# try! Realm Workshop

## try! Swift Tokyo 2017



kk@realm.io

# Katsumi Kishikawa

Realm Inc.




kk@realm.io

# #realm_jp



kk@realm.io

**WiFi:** `try_Swift_Tokyo`
**Password:** `tryswifttokyo`



kk@realm.io

# What's Realm



kk@realm.io

# What's Realm

- モバイルデバイス専用に新しく作られたデータベース
- モバイル用途に最適化されている
  - 遅延ロード
  - ゼロコピー・アーキテクチャ
  - ネイティブ・リンク
- 独自エンジン（SQLiteを使っていない）
- オープンソース


jp@realm.io

# What's Realm

- 高速な動作
- 暗号化を標準装備
- マルチプラットフォーム
- 直感的でシンプルなAPI
- 丁寧なサポート（日本語OK）



jp@realm.io

# Multi platform

- Realm Objective-C/Swift
- Realm Java
- Realm React NativeRealm
- Realm Xamarin



jp@realm.io

# Model definition

```
class Article: Object {
    dynamic var id: Int = 0
    dynamic var title: String = ""
    dynamic var content: String = ""
    dynamic var creationDate: NSDate = NSDate()
    dynamic var deletionDate: NSDate? = nil
    dynamic var user: User? = nil
    let comments = List<Comment>()
}
```

Objectのサブクラス



kk@realm.io

# Model definition

```
class Article: Object {
    dynamic var id: Int = 0
    dynamic var title: String = ""
    dynamic var content: String = ""
    dynamic var c                    NSDate()
    dynamic var                      = nil
    dynamic var user: User? = nil
    let comments = List<Comment>()
}
```

Objectのサブクラス

（基本的に）dynamic var



kk@realm.io

# Model definition

```
class Article: Object {
    dynamic var id: Int = 0
    dynamic var title: String = ""
    dynamic var content: String = ""
    dynamic var creationDate: NSDate = NSDate()
    dynamic var deletionDate: NSDate? = nil
    dynamic var user: User? = nil
    let comments = List<Comment>()
}
```

例外: Genericsのため



kk@realm.io

# Model definition

```
class Article: Object {
    dynamic var id: Int = 0
    dynamic var title: String = ""
    dynamic var content: String = ""
    dynamic var creationDate: NSDate = NSDate()
    dynamic var deletionDate: NSDate? = nil
    dynamic var user: User? = nil
    let comments = List<Comment>()
}
```

クラス=テーブル

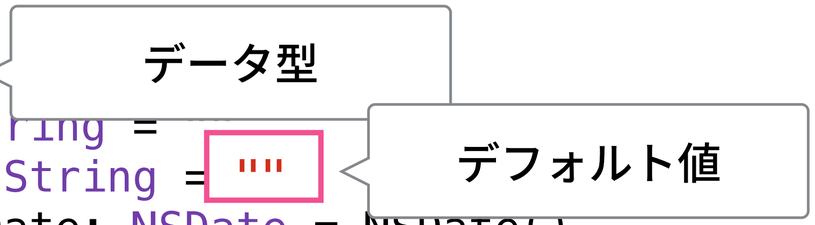
プロパティ=カラム



kk@realm.io

# Model definition

```
class Article: Object {
    dynamic var id: Int =
    dynamic var title: String =
    dynamic var content: String = ""
    dynamic var creationDate: NSDate = NSDate()
    dynamic var deletionDate: NSDate? = nil
    dynamic var user: User? = nil
    let comments = List<Comment>()
}
```






kk@realm.io

# Model definition

```
class Article: Object {
    dynamic var id: Int = 0
    dynamic var title: String = ""
    dynamic var content: String = ""
    dynamic var creationDate: NSDate = NSDate()
    dynamic var deletionDate: NSDate
    dynamic var user: User? = nil
    let comments = List<Comment>()
}
```

１対１の関連



kk@realm.io

# Model definition

```
class Article: Object {
    dynamic var id: Int = 0
    dynamic var title: String = ""
    dynamic var content: String = ""
    dynamic var creationDate: NSDate = NSDate()
    dynamic var deletionDate: NSDate? = nil
    dynamic var user: User? = nil
    let comments = List<Comment>()
}
```

１対多の関連



kk@realm.io

# Model definition

```
class Article: Object {
    dynamic var id: Int = 0
    dynamic var title: String = ""
    dynamic var content: String = ""
    dynamic var creationDate: NSDate = NSDate()
    dynamic var modificationDate: NSDate = NSDate()
    dynamic var deletionDate: NSDate? = nil
    dynamic var user: User? = nil
    let comments = List<Comment>()

    override class func primaryKey() -> String? {
        return "id"
    }
}
```

プライマリキー



kk@realm.io

# Save Object

```
let realm = try! Realm()

let article = Article()
article.title = titleTextfield.text
article.contents = contentTextfield.text

try! realm.write { in
    realm.add(article)
}
```



kk@realm.io

# Save Object

```
let realm = try! Realm()

let article = Article()
article.title = titleTextfield.text
article.contents = contentTextfield.text

try! realm.write { in
    realm.add(article)
}
```



kk@realm.io

# Save Object

```
let realm = try! Realm()

let article = Article()
article.title = titleTextfield.text
article.contents = contentTextfield.text

try! realm.write { in
    realm.add(article)
}
```



kk@realm.io

# Save Object

```
let realm = try! Realm()

let article = Article()
article.title = titleTextfield.text
article.contents = contentTextfield.text

realm.write { () in
    realm.add(article)
}
```



kk@realm.io

# Save Object

```
let realm = try! Realm()

let article = Article()
article.title = titleTextfield.text
article.contents = contentTextfield.text

let currentUser = realm.object(ofType: User.self,
                               forPrimaryKey: userID)
article.user = currentUser

try! realm.write { in
    realm.add(article)
}
```



kk@realm.io

# Query

```
let realm = try! Realm()

let currentUser = realm.object(ofType: User.self,
                               forPrimaryKey: userID)
let articles = realm
                    .objects(Article)
                    .filter("user = %@", currentUser)

let article = articles[indexPath.row]

cell.titleTextLabel.text = article.title
cell.contentTextLabel.text = article.contents
cell.userNameLabel.text = article.user.name
```



kk@realm.io

# Query

```
let realm = try! Realm()

let currentUser = realm.object(ofType: User.self,
                               forPrimaryKey: userID)
let articles = realm
                    .objects(Article)
                    .filter("user = %@", currentUser)

let article = articles[indexPath.row]

cell.titleTextLabel.text = article.title
cell.contentTextLabel.text = article.contents
cell.userNameLabel.text = article.user.name
```



kk@realm.io

# Query

```
let realm = try! Realm()

let currentUser = realm.object(ofType: User.self,
                               forPrimaryKey: userID)
let articles = realm
                    .objects(Article.self)
                    .filter("user = %@", currentUser)

let article = articles[indexPath.row]

cell.titleTextLabel.text = article.title
cell.contentTextLabel.text = article.contents
cell.userNameLabel.text = article.user.name
```



kk@realm.io

# Query

```
let realm = try! Realm()

let currentUser = realm.object(ofType: User.self,
                               forPrimaryKey: userID)
let articles = realm
                    .objects(Article.self)
                    .filter("user = %@", currentUser)

let article = articles[indexPath.row]

cell.titleTextLabel.text = article.title
cell.contentTextLabel.text = article.contents
cell.userNameLabel.text = article.user.name
```



kk@realm.io

# Query

```
let realm = try! Realm()

let currentUser = realm.object(ofType: User.self,
                               forPrimaryKey: userID)
let articles: Results<Article> = realm
                                    .objects(Article.self)
                                    .filter("user = %@", currentUser)

let article = articles[indexPath.row]

cell.titleTextLabel.text = article.title
cell.contentTextLabel.text = article.contents
cell.userNameLabel.text = article.user.name
```



kk@realm.io

# RealmConfiguration

```
let config = Realm.Configuration(fileURL: URL(fileURLWithPath: "..."),
                                 encryptionKey: key,
                                 deleteRealmIfMigrationNeeded: true,
                                 objectTypes: [Article.self, User.self, Comment.self])
let realm = try! Realm(configuration: config)
```



kk@realm.io

# RealmConfiguration

```
let config = Realm.Configuration(fileURL: URL(fileURLWithPath: "..
                                 encryptionKey: key,
                                 deleteRealmIfMigrationNeeded: true,
                                 objectTypes: [Article.self, User.self, Comment.self])
let realm = try! Realm(configuration: config)
```

保存場所／ファイル名



kk@realm.io

# RealmConfiguration

```
let config = Realm.Configuration(fileURL: URL(fileURLWit
                                 encryptionKey: key,
                                 deleteRealmIfMigrationN
                                 objectTypes: [Article.self, User.self, Comment.self])
let realm = try! Realm(configuration: config)
```

暗号化



kk@realm.io

# RealmConfiguration

```
let config = Realm.Configuration(fileURL: URL(fileURLWithPath: "..."),
                                 encryptionKey: key,
                                 deleteRealmIfMigrationNeeded: true,
                                 objectTypes: [Article.se
let realm = try! Realm(configuration: config)
```

マイグレーション時にファイル削除



kk@realm.io

# RealmConfiguration

```
let config = Realm.Configuration(fileURL: URL(fileURLWithPath: "..."),
                                 encryptionKey: key,
                                 deleteRealmIfMigrationNeeded: true,
                                 objectTypes: [Article.self, User.self, Comment.self])
let realm = try! Realm(configuration: config)
```

このファイルで使用するテーブルを限定



kk@realm.io

# RealmConfiguration

```
let config = Realm.Configuration(fileURL: URL(fileURLWithPath: "..."),
                                 encryptionKey: key,
                                 deleteRealmIfMigrationNeeded: true,
                                 objectTypes: [Article.self, User.self, Comment.self])
Realm.Configuration.defaultConfiguration = config
let realm = try! Realm()
```



kk@realm.io

# RealmConfiguration

```
let config = Realm.Configuration(fileURL: URL(fileURLWithPath: "..."),
                                 encryptionKey: key,
                                 deleteRealmIfMigrationNeeded: true,
                                 objectTypes: [Article.self, User.self, Comment.self])
Realm.Configuration.defaultConfiguration = config
let realm = try! Realm()
```



kk@realm.io

# Multi thread

```
func updateUnread(id: String) {
    let realm = try! Realm()
    guard let tweet = let currentUser = realm.object(ofType: User.self,
                                                     forPrimaryKey: userID) else {
        return
    }

    request.performRequestWithHandler { (data, response, error) in
        if let error = error {
            return
        }

        ...

        try! realm.write {
            tweet.unread = false
        }
    }
}
```



kk@realm.io

# Multi thread

```
func updateUnread(id: String) {
    let realm = try! Realm()
    guard let tweet = let currentUser = realm.object(ofType: User.self,
                                                     forPrimaryKey: userID) else {
        return
    }

    request.performRequestWithHandler { (data, response, error) in
        if let error = error {
            return
        }

        ...

        try! realm.write {
            tweet.unread = false
        }
    }
}
```



kk@realm.io

# Multi thread

```
func updateUnread(id: String) {
    let realm = try! Realm()
    guard let tweet = let currentUser = realm.object(ofType: User.self,
                                                     forPrimaryKey: userID) else {
        return
    }

    request.performRequestWithHandler { (data, response, error) -> Void in
        if let error = error {
            return
        }

        let realm = try! Realm()
        if let tweet = let currentUser = realm.object(ofType: User.self,
                                                      forPrimaryKey: userID) {
            try! realm.write {
                tweet.favorited = !tweet.favorited
            }
        }
    }
}
```



