевого уровня

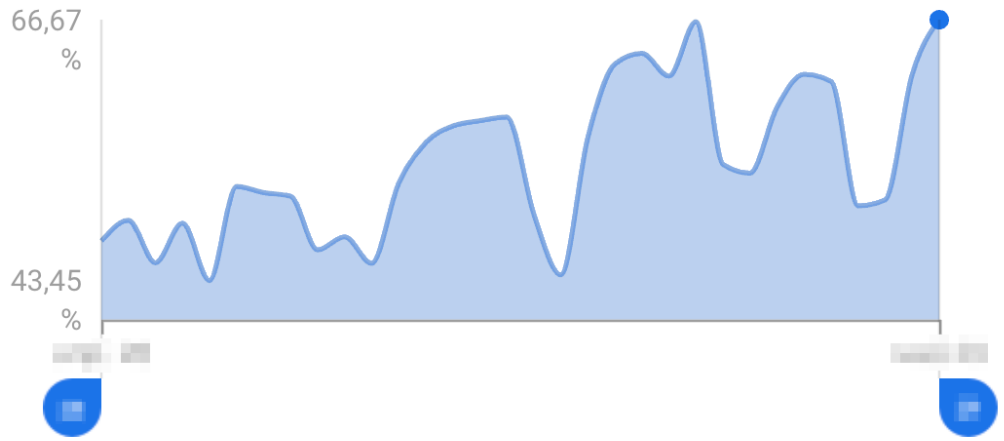
Последние 30 дней

В динамике | Версия

Динамика изменений

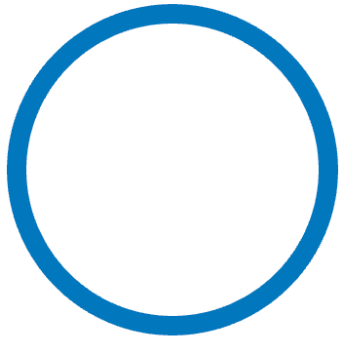
● Коды успешных ответов - [redacted]

