

Offline first apps или как не скачать себе весь Интернет на девайс

Антон Давыдов
Кошелёк & Swoo

Про что поговорим

- Что такое онлайн фёрст, примеры приложений
- Разбор кейса
- Алгоритм синхронизации объектного графа
- Синхронизация бинарных данных и управление локальным хранилищем
- Заключение

Типы приложений

- *Web-based*
- *Read only cache*
- Почти все возможности разрешены в **offline**

Типы Offline функциональности

- Редактирование пользовательских данных (заметки)
- Использование вычислительных мощностей (ML на девайсе)

Свойства Offline first apps

- Автономная работа – основная
- Синхронизация данных – по возможности
- Стабильность к перебоям сети

Автономная работа непосредственно связана с
кэшированием данных и последующей синхронизацией

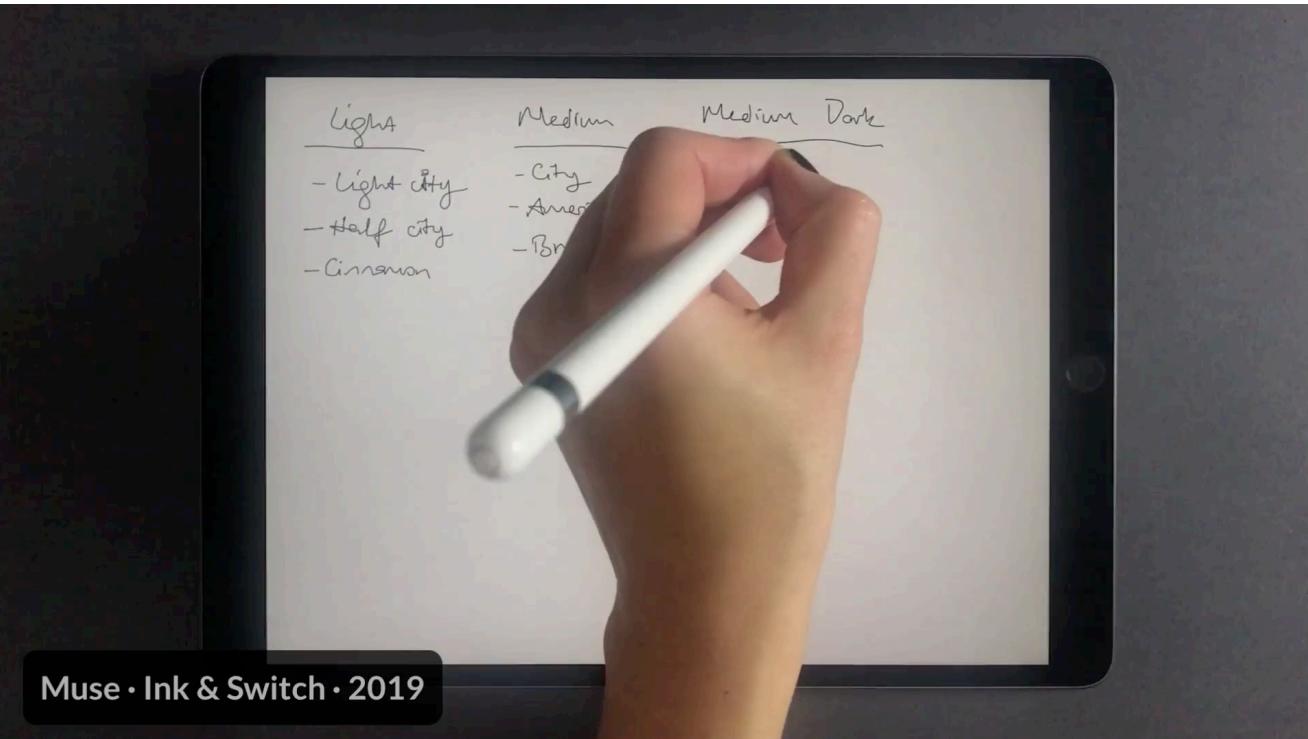
Примеры приложений

- Apple Notes
- Offline Maps
- IPassword
- Photos & Camera

Работа с документами

- Mind Node
- Облачный диск
- ...