kk@realm.io

# Multi thread

```
func updateUnread(id: String) {
    let realm = try! Realm()
    guard let tweet = let currentUser = realm.object(ofType: User.self,
                                                     forPrimaryKey: userID) else {
        return
    }

    request.performRequestWithHandler { (data, response, error) -> Void in
        if let error = error {
            return
        }

        let realm = try! Realm()
        if let tweet = let currentUser = realm.object(ofType: User.self,
                                                      forPrimaryKey: userID) {
            try! realm.write {
                tweet.favorited = !tweet.favorited
            }
        }
    }
}
```



kk@realm.io

# What is
# Realm Mobile Platform?



kk@realm.io

# Realm

- SQLiteの代替となるモバイルデバイス専用のデータベース
- クロスプラットフォーム
  - iOS/Android
  - Java, Swift, Objective-C, C#, JavaScript
- オープンソース


kk@realm.io

# Realm Mobile Database

- SQLiteの代替となるモバイルデバイス専用のデータベース
- クロスプラットフォーム
  - iOS/Android
  - Java, Swift, Objective-C, C#, JavaScript
- オープンソース


kk@realm.io

# Realm Mobile Platform



kk@realm.io

# Realm Mobile Platform

## Realm Mobile Database



kk@realm.io

# Realm Mobile Platform

## Data Sync

リアルタイム同期

## Realm Mobile Database



kk@realm.io

# Realm Mobile Platform

## Realm Mobile Database

### Data Sync

リアルタイム同期

### User Identify

ユーザー識別

realm

kk@realm.io

# Realm Mobile Platform

## Realm Mobile Database

## Data Sync

リアルタイム同期

## Access Control

データ共有

## User Identify

ユーザー識別



kk@realm.io

# Realm Mobile Platform

## Realm Mobile Database

### Data Sync

リアルタイム同期

### Access Control

データ共有

### User Identify

ユーザー識別

### Event Handling

ビジネスロジック

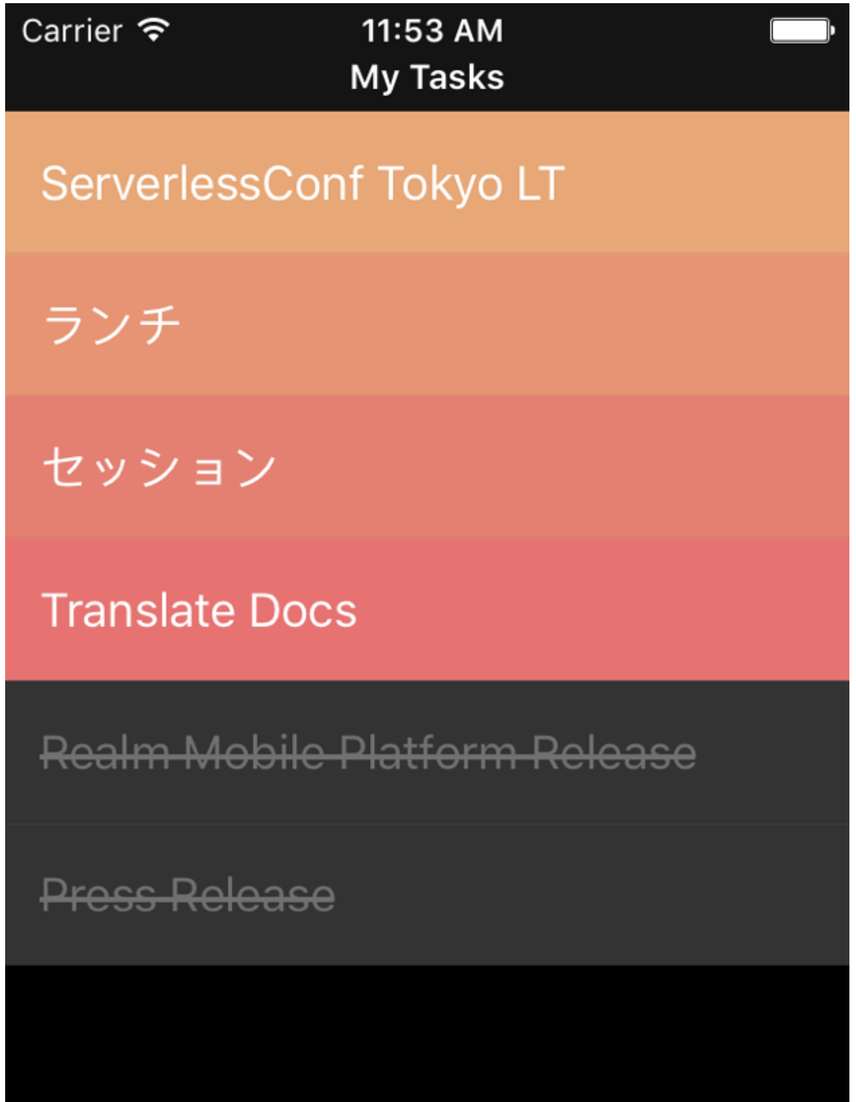


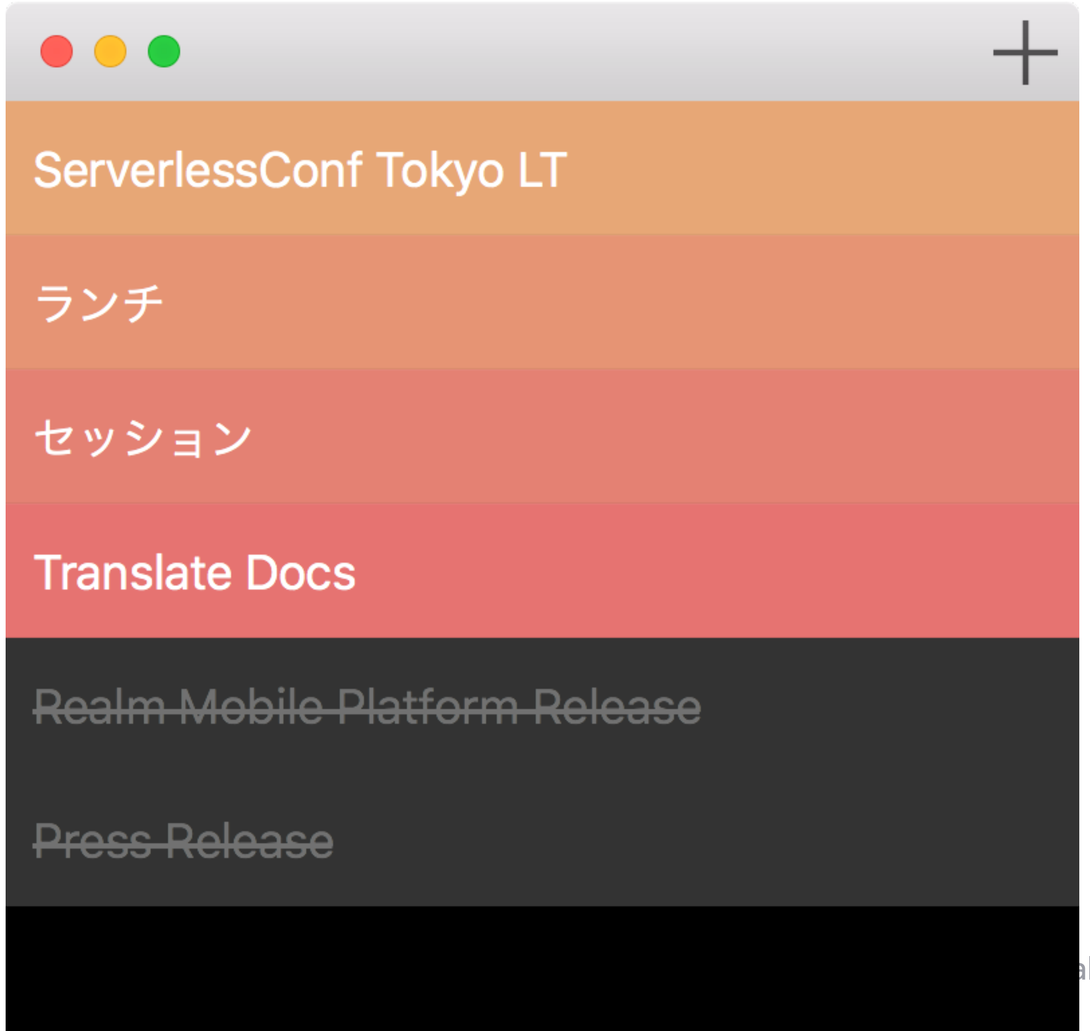
kk@realm.io

# Realm Mobile Database

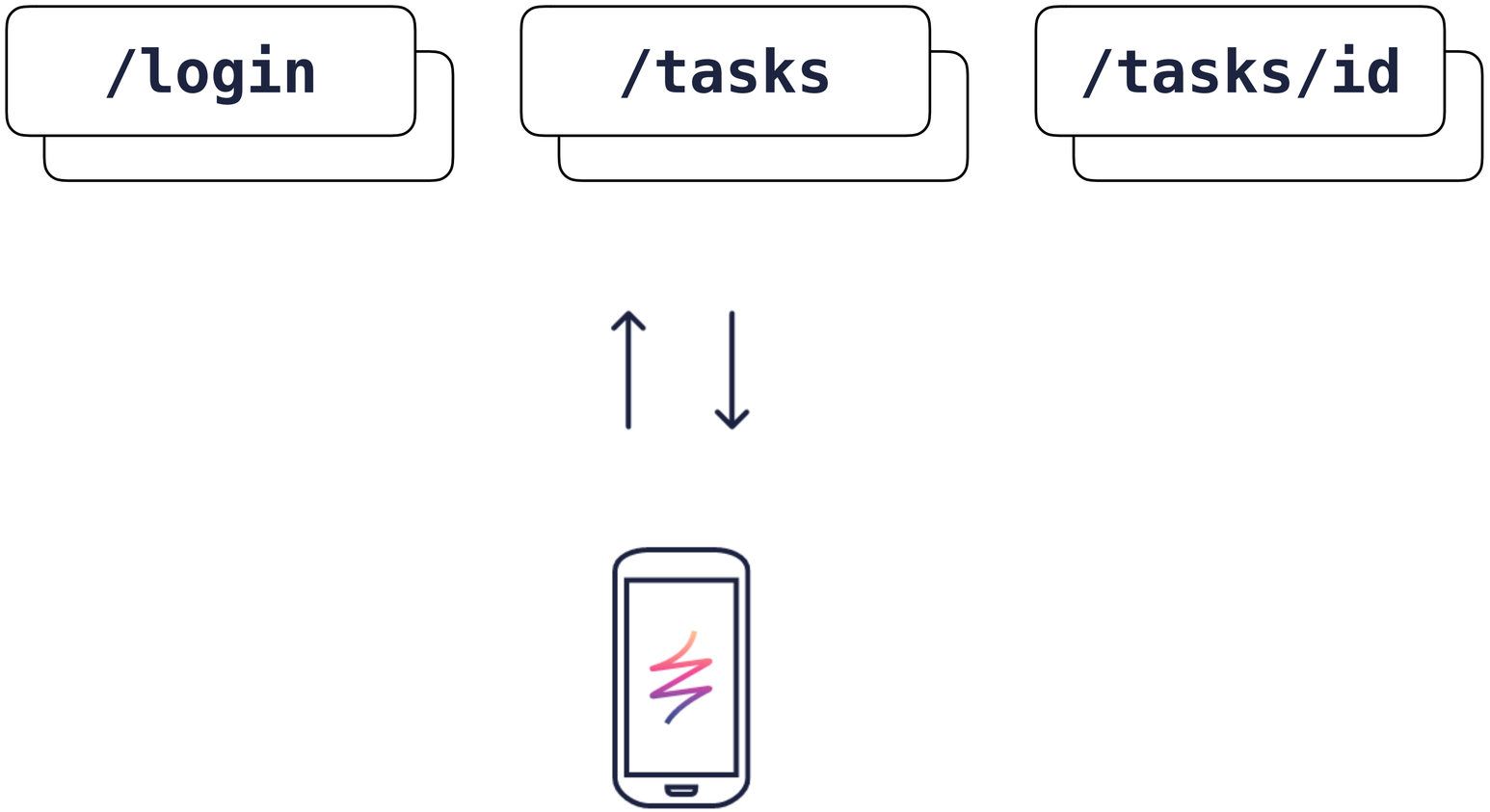
# Realm Mobile Platform

# Demo


Carrier
11:53 AM
My Tasks
ServerlessConf Tokyo LT
ランチ
セッション
Translate Docs
Realm Mobile Platform Release
Press Release