66,67 % +19,69

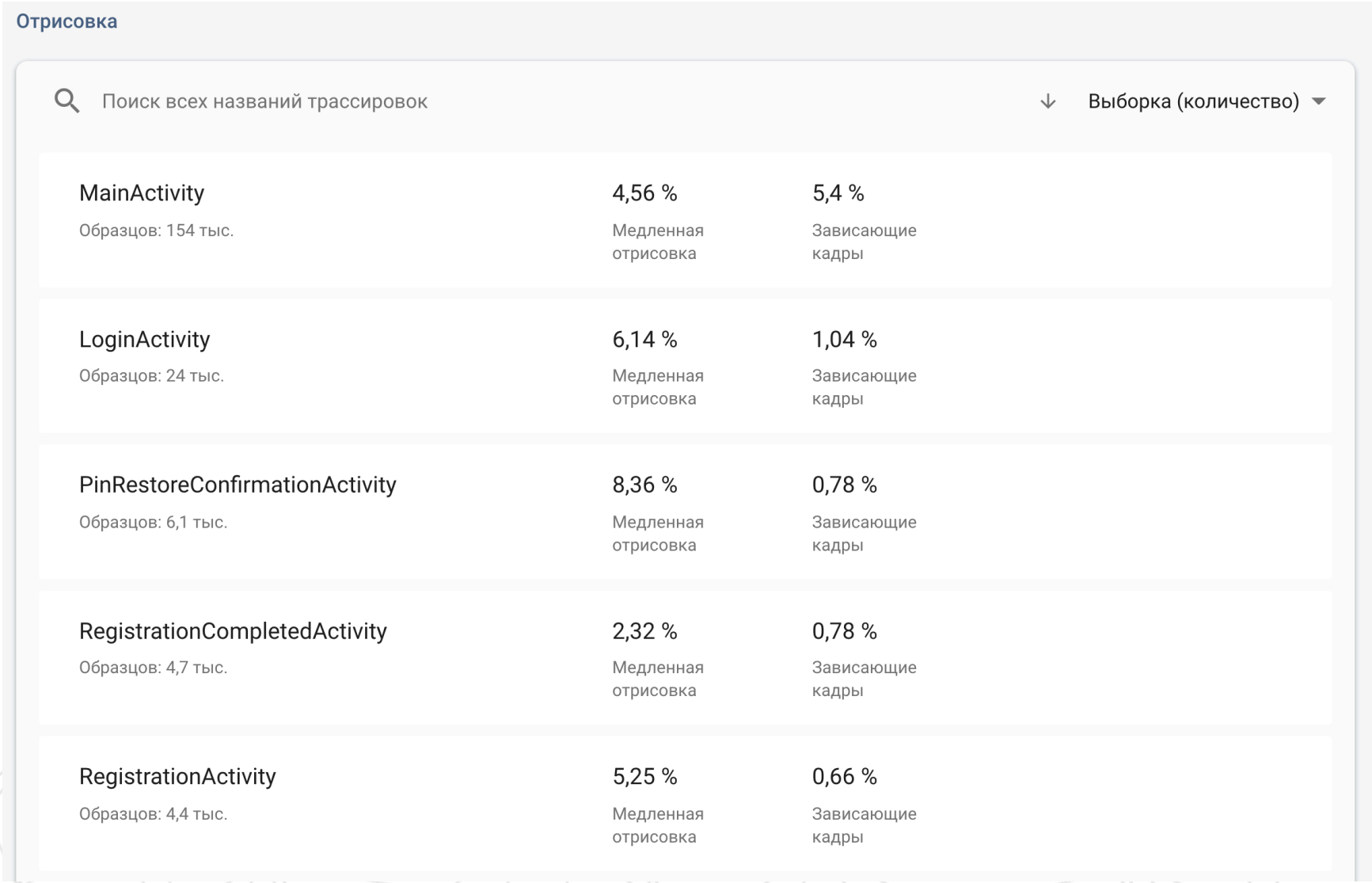


Коды ошибок – Последние 30 дней

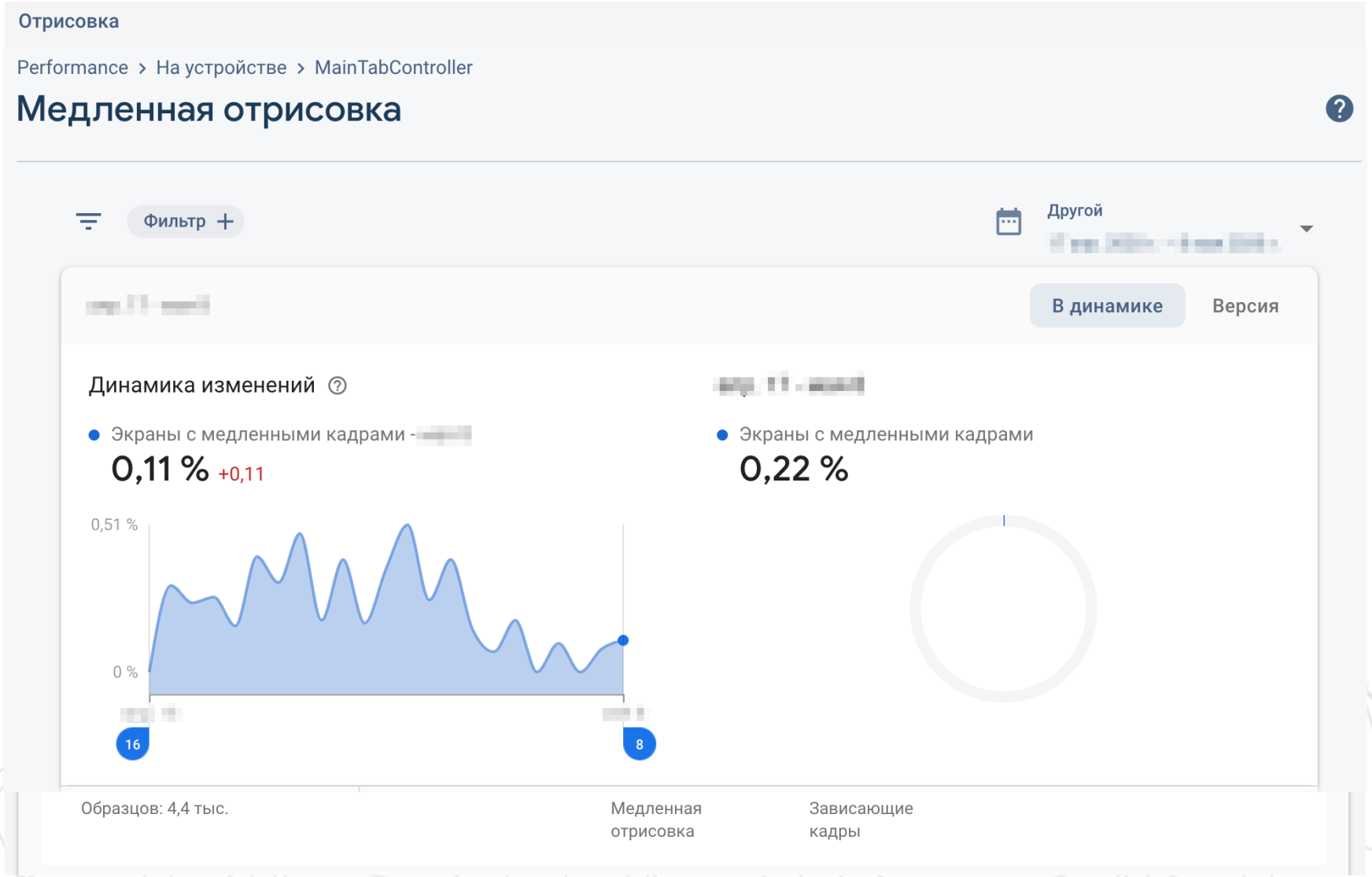
● 500 100 %



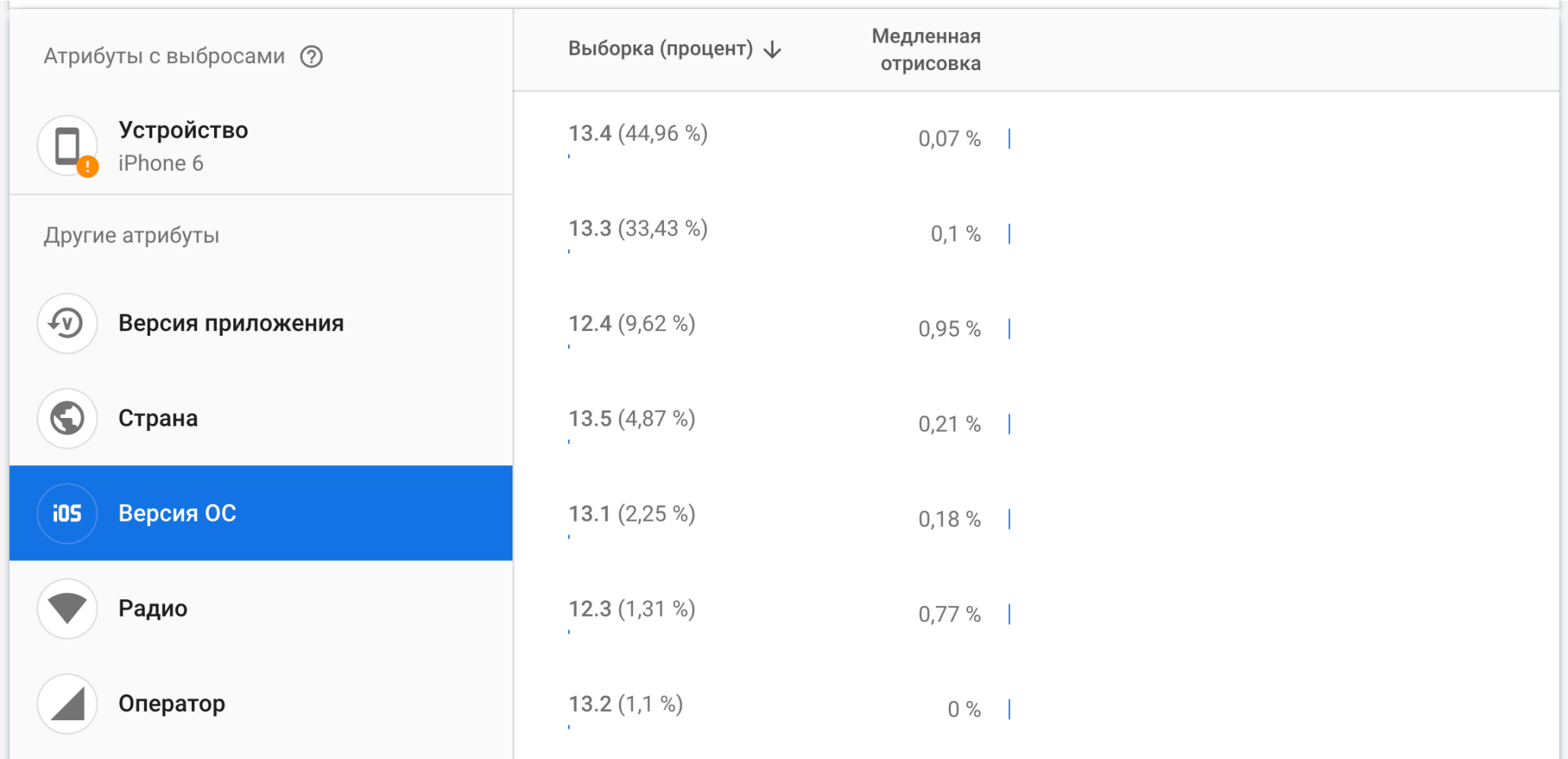
Выявление проблем низкой производительности UI



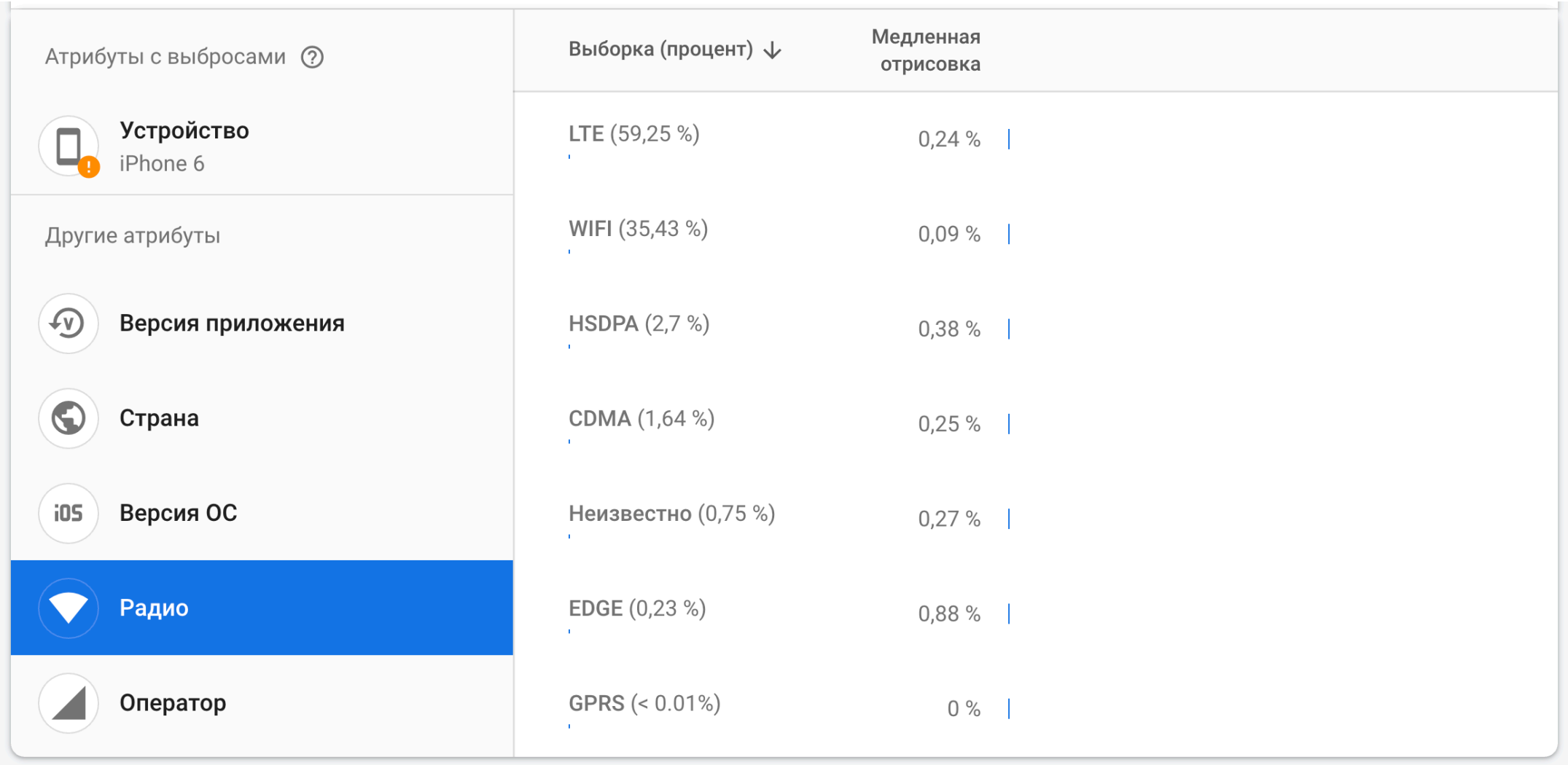
Выявление проблем низкой производительности UI