Muse



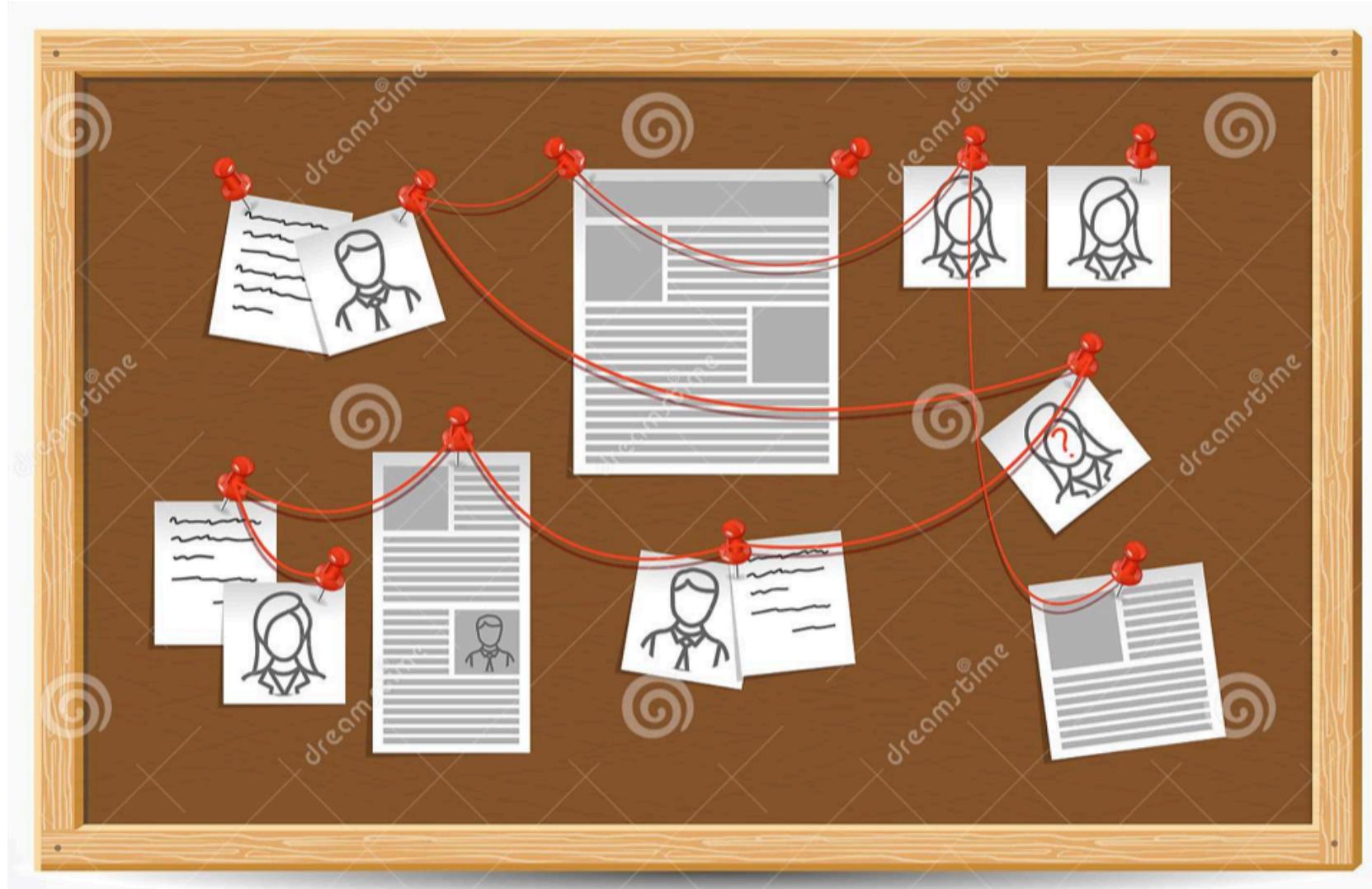
<https://www.inkandswitch.com/muse-studio-for-ideas.html>

В каких приложениях
не хватает оффайна?

Мотивация

- Больше оффлайн саппорта в приложениях!
- Больше приложений для айпада!

Разбор кейса: приложение Police Map



Составляющие части

- Борд
- Элементы: картинки, текст, документы
- Связи
- Стиль
- Позиция
- Пользователи (если будет доступен шаринг)*

Типичные условия

- Бекенд готов, нельзя изменить на API
- Реализован *read only offline*
- Есть веб-приложение
- У пользователя есть несколько мобильных устройств

Почему бэкенд обычно не изменяем

- Бэкенд *3rd party*
- Бэкенд-разработчикам “некогда” / нет ресурсов
- Текущая архитектура не позволяет переделать

Задача

Сделать **offline** редактирование с последующей
синхронизацией на базе существующего API

Текущий API

- Получить список бордов или один борд по айди
- Обновить/добавить элемент на борде

API

```
GET /boards [Board]
POST/GET/PUT /board/<id> Board
POST/DELETE/PUT /board/<id>/items/<id>

Board
{
    "id": string,
    "name": string,
    "dateUpdated": string,
    "items": [Item],
    "links": [string]
}

Item
{
    "id": string,
    "type": int,

    // text
    "text": string,

    // image, doc
    "url": string
}
```

Данные

- Граф объектов (json)
- Бинарные (фото, доки)

Ситуация

Скачали данные и интернет пропал

Наивное решение

При изменение данных в оффлайн помечать сущности

`var isSynced: Bool = false`

При появлении интернета как-то их загружать

Минусы

Не ясно какие конкретно данные обновлены

Рекомендация

- Разделить данные и информацию о синхронизации
- Разбить изменения на атомарные операции/команды