ServerlessConf Tokyo LT
ランチ
セッション
Translate Docs
Realm Mobile Platform Release
Press Release

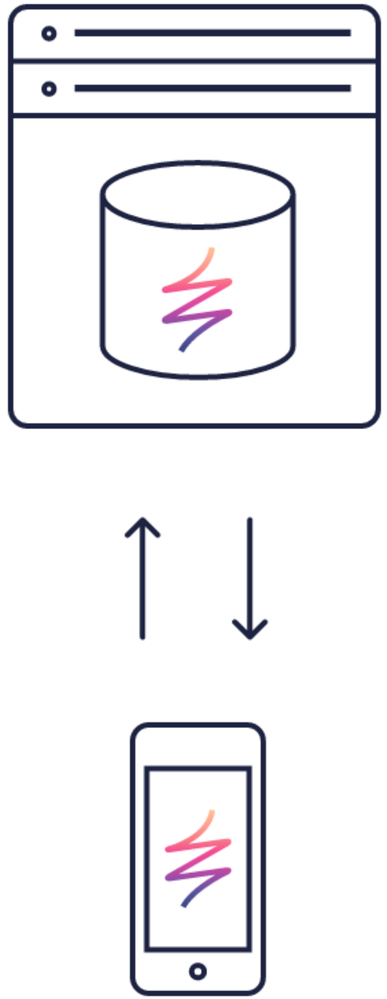
# Typical Mobile App Development






kk@realm.io

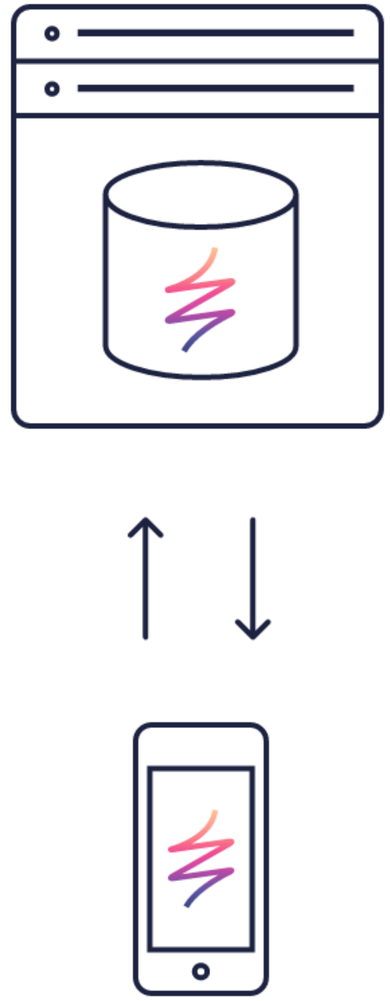
# w/ Realm Mobile Platform





kk@realm.io

# Data Sync

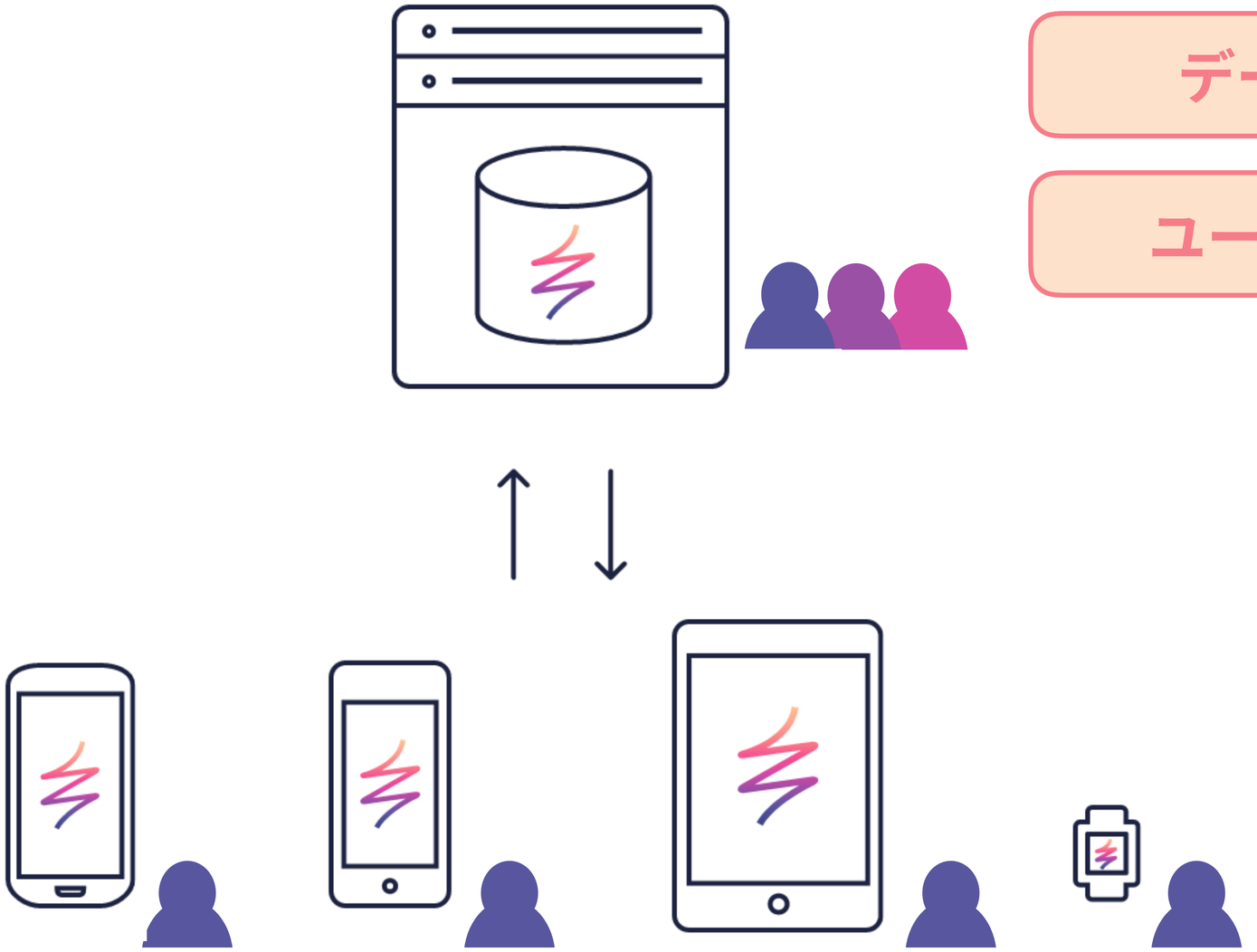


データ同期



kk@realm.io

# User Identify



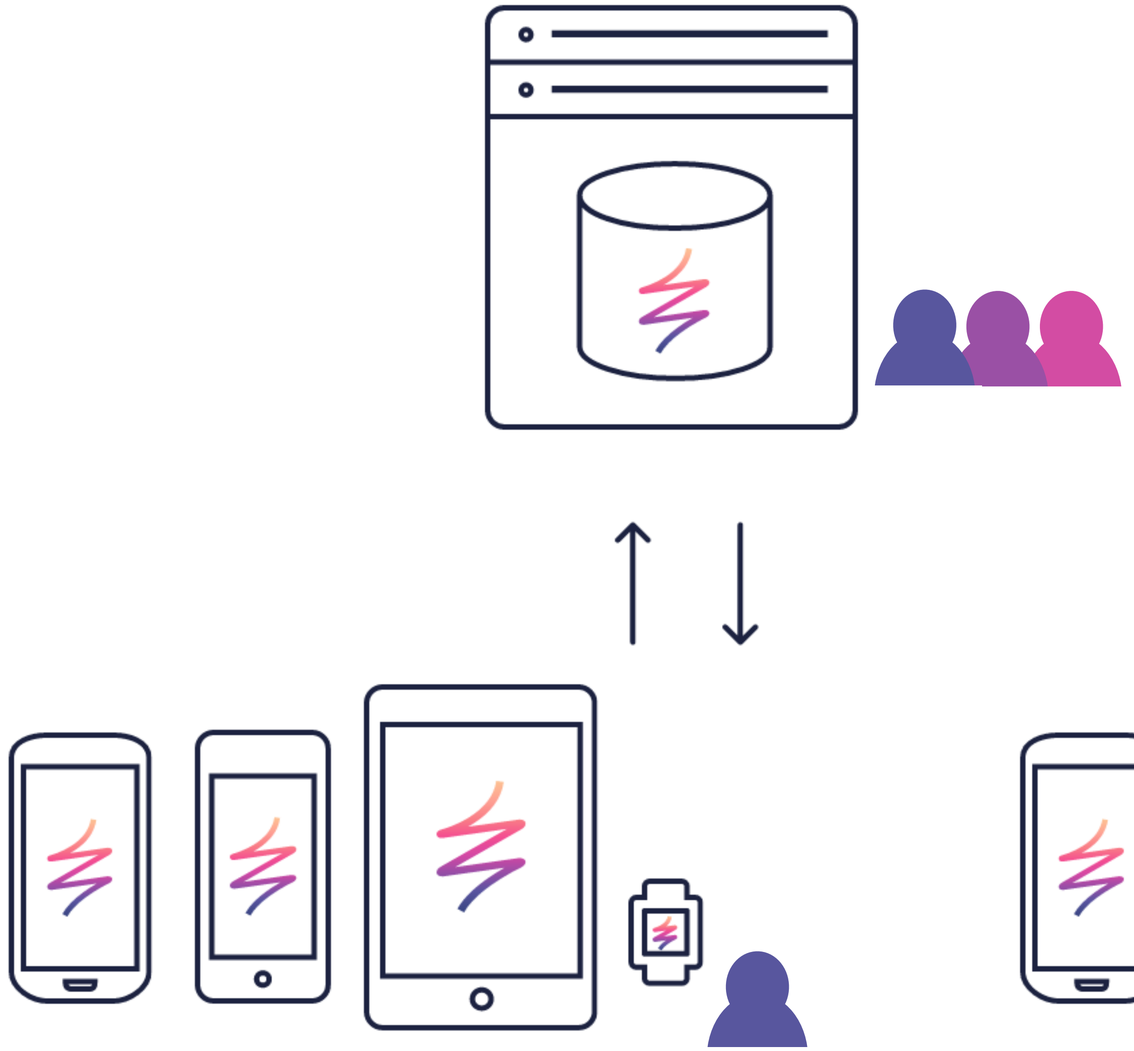
データ同期

ユーザー識別



kk@realm.io

# Access Control

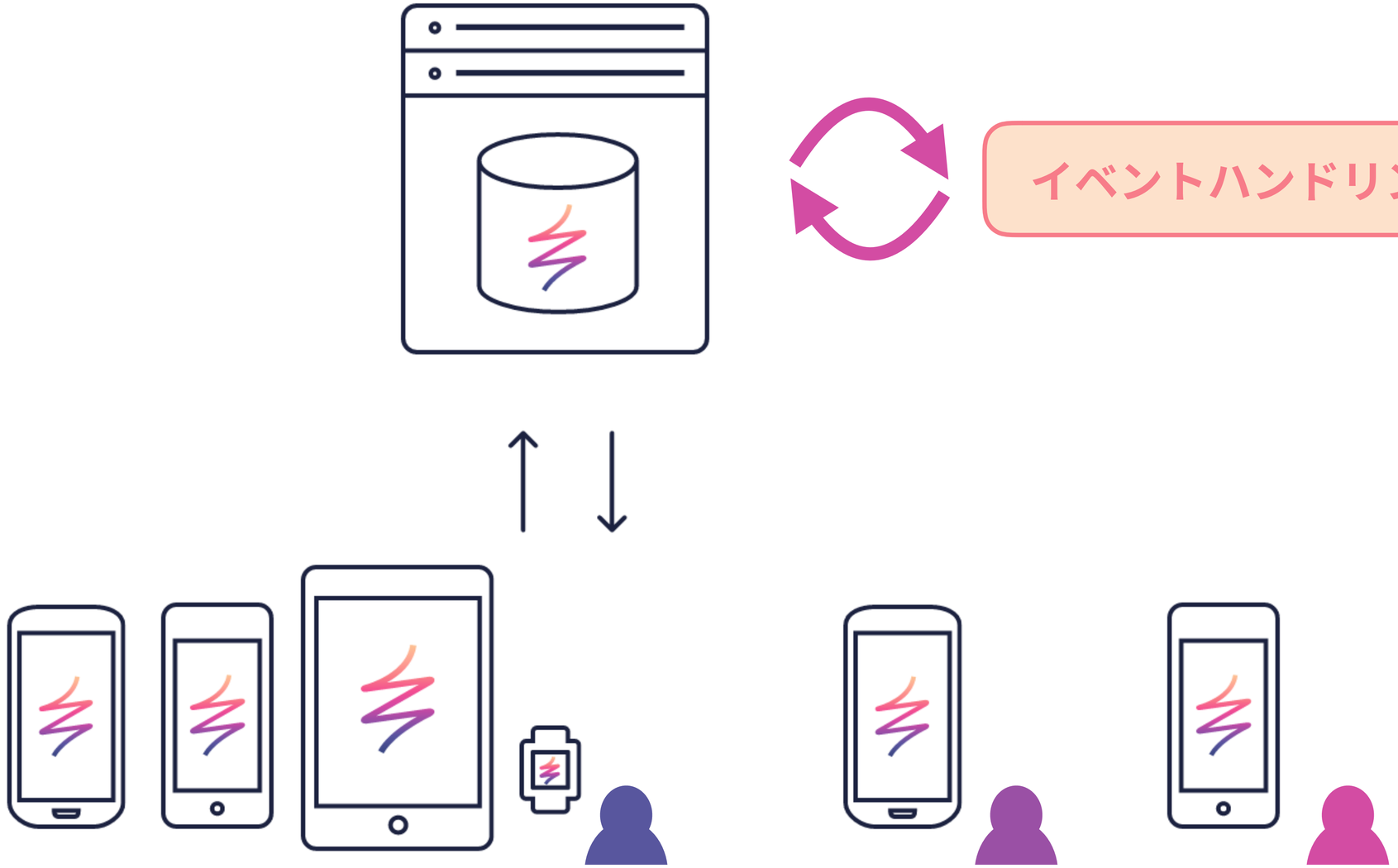


- データ同期
- ユーザー識別
- アクセスコントロール

realm

kk@realm.io

# Event Handling



イベントハンドリング



kk@realm.io

# Realm Mobile Platform

- Mobile Database ＋
- 双方向のデータ同期