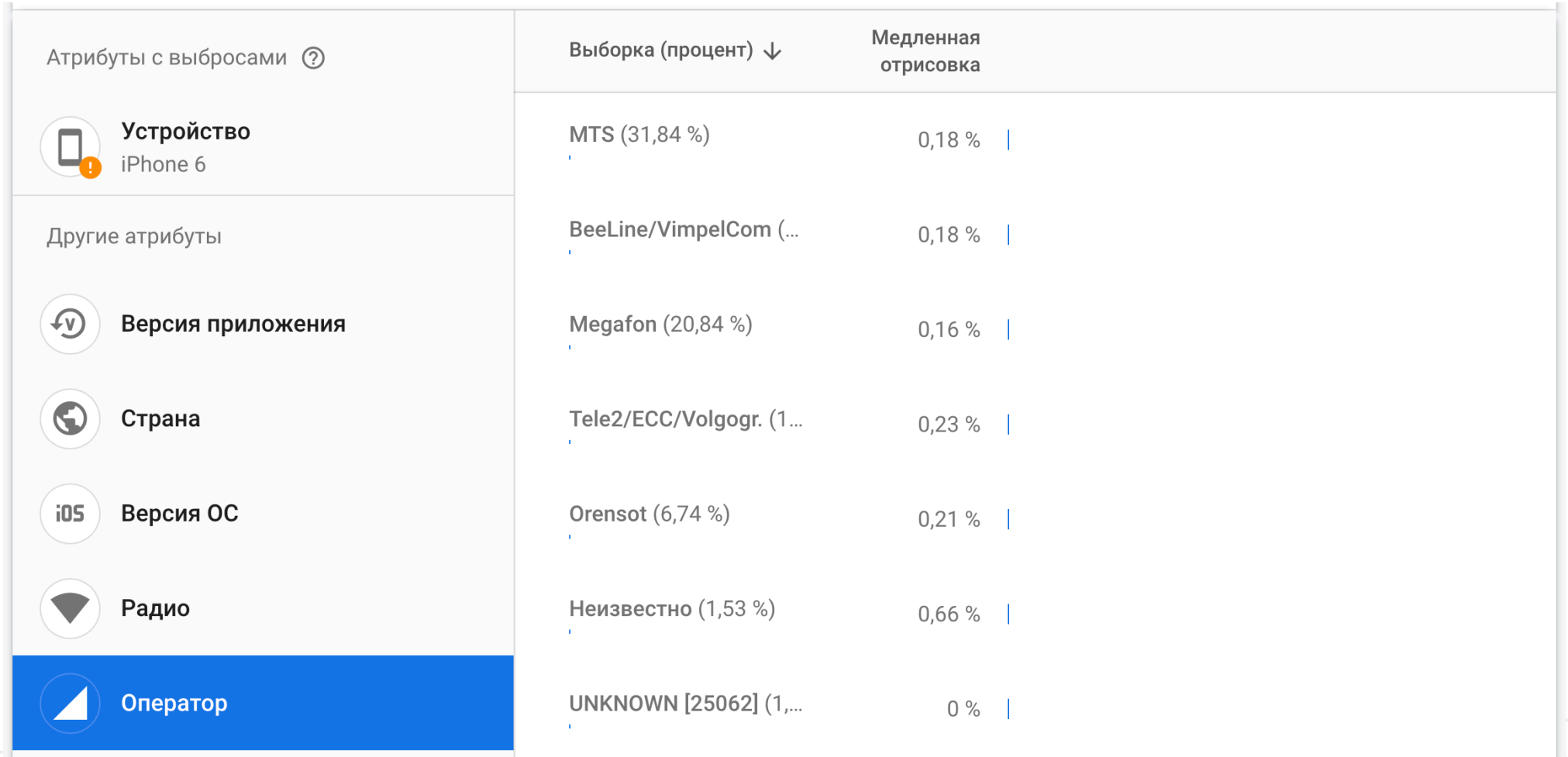
Выявление проблем низкой производительности UI



Выявление проблем низкой производительности UI



Выявление проблем низкой производительности UI



Анализ проблем при помощи Firebase Performance Monitoring





Ваши вопросы

Перенос базы данных в облако Firebase

- Задачи синхронизации данных
- Решения при помощи собственной инфраструктуры
- Решения при помощи платформы Firebase
- Наш опыт применения Firebase Realtime Database







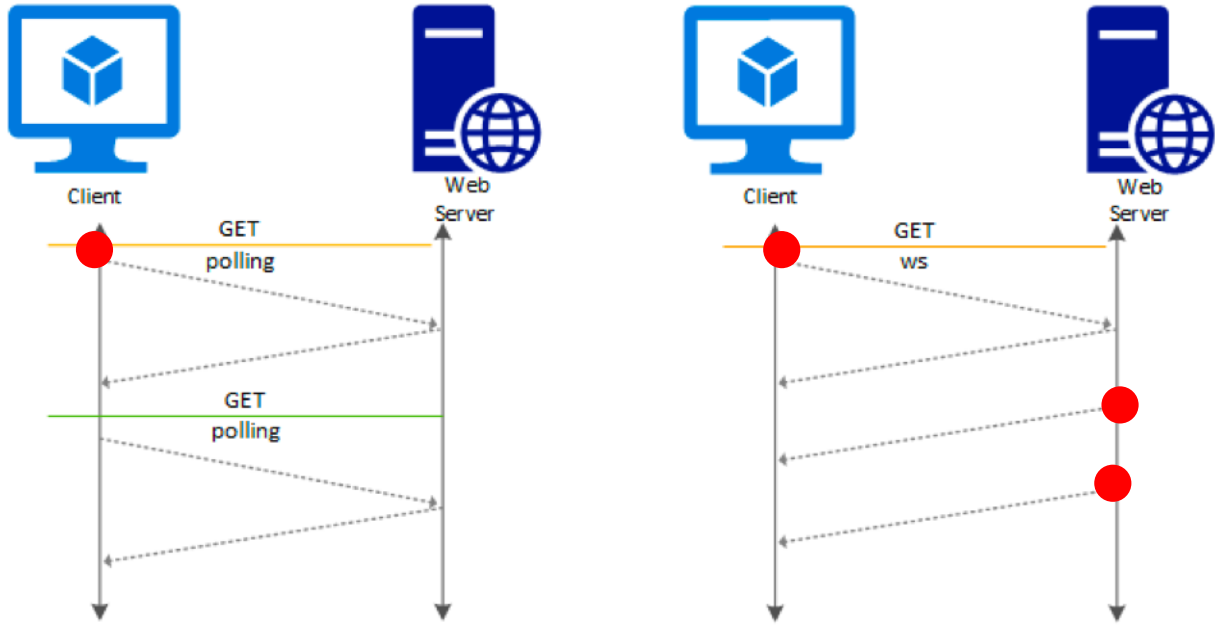
Задачи синхронизации данных на стороне приложения

- Определение факта наличия изменений
- Реализация кеширования данных на стороне клиента
- Способ передачи информации о новых / модифицированных / удаленных записях
- Обработка ошибок сетевого подключения