Операция -

Декларация того, что сделать и с какими параметрами

Операция

- Хранится локально
- Хранится отдельно от основного хранилища
- Содержит минимально необходимую информацию

Пример

Операция добавления

```
struct AddItemOperation {  
    let id: String  
    let boardId: String  
    let type: Int  
  
    ...  
}
```

Пример

Операция удаления

```
struct RemoveItemOperation {  
    let id: String  
}
```

Лог операций

[AddOp, UpdOp, UpdOp, RemOp]

Отображение

```
let addOp = AddItemOperation(id: "id", type: 0)
```

```
POST /board/<id>/items/
```

```
{
```

```
  "id": string,
```

```
  "type": 0
```

```
}
```

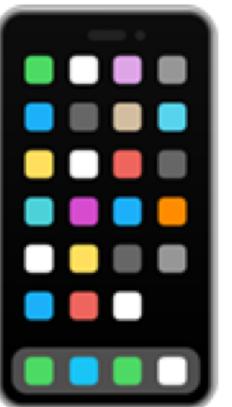
Алгоритм редактирования

Алгоритм в текстовом виде

- Пользователь что-то создает, и создается новая операция
- Операция сохраняется на диск
- Операция “накатывается” на локальное хранилище
- Операция пытается зааплоадиться
- Удаляется из локального хранилища при успехе
- При ошибке попытается зааплоадиться в следующий

Алгоритм

Новый пользователь появляется в сети



Алгоритм

Пользователь скачал данные



Алгоритм

Интернет отключили



Алгоритм

Пользователь начинает редактировать



Алгоритм

Операции сохраняются и обновляют базу



Алгоритм

Появляется интернет



Алгоритм

Операции улетают на сервер



Алгоритм

Операции улетают на сервер



Алгоритм

Клиент и сервер синхронизованы



Обработчики операций

Операции – это только декларация. Обработчики:

`LocalOperationHandler`

`ServerOperationHandler`

```
class LocalOperationHandler {  
    func handle(op: Operation) throws {  
        switch op {  
            case let addOp as AddItemOp:  
                let context = container.newBackgroundContext()  
                let newItem = Item(context: context)  
                newItem.id = addOp.id  
                newItem.type = addOp.type  
                try context.save()  
            default:  
                break  
        }  
    }  
}
```

```
class ServerOperationHandler {
    func handle(op: Operation, completion: @escaping (Result<Void, Error>) → Void) {
        switch op {
            case let addOp as AddItemOp:
                let json = dump(addOp)
                let request = URLRequest( ... )
                URLSession.shared.dataTask(with: request) { result in
                    if case .success = result {
                        remove(op: addOp)
                    }
                    completion(result)
                }
            default:
                break
        }
    }
}
```

Свойства обработки операций

- Атомарные
- Хранятся пока данные не улетели на сервере
- Представляют из себя список
- Должны применяться последовательно

Операции из нескольких шагов*

- Загрузить файл и получить URL
- Создать сущность с URL

Исключения. С какими данными не поработать онлайн

- Выдача доступов
- Настройки

Что делать, если пока
были в оффлайне, кто-
то нашкодил на
сервере?

Скачиваем актуальный
граф с бекенда

Варианты определения, что изменилось

- 1) Умеем определять дифф
- 2) Не умеем

Варианты скачивания

- До отправки операций
- После отправки операций

Не умеем определять дифф

- Скачиваем сначала весь список
- Удаляем текущий локальный
- Накатываем локальные команды
- Аплоадим локальные команды и удаляем их

Умеем определять дифф

- Апложим команды и удаляем их
- Скачиваем список с сервера
- Определяем дифф
- Накатываем его

Важно прийти к
консистентности
данных на клиенте и
сервере

LWW - Last-Write-Wins



- Решение конфликтов с помощью – *Overwrite*
- Важен порядок применения на клиенте и сервере

LWW. Минусы

Возможна потеря пользовательских данных

В какой момент начать
синхронизацию?

Как определить интернет на девайсе?

```
//declare this property where it won't go out of scope relative to your listener
let reachability = try! Reachability()

reachability.whenReachable = { reachability in
    if reachability.connection == .wifi {
        print("Reachable via WiFi")
    } else {
        print("Reachable via Cellular")
    }
}
reachability.whenUnreachable = { _ in
    print("Not reachable")
}

do {
    try reachability.startNotifier()
} catch {
    print("Unable to start notifier")
}
```

<https://github.com/ashleymills/Reachability.swift>

Как реагировать на изменение состояния интернета

- При появлении интернета начать синхронизацию
- При пропадании – остановить процесс или пауза и *wait-retry*

Капитанство

Если приложение определило, что интернет есть, то это значит, что он был мгновение назад. Через мгновение его уже может не быть 🤝🐺