  - リアルタイム
  - シームレス
  - コンフリクトの自動解決
- ユーザー識別
- イベントハンドリング
- サーバープッシュ
- データ共有
  - アクセスコントロール



kk@realm.io

# コンフリクトの解決

- 基本は後勝ち
  - 同じアイテムに別の変更を加えた場合、後に起こった変更が採用されます。
- 同じインデックスへの挿入は時間順になる
- 削除はすべてに勝つ
  - あるアイテムが削除されて、別のデバイスでは同じアイテムを変更した、という場合は、削除だけが起こります。

# Demo

github.com/realm/RealmTasks



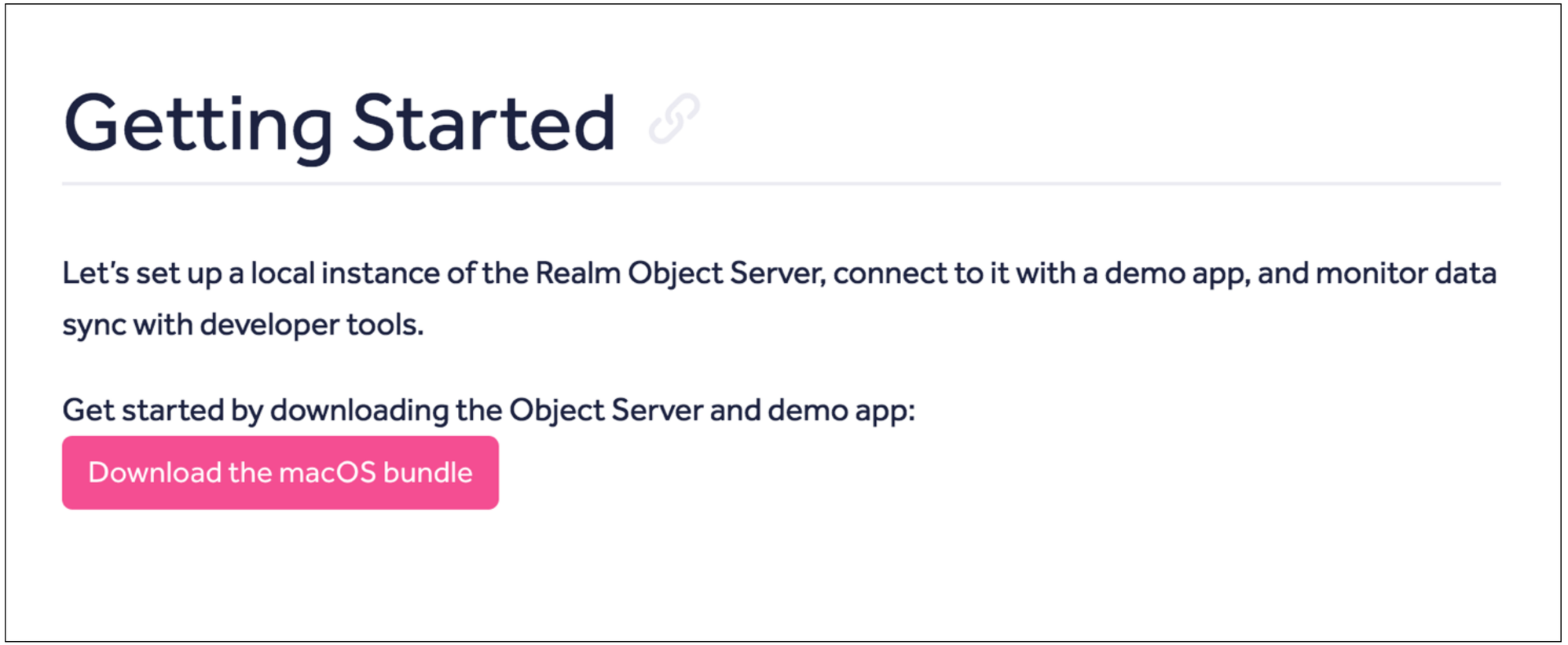
kk@realm.io

# 使ってみる

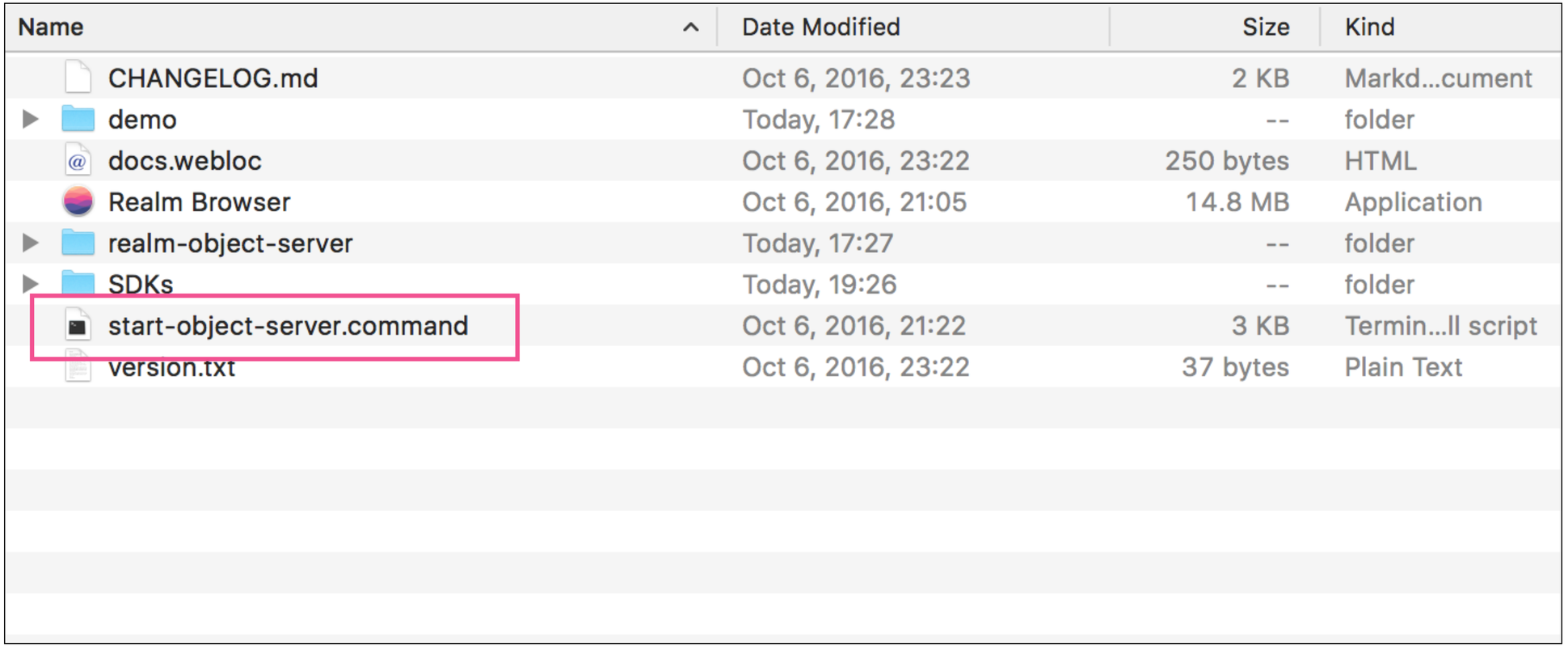


kk@realm.io

# セットアップ


Getting Started
Let's set up a local instance of the Realm Object Server, connect to it with a demo app, and monitor data sync with developer tools.
Get started by downloading the Object Server and demo app:
Download the macOS bundle