Определение факта наличия изменений

- Polling
- Long Polling
- Websocket



Определение факта наличия изменений

- Polling
- Long Polling
- **Websocket**

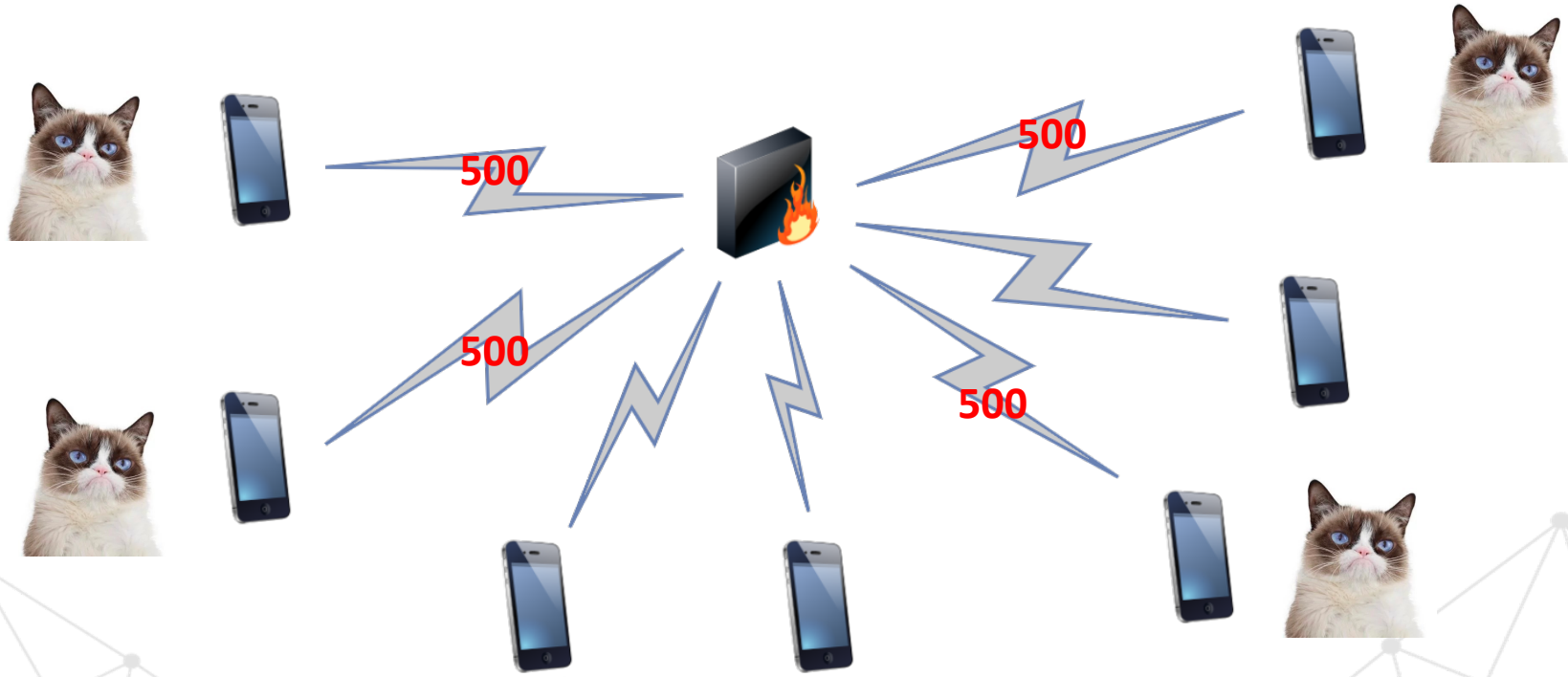


Плюсы:
- меньше объем трафика
- меньше нагрузка на сервер

Минусы:
- сложнее обработка ситуации смены сети клиентом

Определение факта наличия изменений

- Polling в первой версии приложения
- Больше клиентов – больше запросов к бэкенду



Выбор платформы

Основное требование -

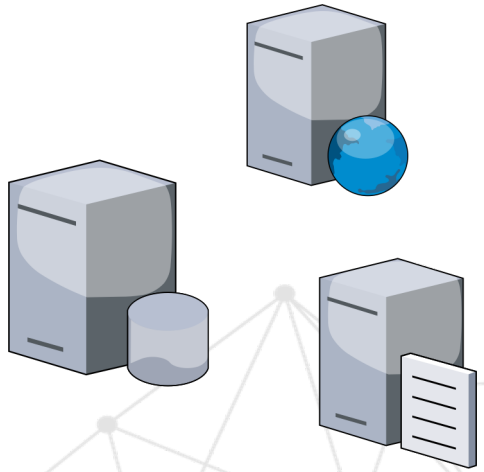
синхронизация данных в режиме реального времени

Cloud-based
BaaS



VS

Своя
инфраструктура



Выбор платформы

Основное требование -

синхронизация данных в режиме реального времени



Firebase Realtime Database

VS

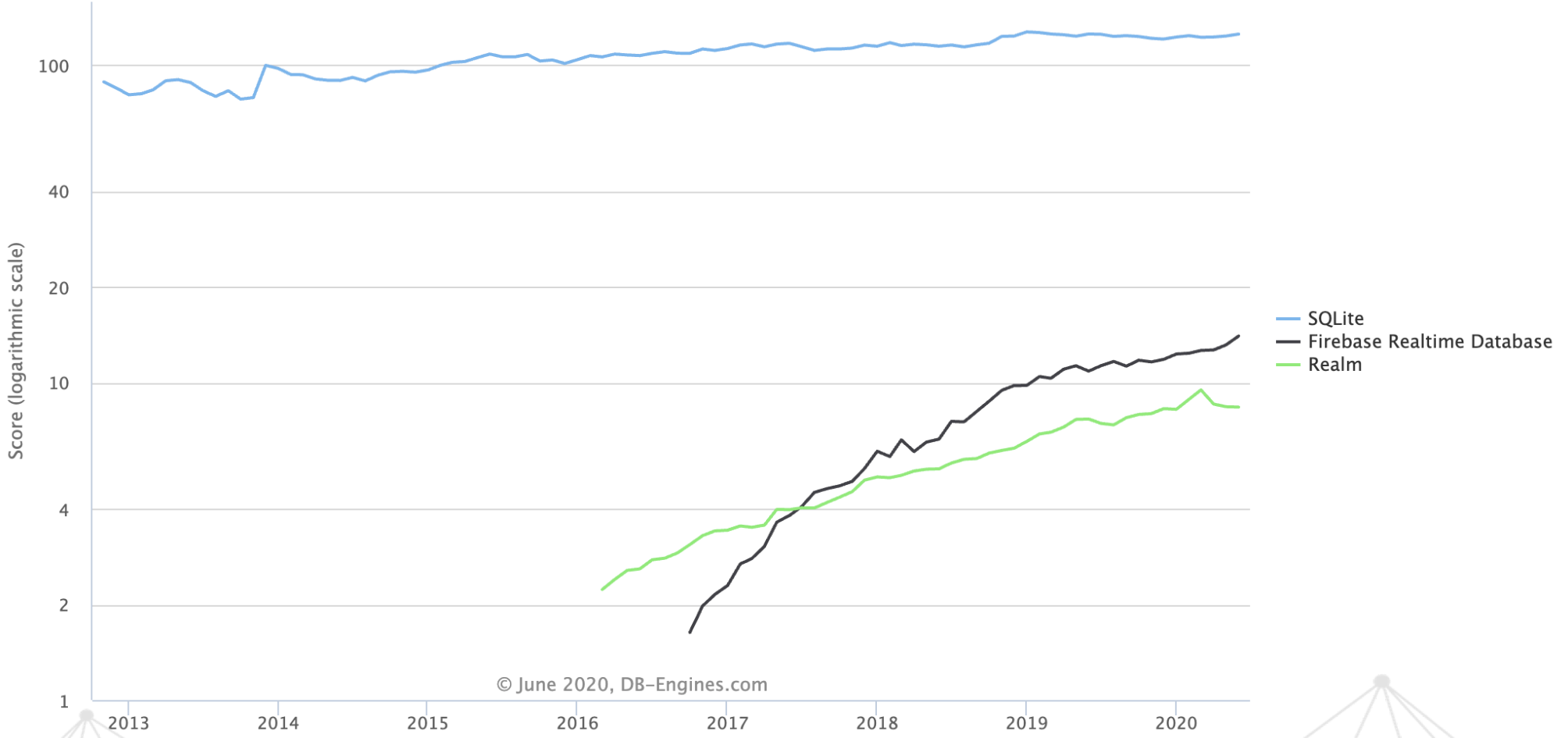


realm

VS

Своя инфраструктура

Выбор платформы



Платформа Firebase



Firebase Realtime Database

Firebase Realtime Database

Облачная **NoSQL** БД, позволяющая выполнять хранение и синхронизацию данных в режиме реального времени