Улучшаем процесс синхронизации

1. Настройка URLCache

E-Tag

Скачиваем только, если что-то изменилось на бекенде

https://ru.wikipedia.org/wiki/HTTP_ETag

2. Унифицировать идентификаторы на клиенте и сервере

Обычный подход: `localId` и `serverId`

Как хотелось бы: `id = UUID().uuidString`

3. Научить бекенд присыпать только диффы

- Список операций
- Только сущности, которые изменились

Версионирование

- дата последнего скачивания в параметрах
- какая-то ревизия с сервера

Что потребуется дополнительно

```
{  
  "id": string,  
  "isDeleted": true  
}
```

4. Идемпотентность

$$f(f(x)) = f(x)$$

Свойство объекта или операции при повторном применении операции к объекту давать тот же результат, что и при первом

Идемпотентность - не выбрасывать ошибку, если

- Удаляешь удаленное
- Добавляешь добавленное

Прочее

- Возможность сделать `undo`
- Параллельность загрузки
- Отправлять операции пачками

Обновление бинарных данных / файлов (фотки, картинки)

Полный алгоритм синхронизации

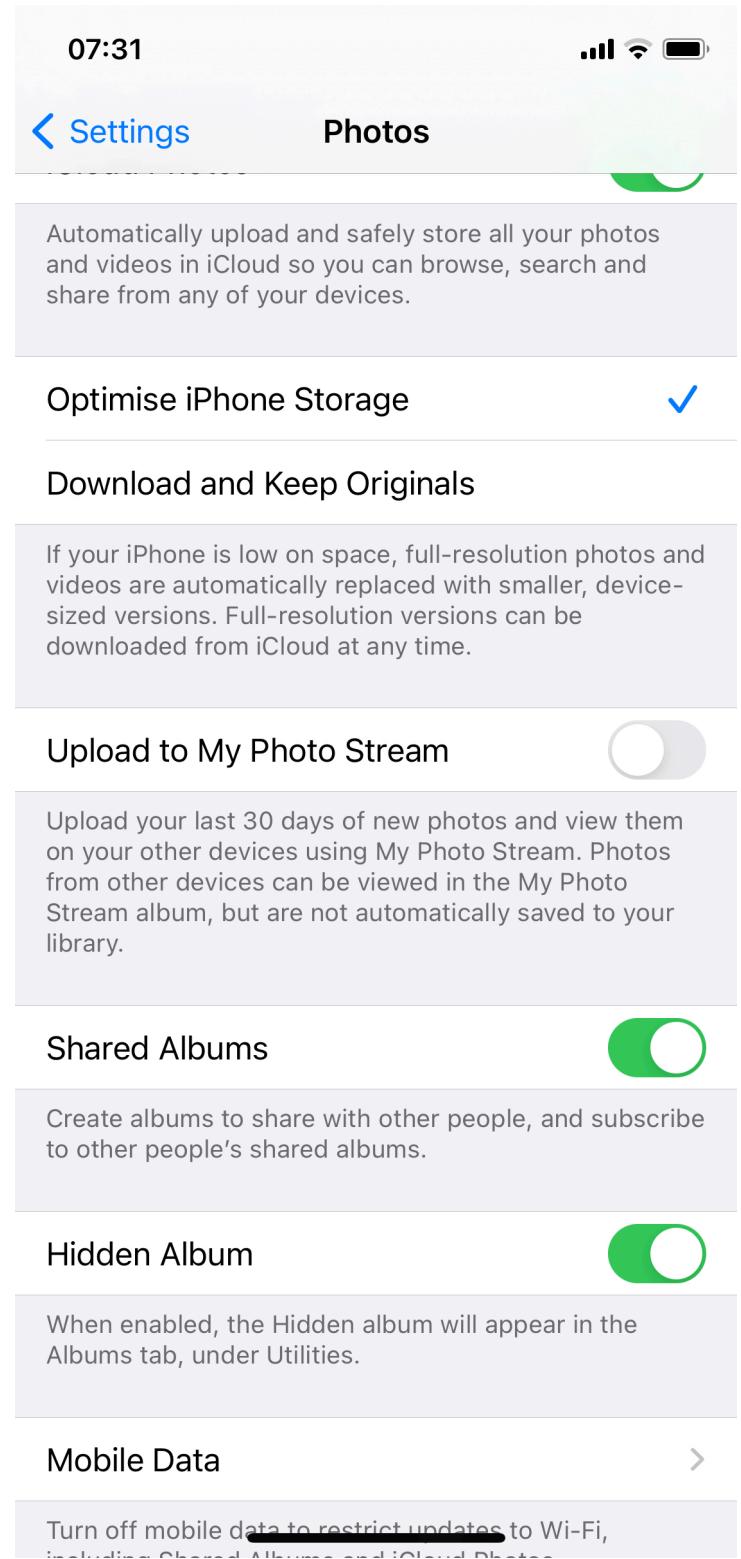
- Скачиваем обновление графа
- Апложим, что сделали офлайн
- Скачиваем бинарные данные / файлы