# Realm Object Serverを起動する



# Adminアカウントを作る


kk@realm.io

realm

Object Server

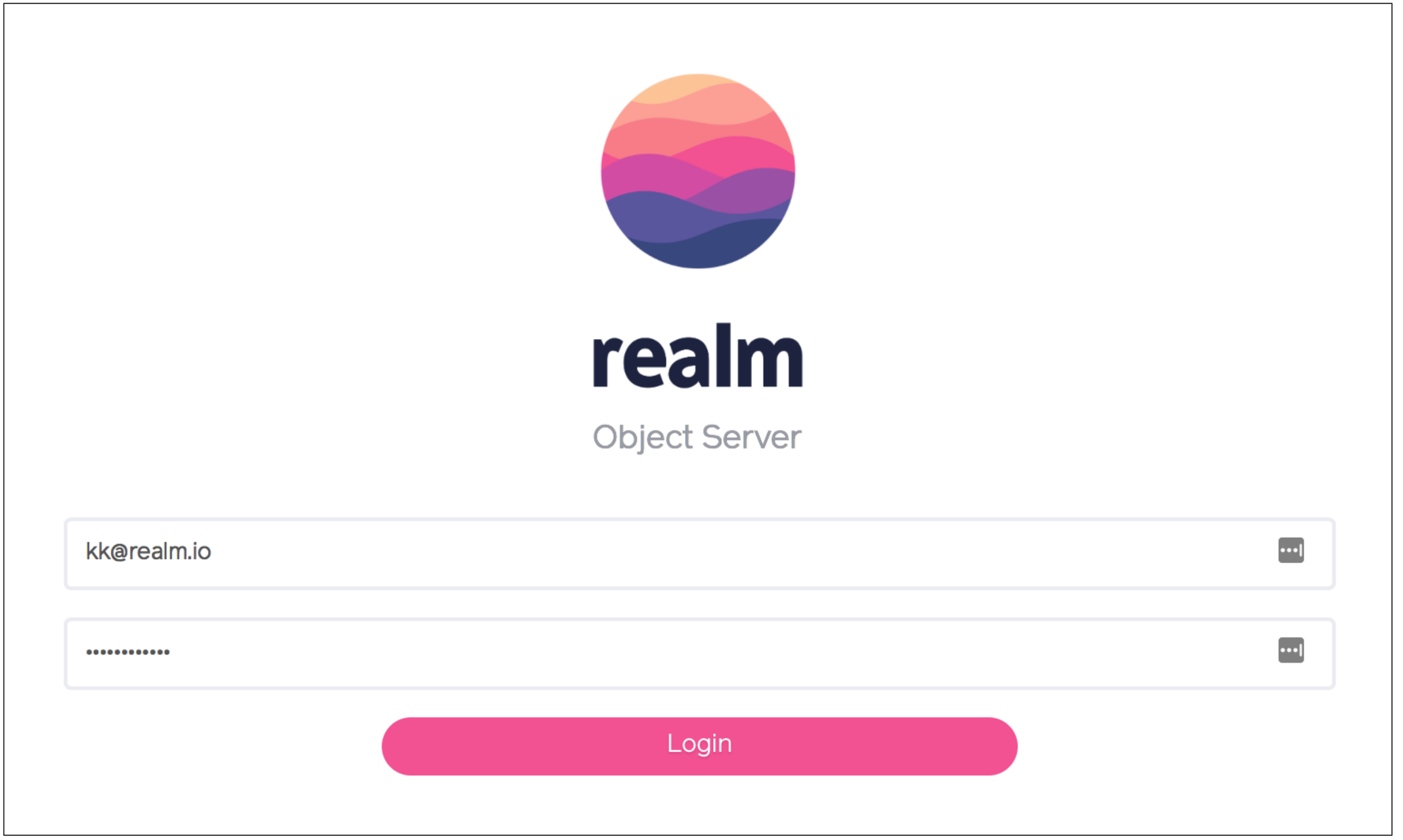
## Create Admin User

The Realm Object Server requires an admin user to start. Register your admin user below to complete the setup and enable the server.

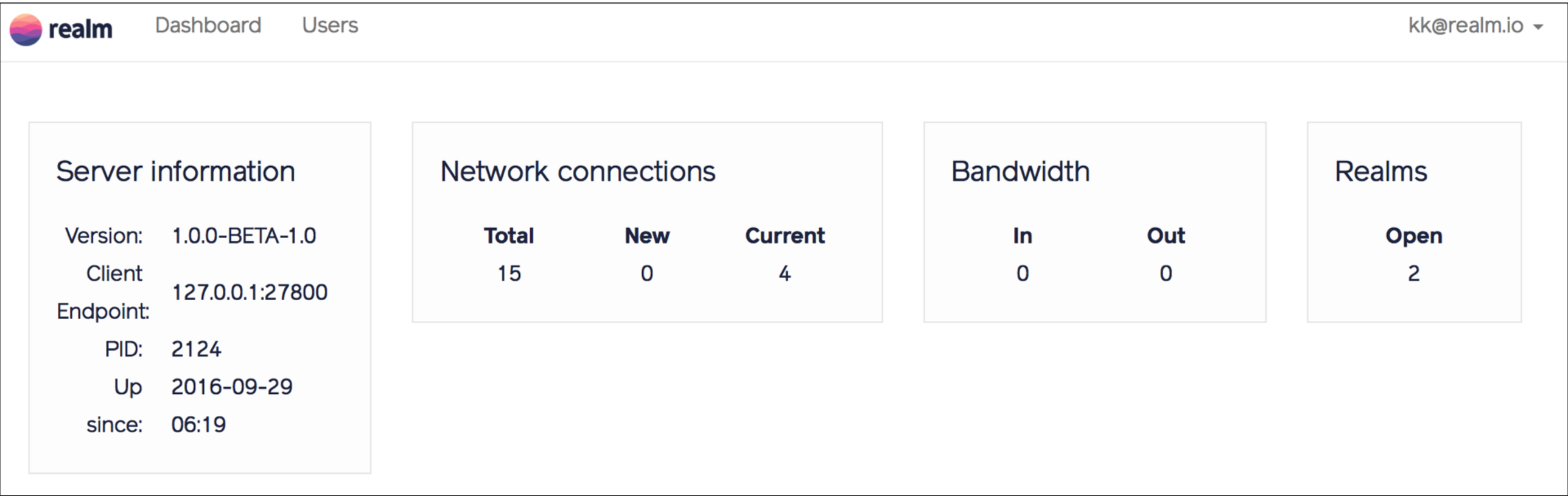
*Admin users have access to the dashboard and all synchronized Realms, so please safeguard the login details.*

| | |
|---|---|
| **Email address** | kk@realm.io |
| **Password** | ••••••••••••••• |
| **Confirm Password** | ••••••••••••••• |

with ♥ from the Realm team


realm
Object Server
kk@realm.io
Login

# ダッシュボード


realm
Dashboard
Users
kk@realm.io
Server information
Version: 1.0.0-BETA-1.0
Client Endpoint: 127.0.0.1:27800
PID: 2124
Up since: 2016-09-29 06:19
Network connections
Total 15
New 0
Current 4
Bandwidth
In 0
Out 0
Realms
Open 2

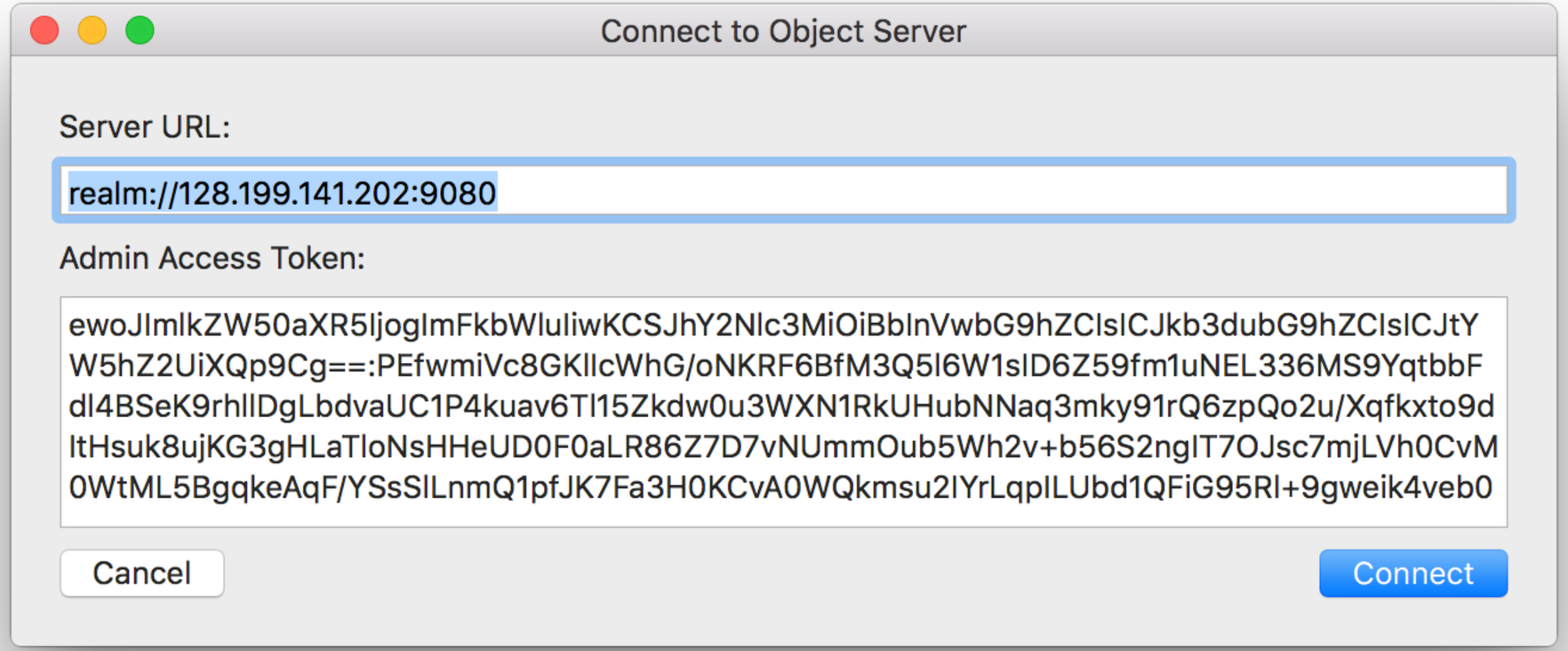
# ダッシュボード


realm
Dashboard
Users
kk@realm.io
User ID
User Credentials
Admin
New User
6e80f8cd22433fd482b185c956d64f62
kk@realm.io
9eb0b3c2b2d4d65877696e5acc86a5d5
test