До 200К одновременных подключений

Синхронизация в режиме реального времени



Платформа Firebase



Cloud Firestore

Firebase Cloud Firestore
«Наследник» Realtime Database с поддержкой автоматического масштабирования

Автоматическое масштабирование

Поддержка запросов для выборки документов

Синхронизация в режиме реального времени



Realtime Database как решение задачи синхронизации данных

Мгновенное уведомление клиента об изменениях

механизм подписки (PUB/SUB)



Realtime Database как решение задачи синхронизации данных

```
private var database: DatabaseReference

val parkingsListener = object : ValueEventListener {
    override fun onDataChange(dataSnapshot: DataSnapshot) {
        // Parse Parkings data and use the values to update the UI
    }

    override fun onCancelled(databaseError: DatabaseError) {
        // Getting Parkings failed, log a message
    }
}


database.child("parkings").addValueEventListener(parkingsListener)
```



Realtime Database как решение задачи синхронизации данных

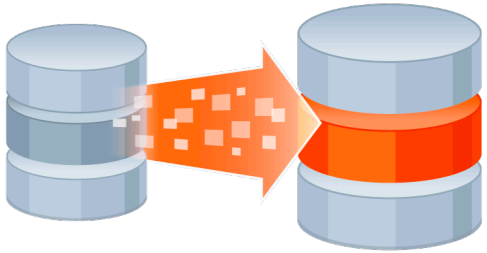


```
var ref = Database.database().reference()


refHandle = ref.child(“parkings”).observe(DataEventType.value,
with: { (snapshot) in
        // Parse Parkings data and use the values to update the UI
}) { (error) in
        // Getting Parkings failed, log a message
}
```

Реализация кеширования данных на стороне клиента

- Необходимость реализации миграций для обработки случаев изменения формата данных



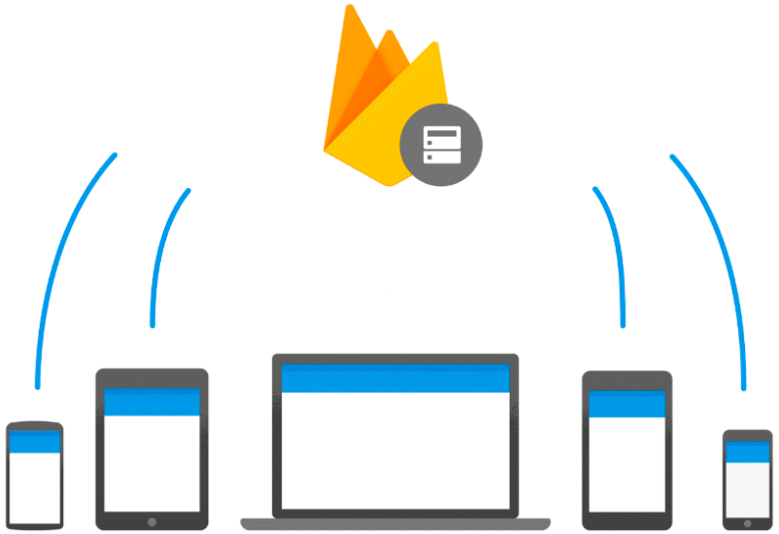
- Вопрос уведомления UI слоя приложения об обновлении данных в кэше





Realtime Database как решение задачи синхронизации данных

- Подход **Offline First** – при включенном функционале «Offline Capabilities» ранее полученные данные извлекаются сначала из локального кэша
- Автоматическая обработка случая изменения формата данных и обновление локального кэша
- Механизм подписки на события обеспечивает прозрачную синхронизацию данных из кэша/сети – единственное событие **onDataChange** для клиента Firebase SDK



Firebase Realtime Database. Offline Capabilities

Включение «Offline Capabilities» позволило сократить объем трафика



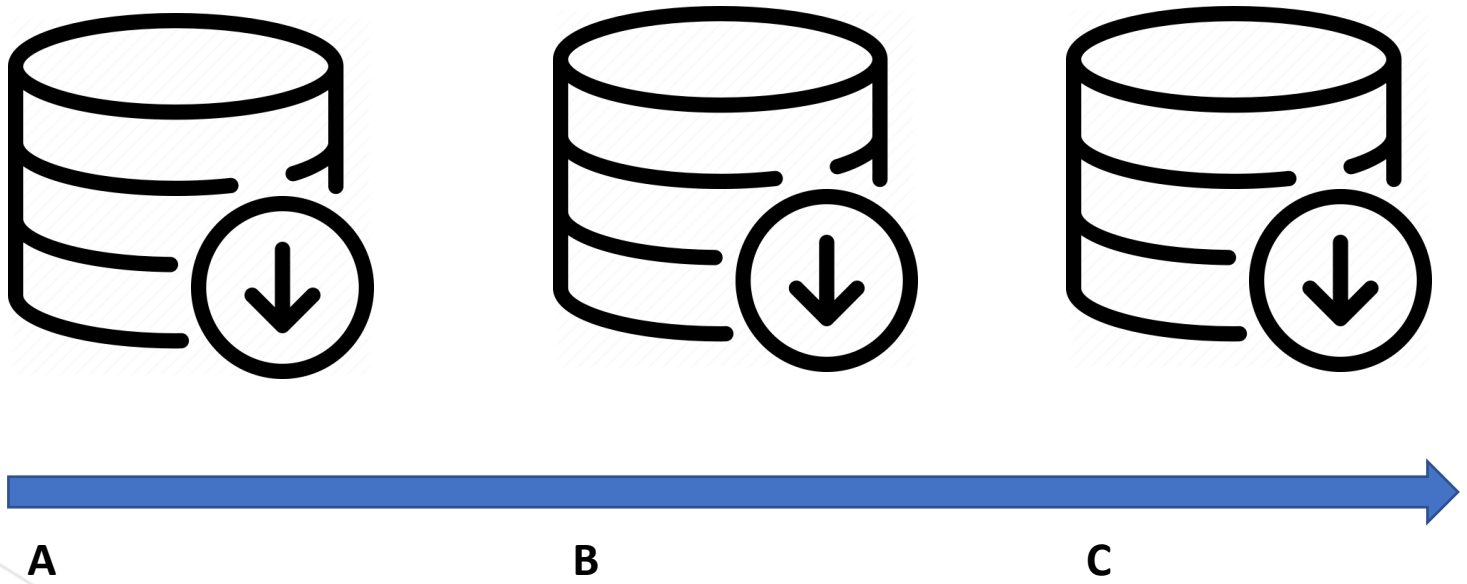
```
Firebase.database.setPersistenceEnabled(true)
```



```
Database.database().isPersistenceEnabled = true
```

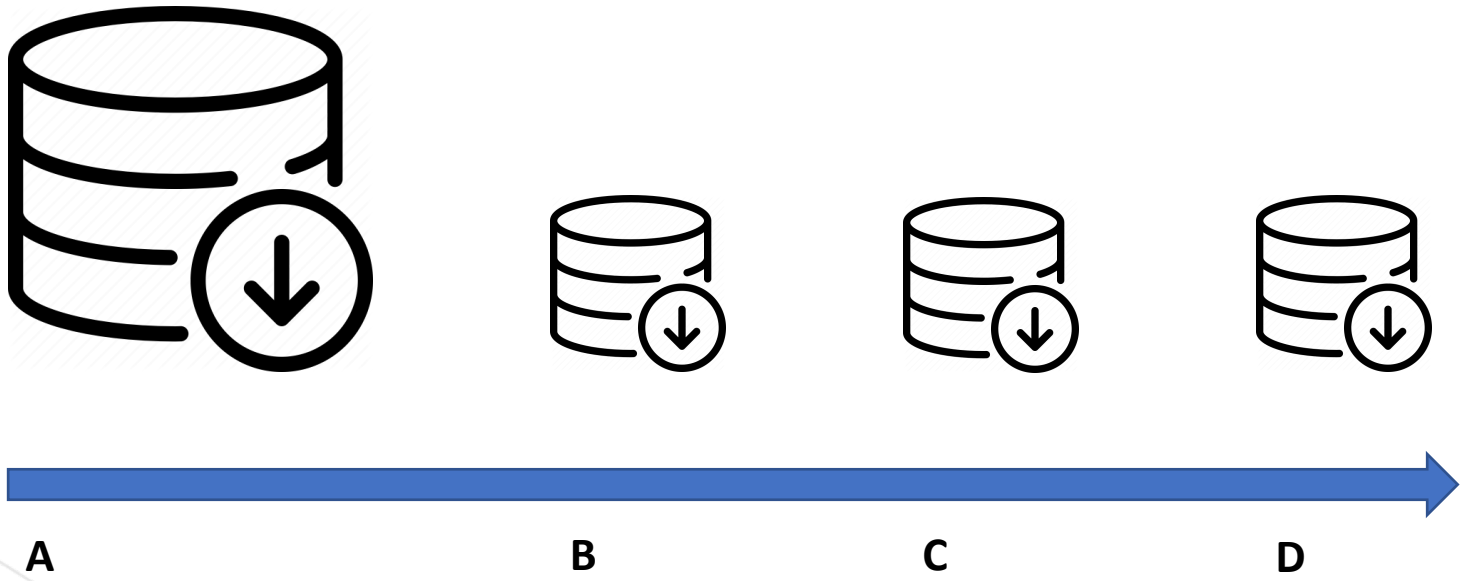
Способ передачи информации об изменениях