Обновление бинарных данных

- Бэкграунд процесс
- Параллелится

Варианты обновления

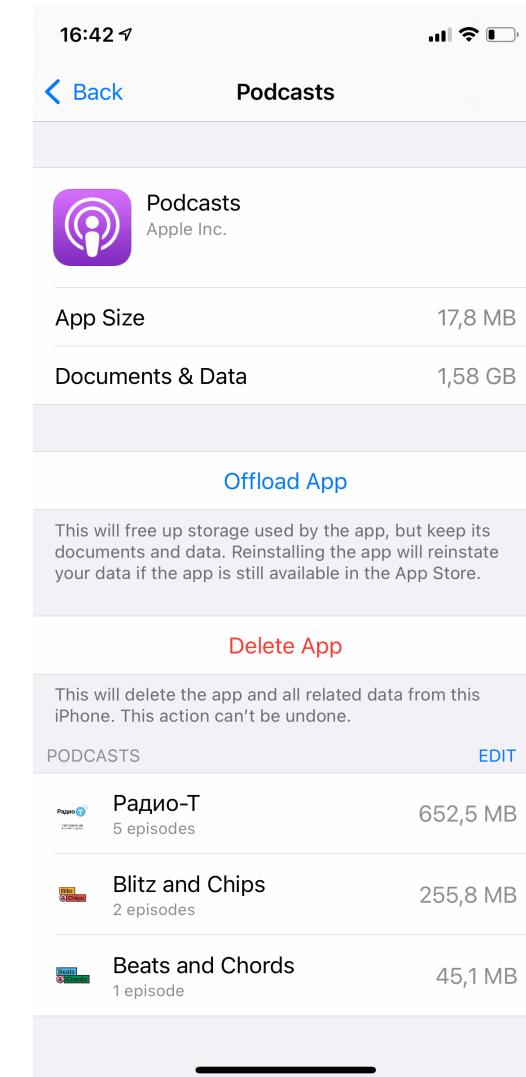
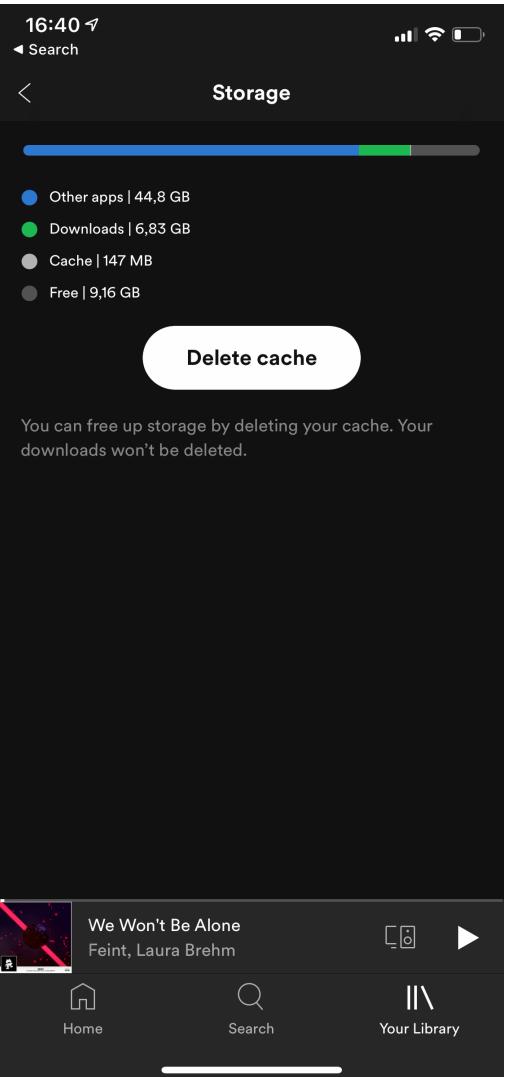
- Скачивать и обновлять все безусловно
- Давать пользователю решать (*Coursera*)
- “Умный” алгоритм (*Photos*)