# Object Server上のデータを見る

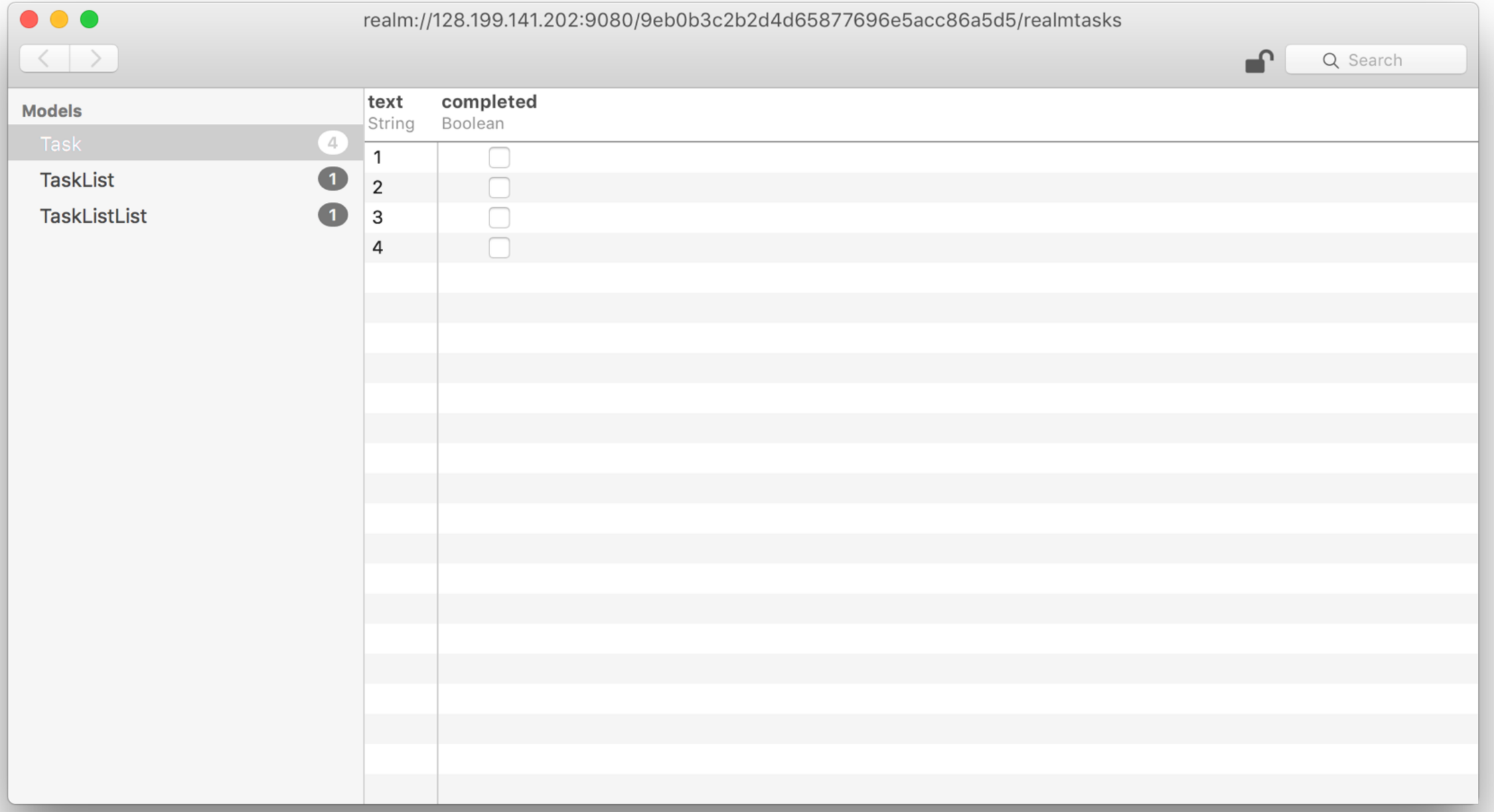

kk@realm.io

# Realm Browser






jp@realm.io

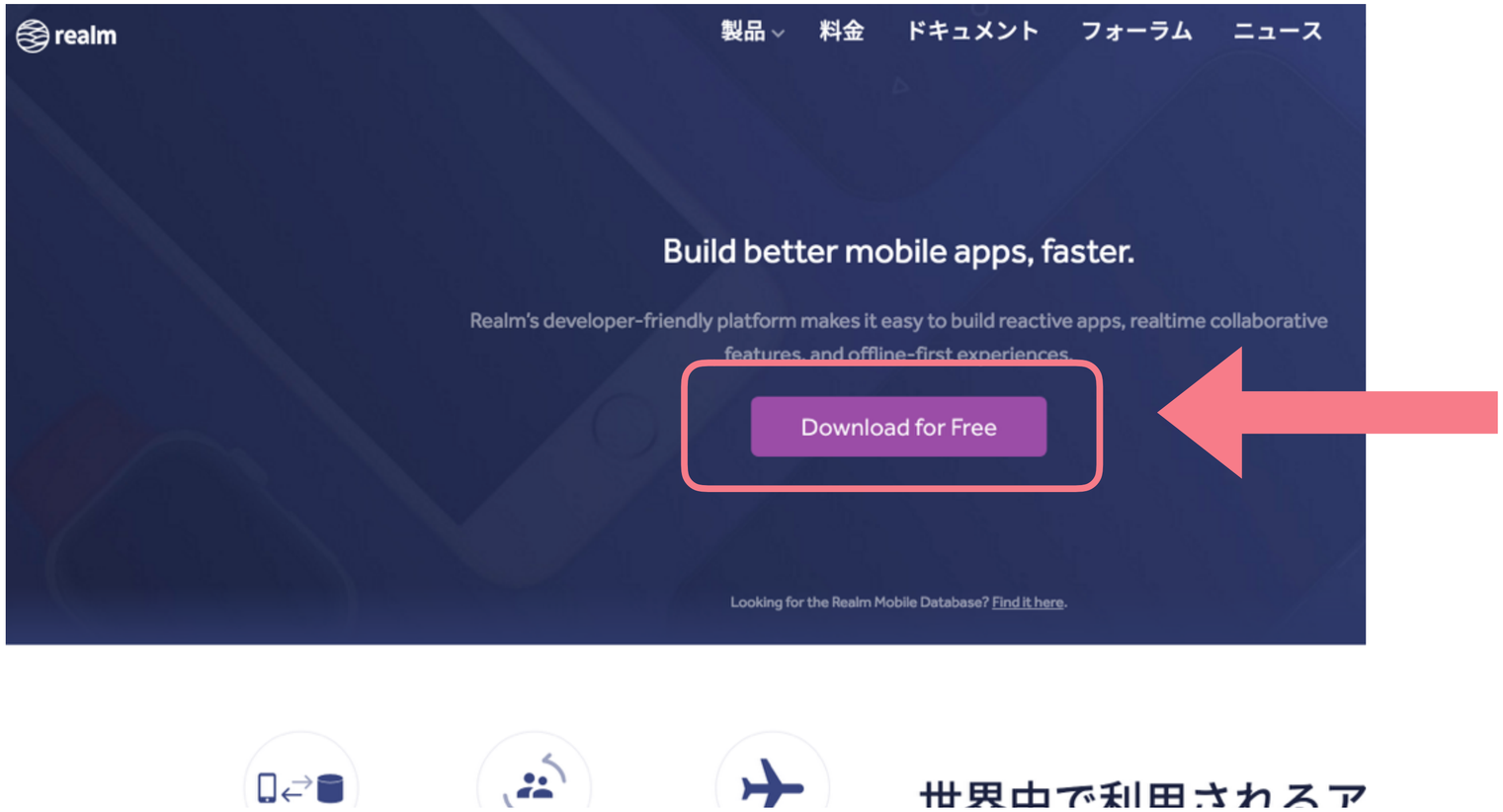

realm://128.199.141.202:9080/9eb0b3c2b2d4d65877696e5acc86a5d5/realmtasks
Search
Models
Task 4
TaskList 1
TaskListList 1
text
String
completed
Boolean
1
2
3
4

# Workshop

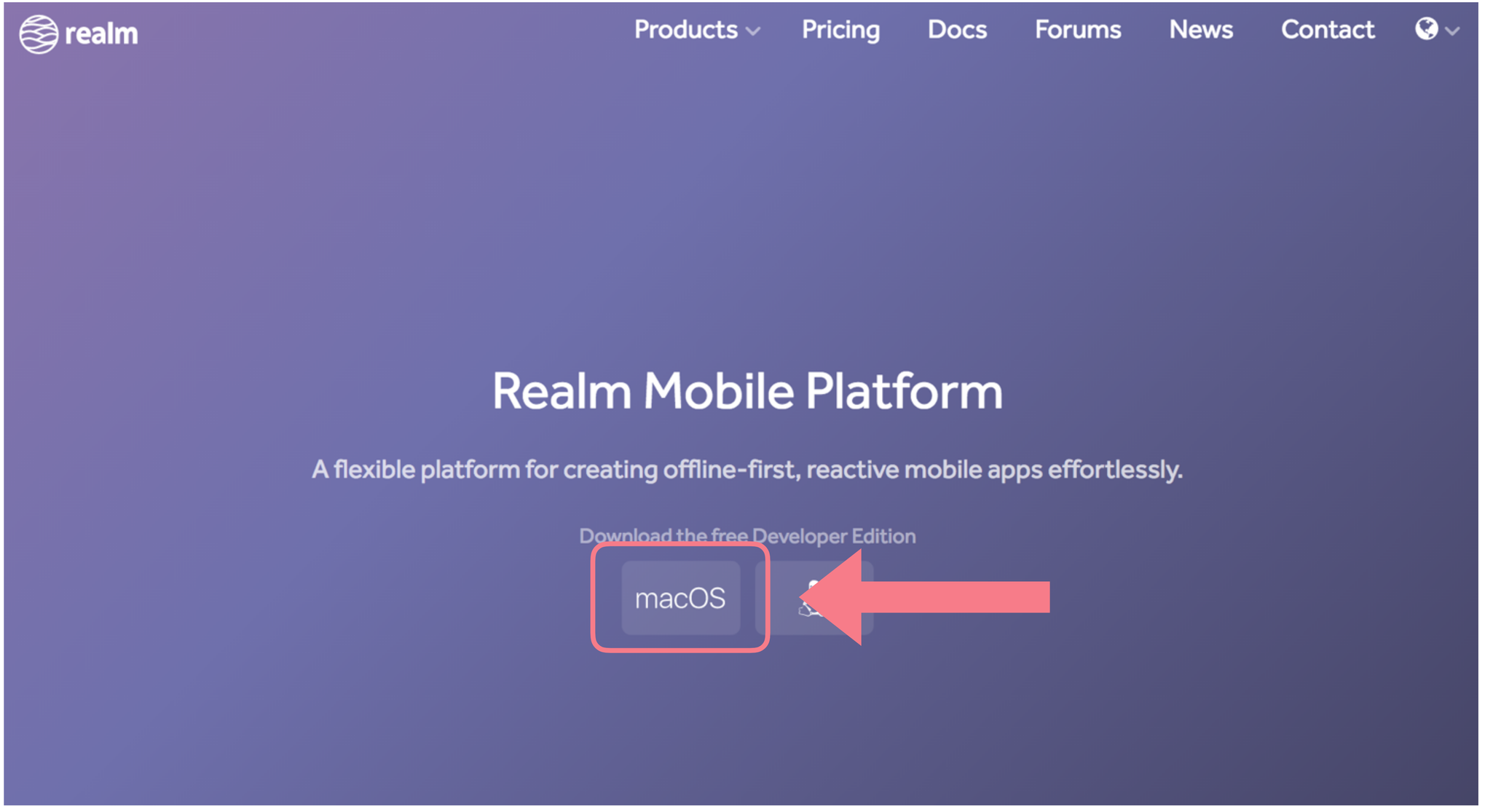


kk@realm.io

# Realmのダウンロード




https://realm.io/jp/


realm
Products
Pricing
Docs
Forums
News
Contact
Realm Mobile Platform
A flexible platform for creating offline-first, reactive mobile apps effortlessly.
Download the free Developer Edition
macOS

# Install the macOS bundle

For macOS, the Realm Object Server comes bundled with a demo app called R
way to learn about the Realm Mobile Platform. (In production, you'll probably w
Server on Linux, or a cloud hosting provider.)

Get started by downloading the Object Server and demo app:

Download the macOS bundle

# Starting Realm Object Server

# 今日作るアプリ



kk@realm.io

# ソースコード

github.com/kishikawakatsumi/Realm-Hands-On



kk@realm.io

# スライド

bit.ly/RealmHandsOn



kk@realm.io

# Agenda



kk@realm.io

# Agenda

- 基本編
  - セットアップ
  - ウォーミングアップ
  - Twitterのタイムラインを表示してみる
  - 引っ張って更新ができるようにする
  - 選択したTweetをLikeできるようにする
- 応用編
  - Tweetを検索できるようにする