- **Простое решение**
  передача полного «снимка» в ответ на каждый запрос данных



Способ передачи информации об изменениях

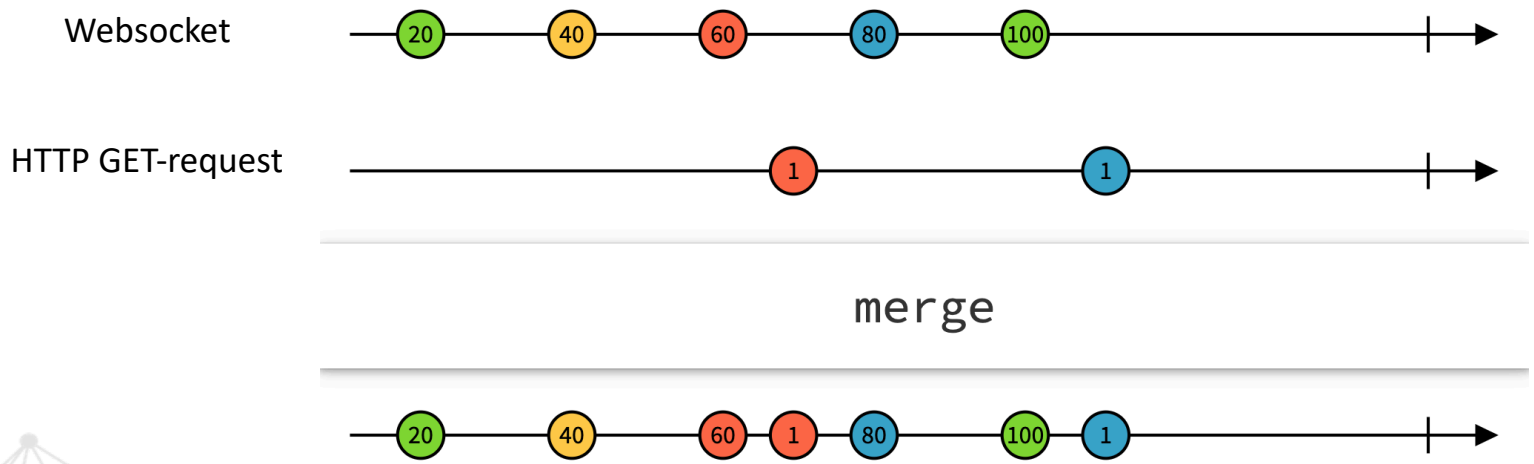
- **Сложное решение** – передача полного «снимка» в ответ на первый запрос и информации только об изменениях в последующих



Передача информации об изменениях при синхронизации данных

Проблема синхронизации данных, полученных по различным каналам

HTTP-запрос – получение «снимка»
Websocket-соединение – получение «изменений»



Realtime Database. Функционал из коробки для решение задачи синхронизации данных

- При изменении отдельных полей синхронизируемой сущности в сторону мобильного клиента передается не весь «снимок», а только «изменения»



- Автоматическое удаление из «снимков» тех данных, которые были удалены на стороне Realtime Database – вопрос слияния «изменений» с последним полученным «снимком» решается на уровне Firebase SDK

- Изменения в синхронизируемых данных на уровне приложения доступны в виде «снимков» (DataSnapshot)

Realtime Database. Синхронизация данных о текущей поездке

JSON-представление текущей поездки

```
{
  "ride": {
    "userID": {
     "start": "2020-05-10T10:15:32Z",
     "from": "b432c",
     "bike_number": "11432",
     "bike_type": "electro"
    }
  }
}
```

Отображение изменений режиме реального времени

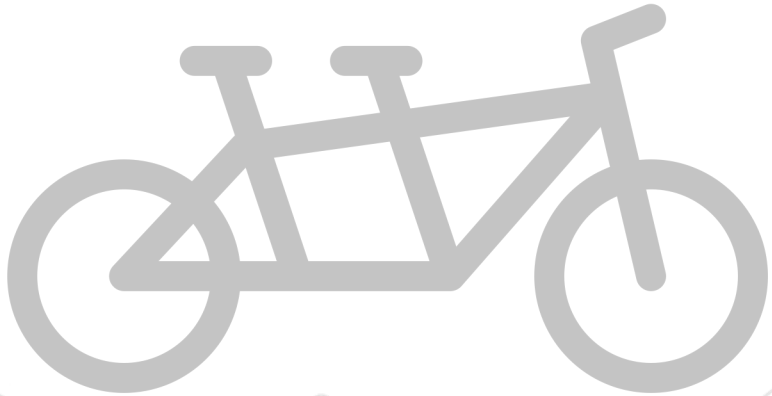
Отображение последнего известного статуса поездки

Realtime Database. Синхронизация данных о парковках

JSON-представление состояния парковки

```
{
  "parkings": {
    "ID1": {
      "latitude": 37.232,
      "longitude": 49.361,
      "address": "Sadovaya street, 122",
      "available_slots": 10,
      "available_bikes": 5,
      "parking_type": "common",
      "payment": true,
      "state": "working"
    }
  }
}
```

- Координаты
- Количество слотов / велосипедов
- Тип парковки
- Статус



Realtime Database. Синхронизация данных о парковках

Событие изменения значения одного поля объекта



Realtime Database. Синхронизация данных о парковках

```
{
  "parkings": {
    "ID1": {
      "latitude": 37.232,
      "longitude": 49.361,
      "address": "Sadovaya street, 122",
      "available_slots": 10,
      "available_bikes": 5,
      "parking_type": "common",
      "payment": true,
      "state": "working"
    }
  }
}
```

Плюсы:

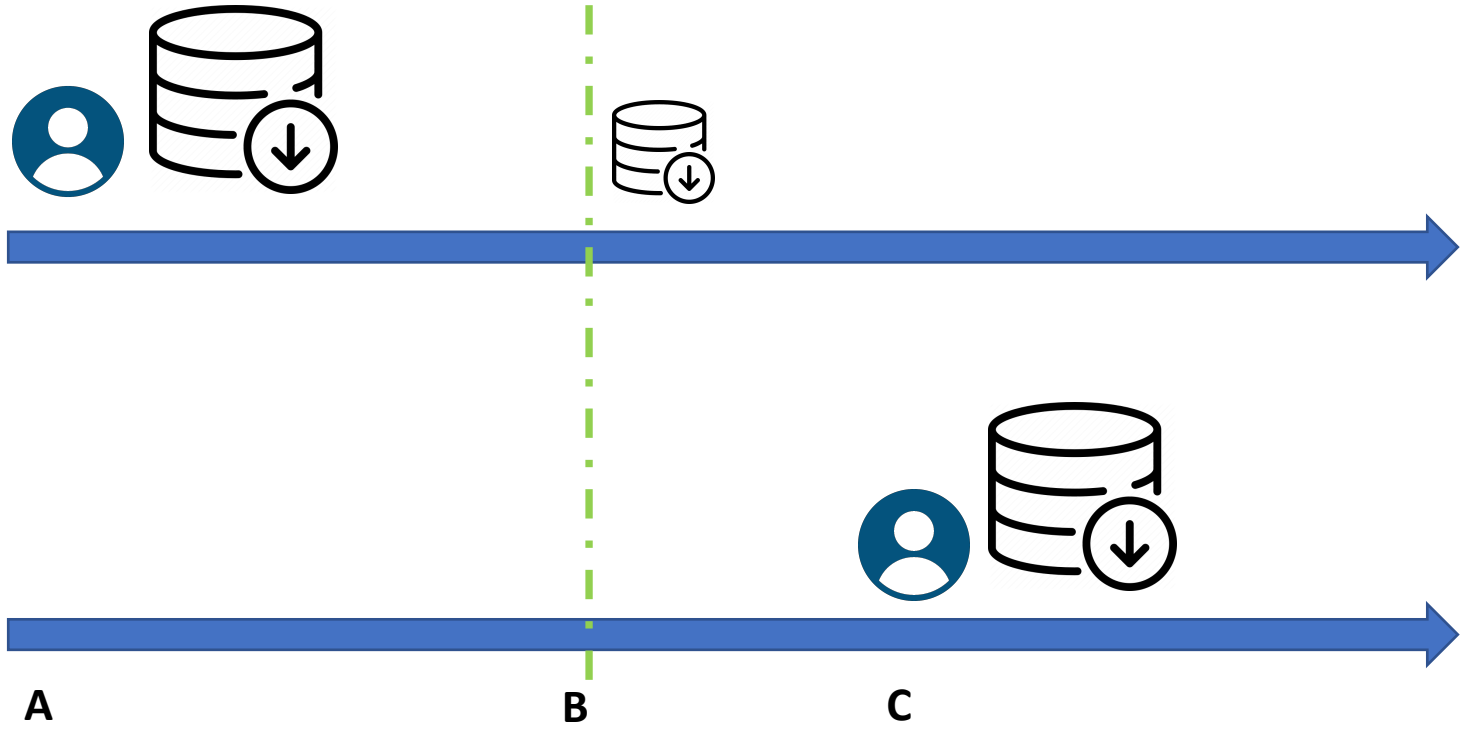
на стороне клиента получается полный снимок данных без необходимости какого-либо слияния данных

Минусы:

изменение в значении хотя бы одного поля приводит к необходимости передачи полного снимка данных

Размер полного снимка:
223 байт

Realtime Database. Объем данных, передаваемых клиенту



Realtime Database. Синхронизация данных о парковках

Подход при котором динамическая и статическая информация хранится в различных узлах

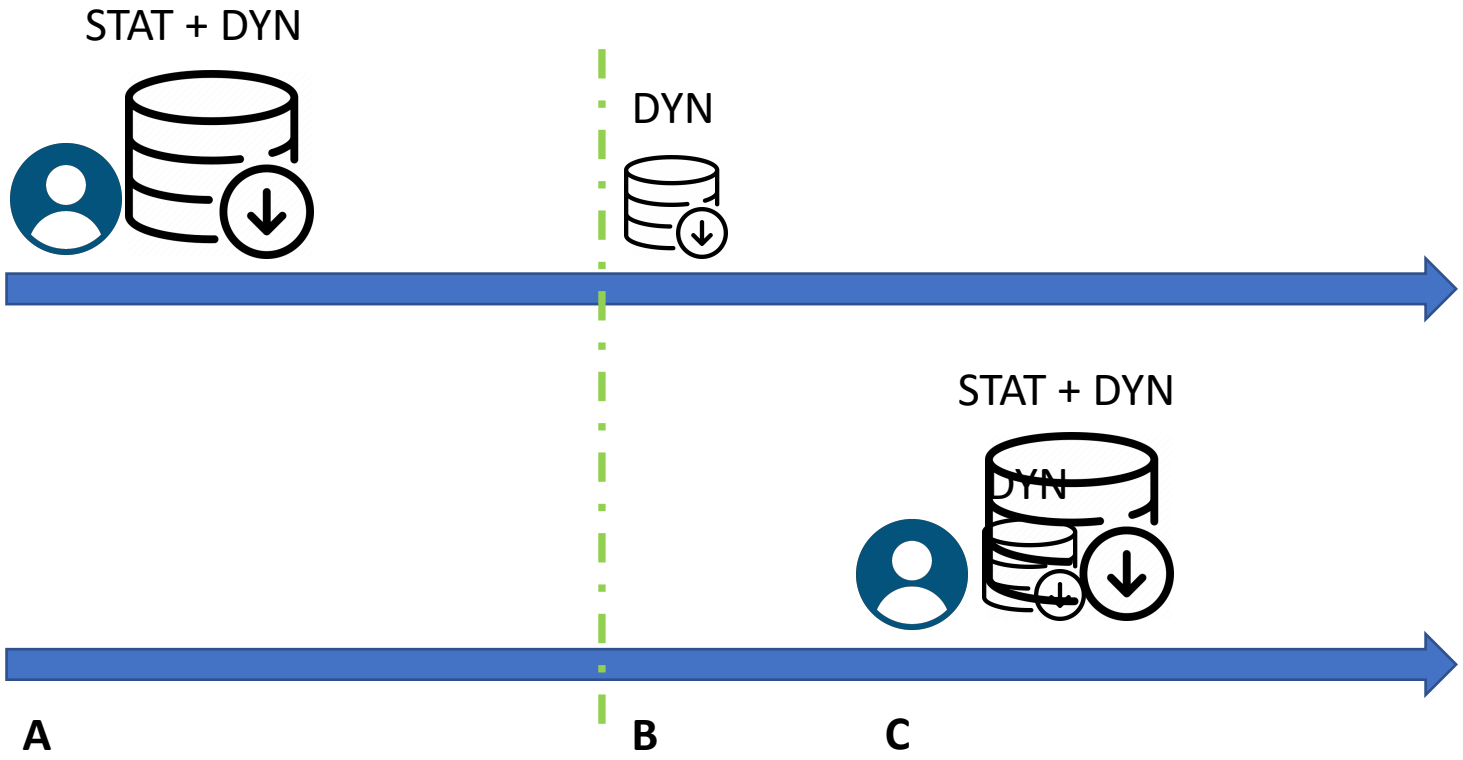
- Координаты
- Тип парковки

```
"static": {
    "ID1": {
      "latitude": 37.232,
      "longitude": 49.361,
      "address": "Sadovaya street, 122",
      "parking_type": "common",
      "payment": true
    }
  }
```

- Количество слотов / велосипедов
- Статус

```
"dynamic": {
    "ID1": {
      "available_slots": 10,
      "available_bikes": 5,
      "state": "working"
    }
  }
```

Realtime Database. Синхронизация данных о парковках



Клиент, подключившийся в момент **C** ранее уже загружал статические данные

Realtime Database. Синхронизация данных о парковках





Плюсы:

даже если с момента предыдущего «сеанса связи с Firebase» изменилась только динамическая информация, то она будет передана без «статики»

Минусы:

немного усложняется логика слияния данных на стороне клиента

Размер снимка:
«статика»: 142 байт
«динамика»: 89 байт

Realtime Database. Инструменты для профилирования трафика

Профилирование объема трафика в зависимости от раздела БД к которому обращаются клиенты

Инструмент - Firebase CLI
firebase database:profile

Downloaded Bytes

Path	Total	Count	Average
/dynamic	562.70 kB	42	13.40 kB
/static	259.85 kB	33	7.87 kB
/	9.71 kB	32	303.375 B
/	1.06 kB	35	30.285714285714285 B

Realtime Database. Инструменты для профилирования трафика

Профилирование объема трафика в зависимости от раздела БД к которому обращаются клиенты

Инструмент - Firebase CLI
firebase database:profile --raw



Детальный информация об операциях чтения/записи

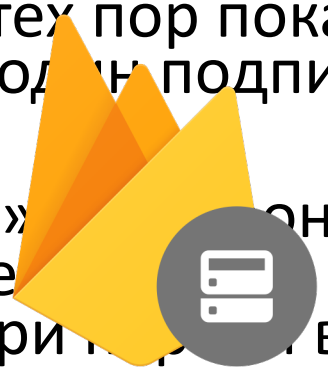
- Путь по которому производилась операция
- IP-адрес клиента
- User-Agent
- Объем переданных данных
- События подключения/отключения клиента



Realtime Database как решение задачи синхронизации данных

Итого в базовом функционале:

- Firebase SDK автоматически отслеживает изменения в сетевом подключении и выполняет повторное подключение к бэкенду Firebase Realtime Database
- Соединение с бэкендом автоматически устанавливается и поддерживается активным до тех пор пока в приложении существует хотя бы один подписчик на изменения в «снимке»
- Данные, сохраненные в «снимок» [illegible] клиента отправляются в фоновом режиме [illegible] синхронизированы с сервером при [illegible] возможности