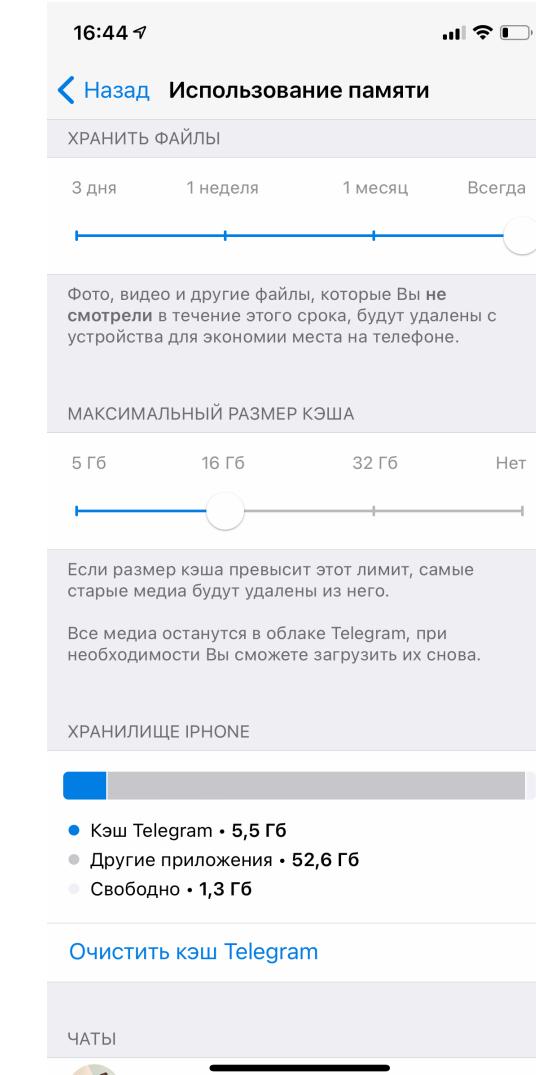
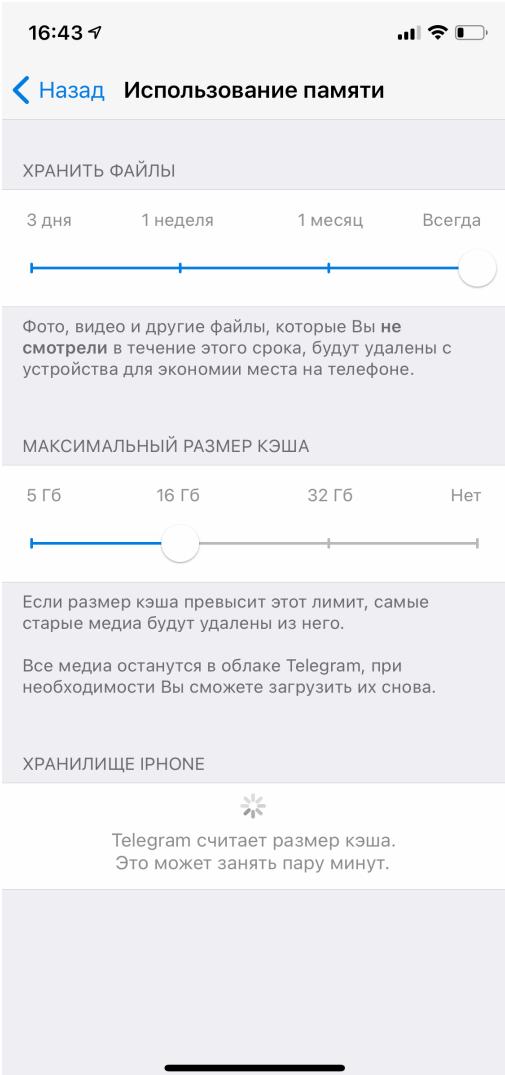
Storage management

- Размер диска iPhone небезграничен
- Пользователь хочет управлять и удалять лишнее

Примеры UI по управлению хранилищем



Примеры UI по управлению хранилищем



Три варианта

- Не думать о свободной и занимаемой памяти
- Включать пользователя в процесс контроля
- Смотреть на количество свободной и обрабатывать

Кейс из опыта - умный алгоритм

- Многопользовательность
- Скачивать легковесные файлы безусловно (текстовые)
- Слоумо видео не скачивать, только стриммить
- Скачивать фотографию и видео при заходе в нее

Кейс из опыта - умный алгоритм

#2

- Размер одного пользователя – не больше 2 ГБ
- LRU кеш
- Переход в *read only* режим
- Уметь очищать кэш

Как узнать сколько свободного места на девайсе?

```
let systemAttributes = try? FileManager.default.attributesOfFileSystem(forPath: path)
let freeSize = systemAttributes?[FileAttributeKey.systemFreeSize] as? NSNumber
let result = freeSize?.uint64Value
```