kk@realm.io

# セットアップ



kk@realm.io

# セットアップ

- ビルド済みフレームワーク
- CocoaPods
- Carthage

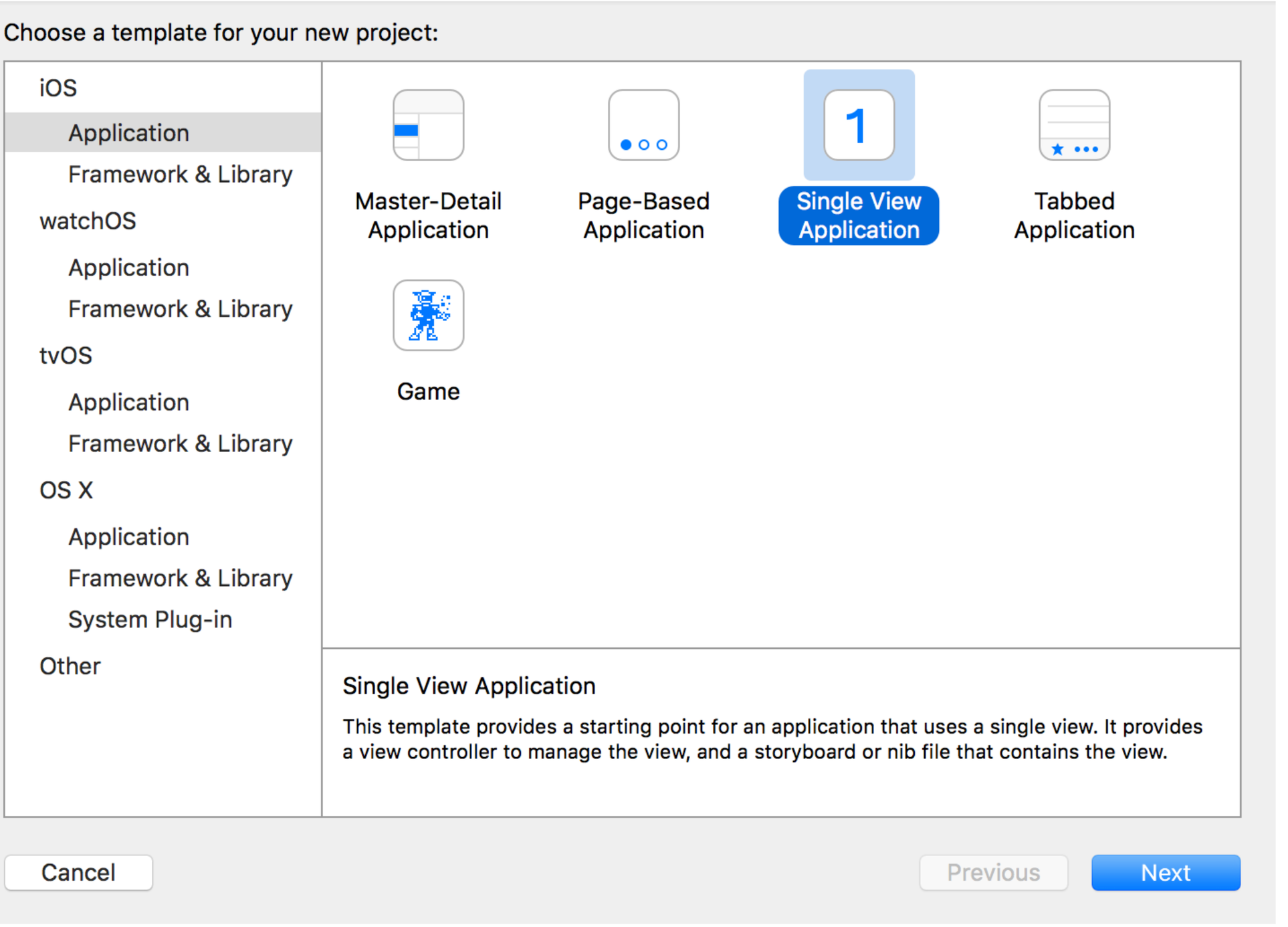

kk@realm.io

# セットアップ

- ビルド済みフレームワーク
- CocoaPods
- Carthage

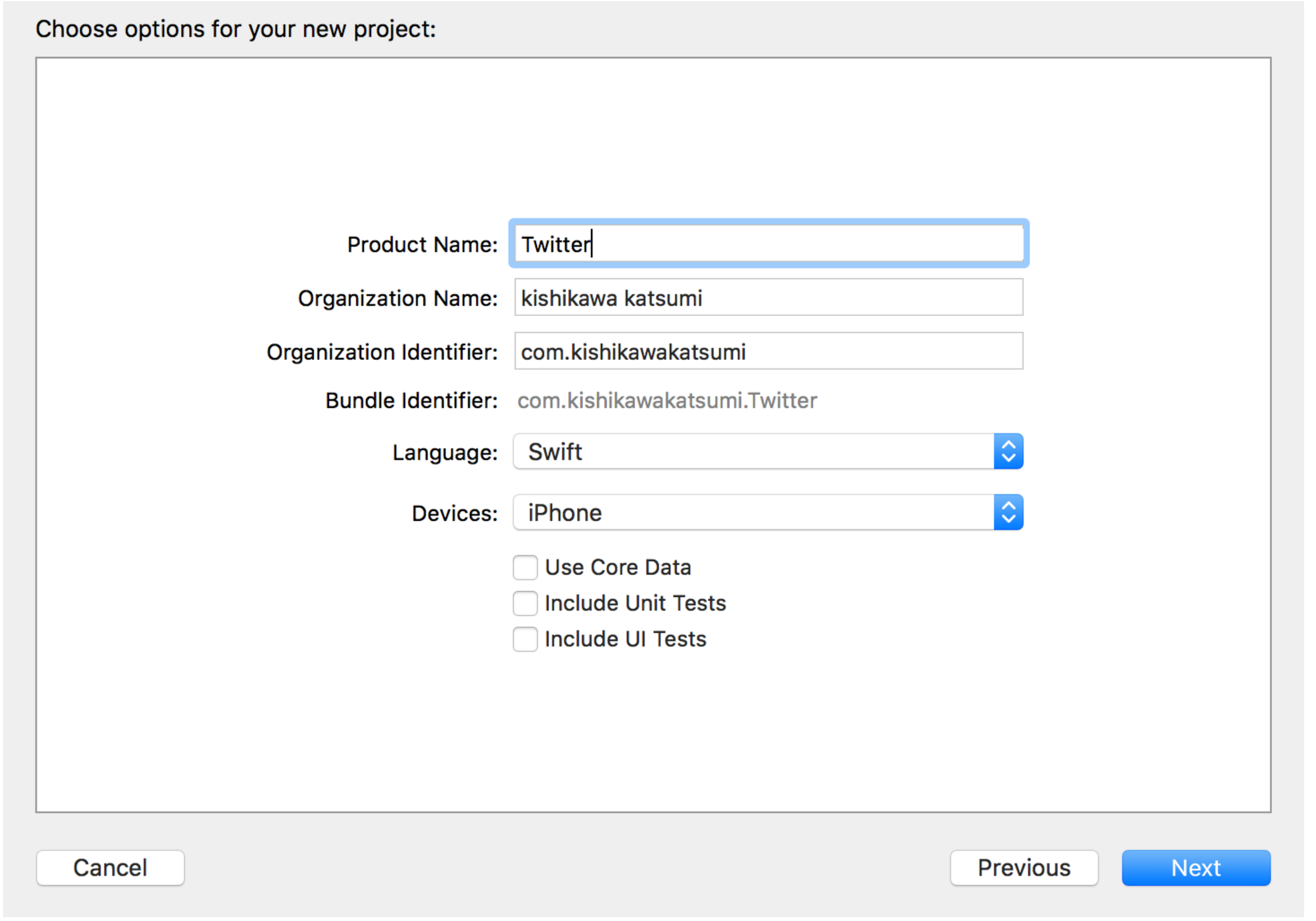


kk@realm.io



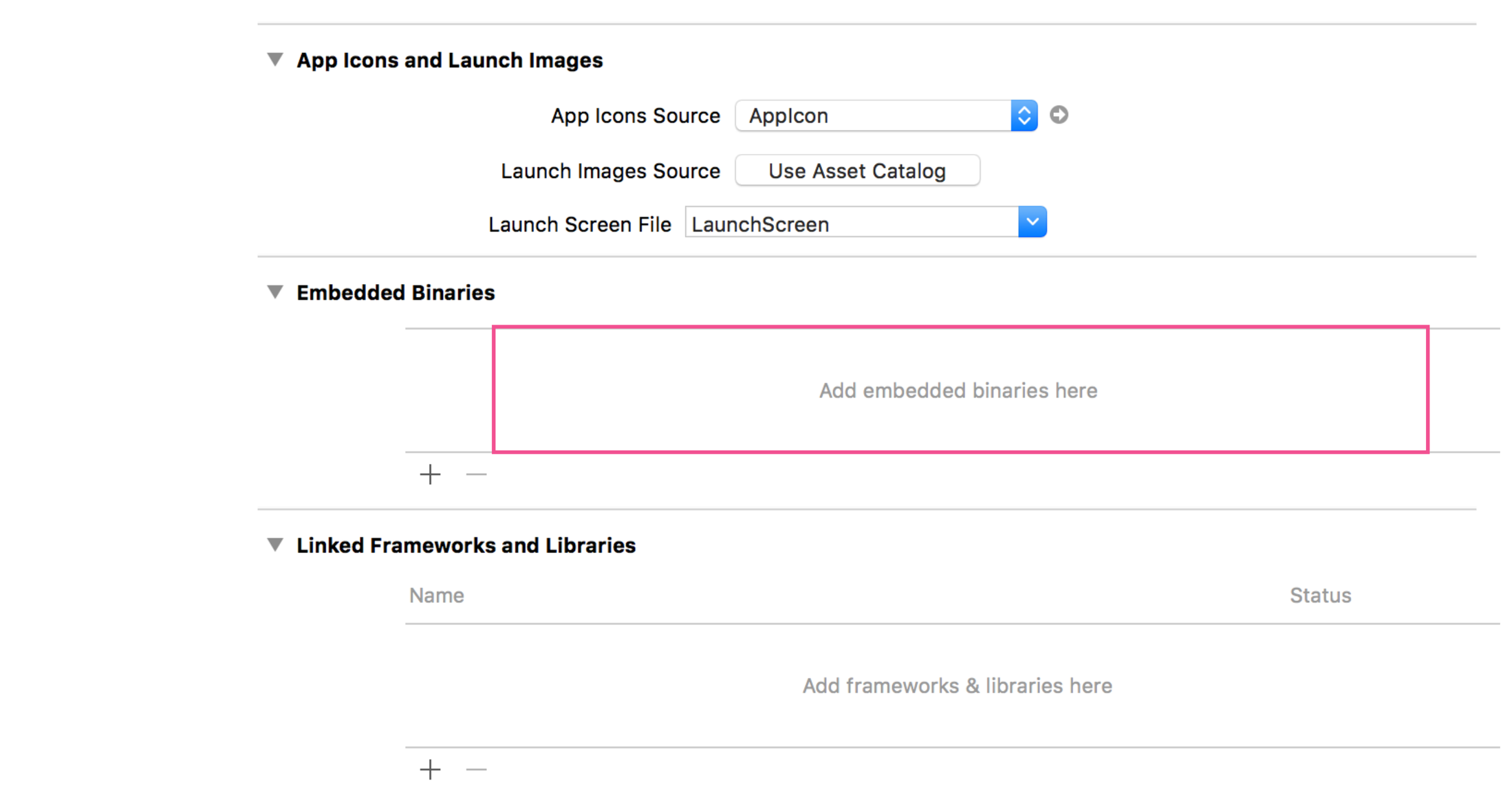



kk@realm.io



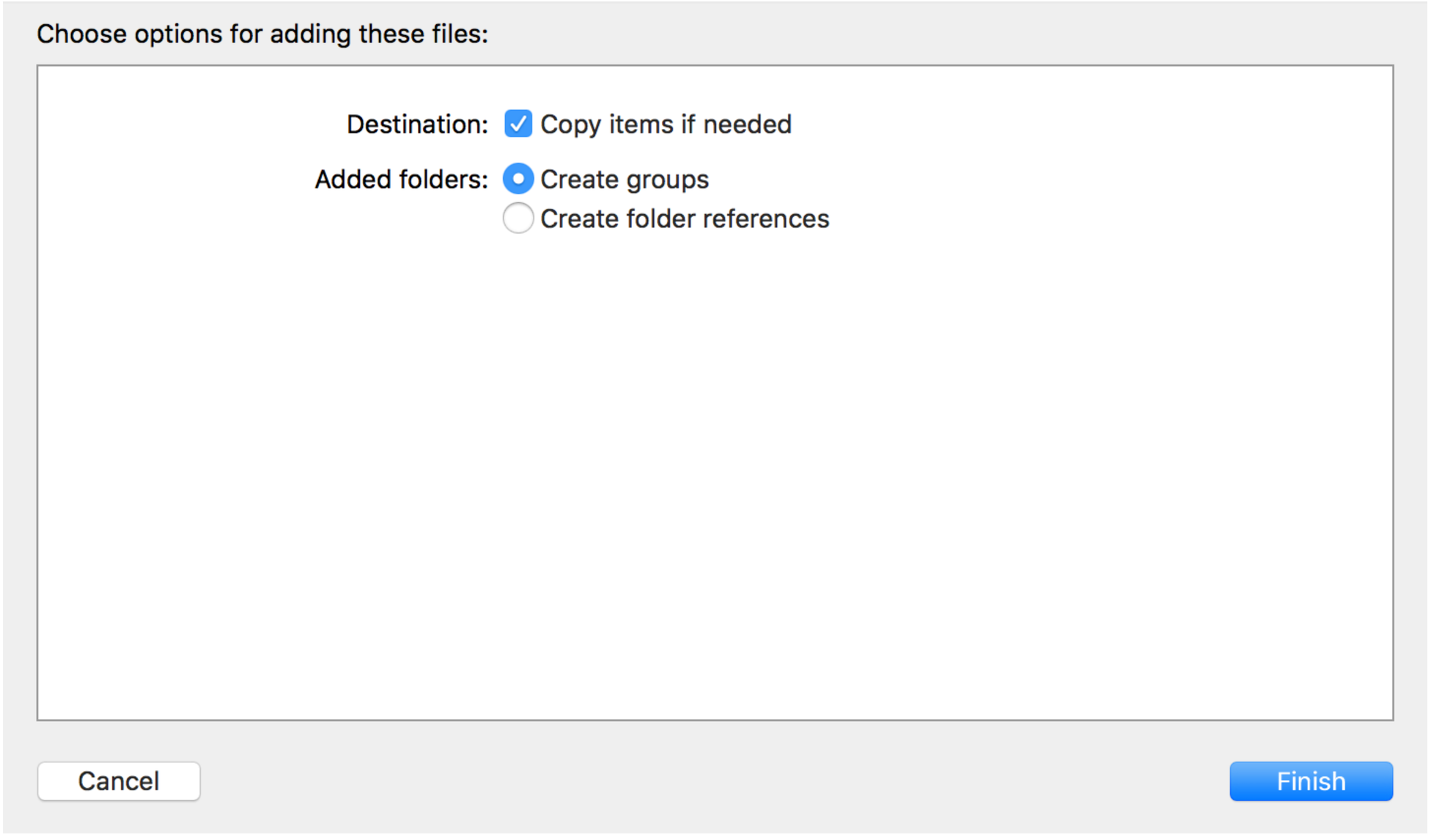



kk@realm.io


App Icons and Launch Images
App Icons Source
AppIcon
Launch Images Source
Use Asset Catalog
Launch Screen File
LaunchScreen
Embedded Binaries
Add embedded binaries here
Linked Frameworks and Libraries
Name
Status
Add frameworks & libraries here

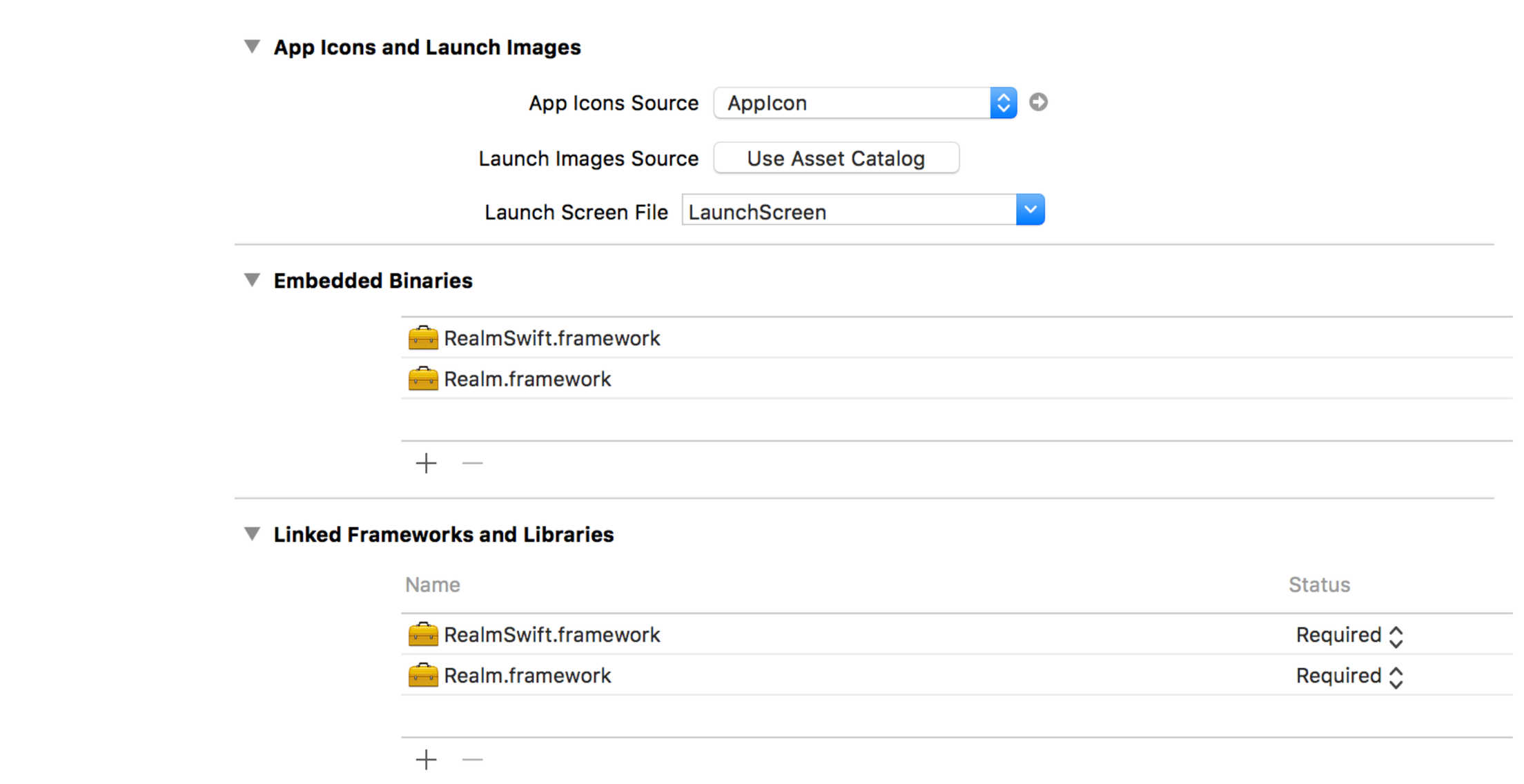


kk@realm.io


Choose options for adding these files:
Destination:
Copy items if needed
Added folders:
Create groups
Create folder references
Cancel
Finish



kk@realm.io


App Icons and Launch Images
App Icons Source
AppIcon
Launch Images Source
Use Asset Catalog
Launch Screen File
LaunchScreen
Embedded Binaries
RealmSwift.framework
Realm.framework
Linked Frameworks and Libraries
Name
Status
RealmSwift.framework
Required
Realm.framework
Required



kk@realm.io

# 確認する

```
import UIKit
import RealmSwift

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(application: UIApplication,
                     didFinishLaunchingWithOptions
        launchOptions: [NSObject: AnyObject]?) -> Bool {
        let realm = try! Realm()
        print(realm)

        return true
    }

}
```



kk@realm.io

# ウォームアップ


kk@realm.io

# ウォームアップ

```
import Foundation
import RealmSwift

class Tweet: Object {
    dynamic var name = ""
    dynamic var text = ""
    dynamic var iconURL = ""
    dynamic var id = ""
    dynamic var createdAt = Date()
}
```



kk@realm.io

# ウォームアップ

```
let tweet = Tweet()
tweet.name = "test name"
tweet.text = "test text"

try! realm.write {
    realm.add(tweet)
}

let tweets = realm.objects(Tweet.self)

for tweet in tweets {
    print(tweet.name)
    print(tweet.text)
    print(tweet.createdAt)
}
```



kk@realm.io

# Twitterのタイムラインを
# 表示してみる


kk@realm.io

# 1-1. タイムラインを表示

```
import Foundation
import RealmSwift

class Tweet: Object {
    dynamic var name = ""
    dynamic var text = ""
    dynamic var iconURL = ""
    dynamic var id = ""
    dynamic var createdAt = Date()
}
```



kk@realm.io

# 1-2. タイムラインを表示

```
import Foundation
import RealmSwift

class Tweet: Object {
    dynamic var name = ""
    dynamic var text = ""
    dynamic var iconURL = ""
    dynamic var id = ""
    dynamic var createdAt = Date()

    static var dateFormatter: DateFormatter {
        let dateFormatter = DateFormatter()
        dateFormatter.locale = Locale(identifier: "en_US_POSIX")
        dateFormatter.dateFormat = "EEE MMM dd HH:mm:ss Z yyyy"
        return dateFormatter
    }

    convenience init(tweetDictionary: [String: AnyObject]) {
        self.init()
        let user = tweetDictionary["user"] as! [String: AnyObject]
        name = user["name"] as! String
        text = tweetDictionary["text"] as! String

        iconURL = user["profile_image_url_https"] as! String

        id = tweetDictionary["id_str"] as! String