Firebase Realtime Database

Realtime Database как решение задачи синхронизации данных





Ваши вопросы

Авторизация пользователей без бэкенда

- Способы авторизации пользователя в платформе Firebase
- Механизм предоставления доступа пользователям к заданным разделам Firebase Realtime Database



Firebase Authentication

Firebase Authentication + Firebase Realtime Database

Вынесение пользовательских данных в Firebase Realtime Database

Защита раздела БД, в котором хранятся списки избранных парковок и данные о текущей поездке



Добавление/удаление парковки из избранного возможен
в фоновом режиме благодаря Offline Capabilities

Мгновенная синхронизация «избранных» между всеми устройствами
пользователя

Firebase Authentication + Firebase Realtime Database

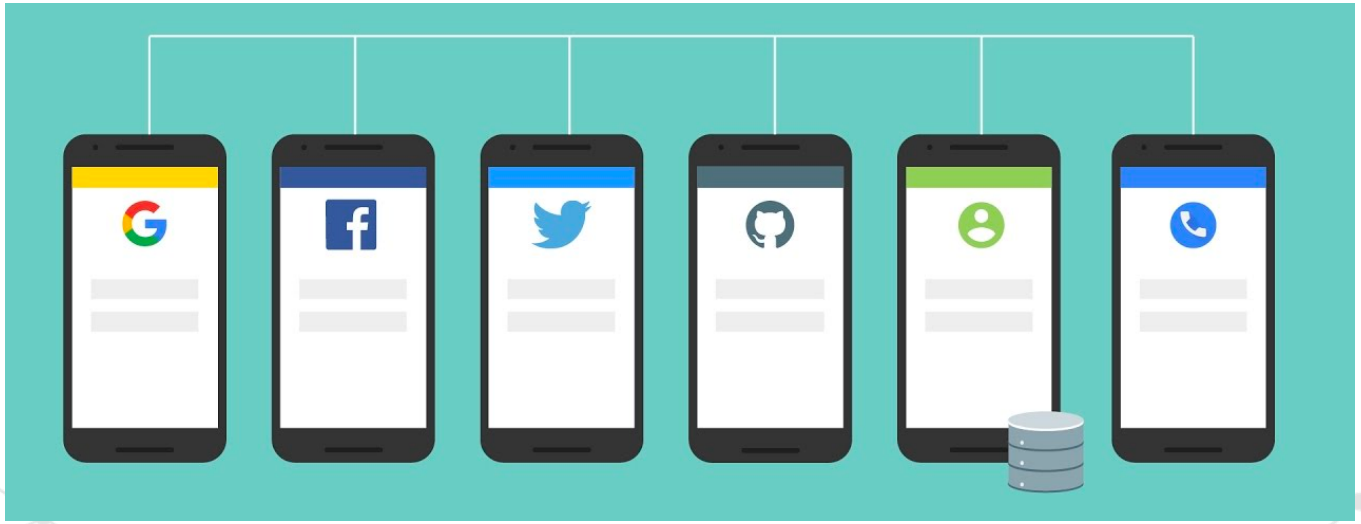
Ограничение доступа в раздел пользователя на основе авторизации
Правила описываются в виде JSON

```
{
  "rules": {
        "favourites": {    ← Название раздела к которому предоставляется доступ
              "$uid": {
                    ".read": "$uid === auth.uid",
                    ".write": "$uid === auth.uid"
                       }
               }
        }
}
```

Доступ на чтение и запись в раздел /favourites/UserId1/ доступен пользователю с UID = UserId1

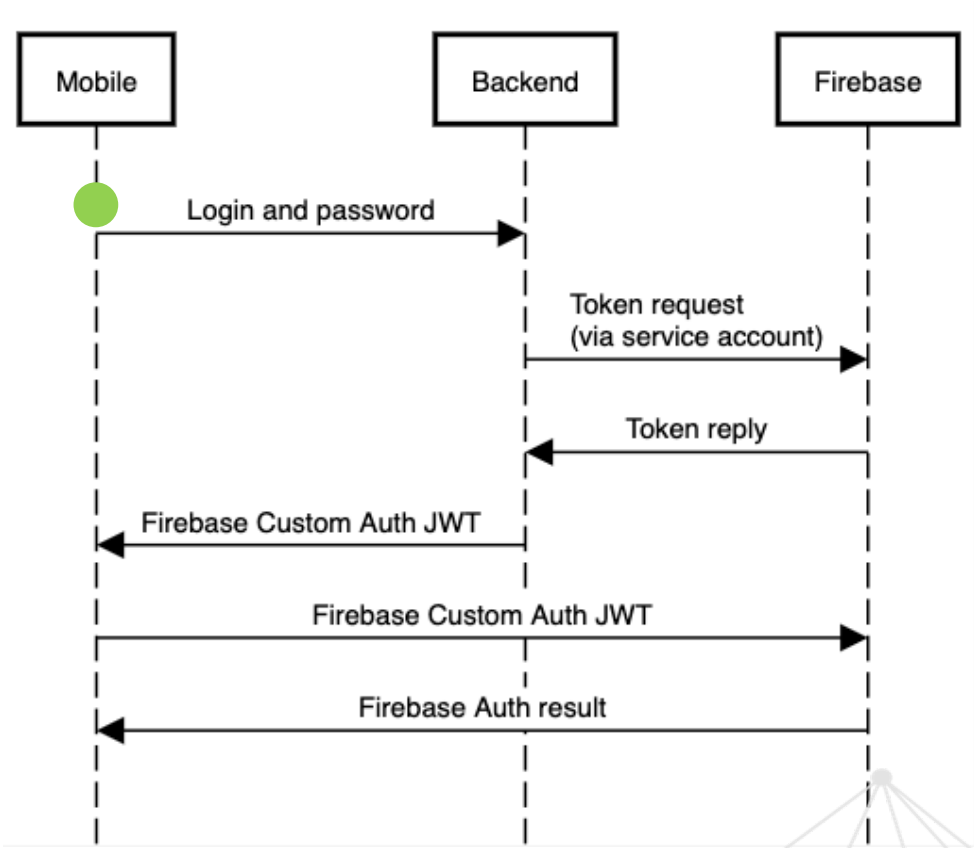
Firebase Authentication. Способы авторизации пользователя

- Email + пароль
- Социальные сети
- Номер телефона (одноразовый пароль)
- **Авторизация средствами своего бэкенда – возможность сообщить UID пользователя в нашей системе**



Firebase Authentication. Авторизация средствами своего бэкенда

- Клиент выполняет авторизацию на нашем бэкенде и получает JWT-токен
- Полученный JWT-токен мобильный клиент передает в Firebase SDK для авторизации в платформе Firebase



Firebase Authentication. Авторизация средствами своего бэкенда

Генерация JWT-токена на стороне нашего бэкенда




```
FirebaseOptions options = new FirebaseOptions.Builder()
    .setCredentials(GoogleCredentials.getApplicationDefault())
    .setServiceAccountId("my-client-id@my-project-id.iam.gserviceaccount.com")
    .build();
FirebaseApp.initializeApp(options);


String uid = "some-uid"; // Get UID for user from DB here

String customToken = FirebaseAuth.getInstance().createCustomToken(uid);
// Send token back to client
```

Firebase. Факторы влияющие на стоимость сервиса

- Трафик в сторону клиента как основная статья расходов
- Бесплатные лимиты

Authentication		
Phone Auth - US, Canada, and India	10k/month	$0.01/verification
Phone Auth - All other countries	10k/month	$0.06/verification
Other Authentication services	✓	✓

Backendless приложение на базе платформы Firebase

Построение прототипа или небольшой «backendless» системы

Firebase Realtime Database / Firebase Cloud Firestore – база данных

Firebase Authentication – аутентификация

Firebase Cloud Functions – бизнес-логика (JS/TS/Go)

Firebase Store – файловое хранилище

Firebase Cloud Messaging - PUSH-сообщения



Заключение

Применение платформы Firebase позволило нам

- Реализовать синхронизацию данных о парковках и текущей поездке в режиме реального времени
- Сократить объем сетевого трафика (в том числе за счет разделения статики и динамики)
- Снизить нагрузку на бэкенд



Заключение

Спасибо за внимание



Ваши вопросы