Как узнать сколько памяти занимает песочница?

```
- (BOOL)getBytesAllocatedSize:(unsigned long long *)size forDirectoryAtURL:(NSURL *)directoryURL error:(NSError * __autoreleasing *)error
{
    NSParameterAssert(size != NULL);
    NSParameterAssert(directoryURL != nil);

    // We'll sum up content size here
    unsigned long long accumulatedSize = 0;

    // prefetching some properties during traversal will speed up things a bit.
    NSArray *prefetchedProperties = @{
        NSURLIsRegularFileKey,
        NSURLIsLocalizedNameKey,
        NSURLTotalFileAllocatedSizeKey,
        NSURLTotalFileAllocatedSizeKey,
    };

    // The error handler simply signals errors to outside code.
    _block BOOL errorDidOccur = NO;
    BOOL (*errorHandler)(NSURL *, NSError *) = ^(NSURL *url, NSError *localError) {
        if (error == nil) {
            error = localError;
            errorDidOccur = YES;
        }
        return NO;
    };

    // We have to enumerate all directory contents, including subdirectories.
    NSDirectoryEnumerator *enumerator = [[NSFileManager defaultManager] enumeratorAtURL:directoryURL
        includingPropertiesForKeys:@[prefetchedProperties
            options:(NSURLEnumerationOptions)kNSURLEnumerationOptions]
        errorHandler:errorHandler];

    // Start the traversal.
    for (NSURL <@>contentItemURL in enumerator) {
        // Bail out on errors from the errorHandler.
        if (errorDidOccur)
            return NO;

        // Get the type of this item, making sure we only sum up sizes of regular files.
        NSNumber *isRegularFile;
        if ([contentItemURL getResourceValue:&isRegularFile forKey:kNSURLIsRegularFileKey error:error])
            return NO;
        if (!isRegularFile.boolValue)
            continue; // Ignore anything except regular files.

        // To get the file's size we first try the most comprehensive value in terms of what the file may use on disk.
        // This includes metadata, compression (on file system level) and block size.
        NSNumber *fileSize;
        if ([contentItemURL getResourceValue:&fileSize forKey:kNSURLTotalFileSizeAllocatedSizeKey error:error])
            return NO;

        // In case the value is unavailable we use the fallback value (excluding meta data and compression)
        // The value should always be available.
        if (fileSize == nil) {
            if ([contentItemURL getResourceValue:&fileSize forKey:kNSURLFileAllocatedSizeKey error:error])
                return NO;

            NSAssert(fileSize != nil, @"Whoah? NSURLFileAllocatedSizeKey should always return a value");
        }

        // We're good, add up the value.
        accumulatedSize += [fileSize unsignedLongLongValue];
    }

    // Bail out on errors from the errorHandler.
    if (errorDidOccur)
        return NO;

    // We finally got it!
    *size = accumulatedSize;
    return YES;
}
```

<https://stackoverflow.com/questions/2188469/how-can-i-calculate-the-size-of-a-folder/16139991>

Минусы

Если файлов очень много, то подсчет занимает время

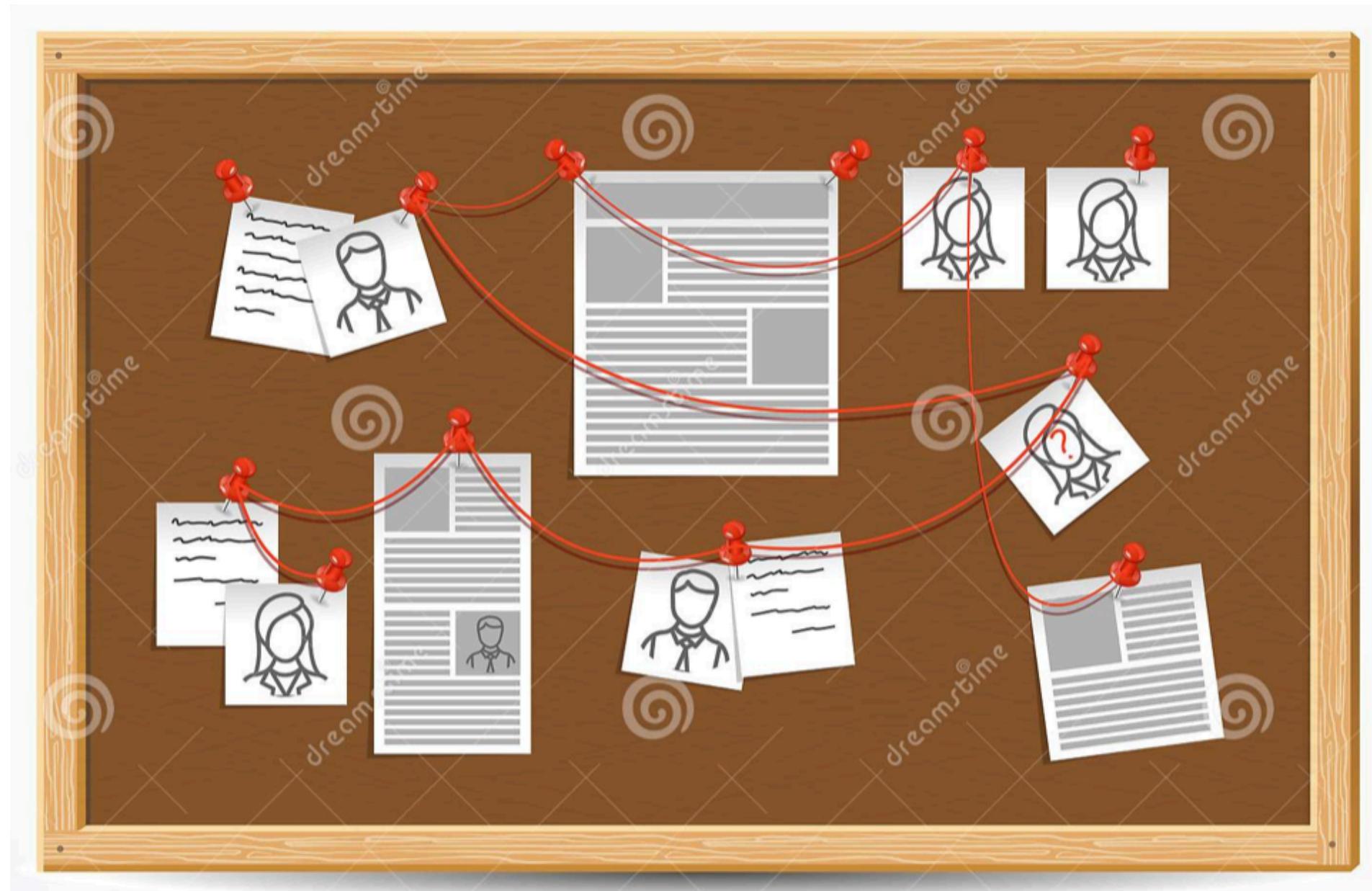
Альтернатива

Считать размер при записи / удалении файлов

Что делать, если места на девайсе осталось мало?

- Запрещать редактирование
- Объяснить пользователю

✓ Задача выполнена



Суммируем

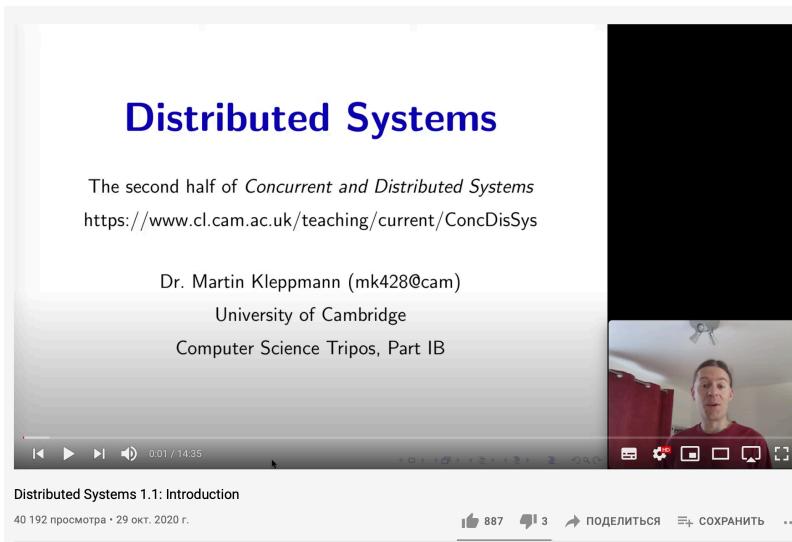
- Разработали алгоритм синхронизации на базе существующего API
- Обсудили стратегии скачивания тяжелых данных
- Научились управлять пользовательским хранилищем

Что дальше?

- Изменения на UI (*Optimistic UI*)
- Синхронизация без конфликтов и потерь данных
- *Realtime collaboration*

Материалы

Distributed Systems Lectures



[https://www.youtube.com/watch?
v=UEAMfLPZZhE&list=PLeKd45zvjcDFUEvoahrHdUFeq
7RItdiB](https://www.youtube.com/watch?v=UEAMfLPZZhE&list=PLeKd45zvjcDFUEvoahrHdUFeq7RItdiB)

Local First Software

Local-first software

You own your data, in spite of the cloud

Cloud apps like Google Docs and Trello are popular because they enable real-time collaboration with colleagues, and they make it easy for us to access our work from all of our devices. However, by centralizing data storage on servers, cloud apps also take away ownership and agency from users. If a service shuts down, the software stops functioning, and data created with that software is lost.

In this article we propose “local-first software”: a set of principles for software that enables both collaboration *and* ownership for users. Local-first ideals include the ability to work offline and collaborate across multiple devices, while also improving the security, privacy, long-term preservation, and user control of data.

We survey existing approaches to data storage and sharing, ranging from email attachments to web apps to Firebase-backed mobile apps, and we examine the trade-offs of each. We look at Conflict-free Replicated Data Types (CRDTs): data structures that are multi-user from the ground up while also being fundamentally local and private. CRDTs have the potential to be a foundational technology for realizing local-first software.

Ink & Switch

[Martin Kleppmann](#)

[Adam Wiggins](#)

[Peter van Hardenberg](#)

[Mark McGranaghan](#)

April 2019

<https://www.inkandswitch.com/local-first.html>

Домашнее задание

Реализовать один из вариаций рассмотренного алгоритма

Q&A



tg @dydusOxI4

tg канал https://t.me/km_engineering