        createdAt = Tweet.dateFormatter.date(from: tweetDictionary["created_at"] as! String)!
    }
}
```



kk@realm.io

# 1-3タイムラインを表示

```
import UIKit

class TimelineCell: UITableViewCell {
    @IBOutlet weak var iconView: UIImageView!
    @IBOutlet weak var nameLabel: UILabel!
    @IBOutlet weak var tweetTextView: UITextView!

    override func prepareForReuse() {
        iconView.image = nil
        nameLabel.text = nil
        tweetTextView.text = nil
    }
}
```

# 1-4a.タイムラインを表示

```
import UIKit
import Accounts
import Social
import RealmSwift

class TimelineViewController: UITableViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
    }

}
```



kk@realm.io

# 1-4b.タイムラインを表示

```
import UIKit
import Accounts
import Social
import RealmSwift

class TimelineViewController: UITableViewController {

    var account: ACAccount?

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    func getHomeTimeline() {
        let requestURL = URL(string: "https://api.twitter.com/1/statuses/home_timeline.json")
        let request = SLRequest(forServiceType: SLServiceTypeTwitter, requestMethod: .GET, url: requestURL, parameters: nil)
        request?.account = account

        request?.perform { (data, response, error) -> Void in
            if let error = error {
                self.showAlert(error.localizedDescription)
                return
            }

            do {
                let results = try JSONSerialization.jsonObject(with: data!, options: [])
                if let results = results as? NSDictionary {
                    let errors = results["errors"] as! [[String: AnyObject]]
                    let message = errors.last!["message"] as! String
                    self.showAlert(message)
                    return
                }

                let timeline = results as! [[String: AnyObject]]

                let realm = try! Realm()
                try! realm.write {
                    timeline.forEach { (tweetDictionary) -> () in
                        let tweet = Tweet(tweetDictionary: tweetDictionary)
                        realm.add(tweet)
                    }
                }
            } catch let error as NSError {
                self.showAlert(error.localizedDescription)
            }
        }
    }

    func showAlert(_ message: String) {
        DispatchQueue.main.async {
            let alertController = UIAlertController(title: "Error", message: message, preferredStyle: .alert)
            alertController.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
            self.present(alertController, animated: true, completion: nil)
        }
    }

}
```



kk@realm.io

# 1-4c.タイムラインを表示

```
override func viewDidLoad() {
    super.viewDidLoad()

    let accountStore = ACAccountStore()
    let accountType = accountStore.accountType(withAccountTypeIdentifier: ACAccountTypeIdentifie
    accountStore.requestAccessToAccounts(with: accountType, options: nil) { (granted, error) -> V
        if granted {
            let accounts = accountStore.accounts(with: accountType)
            if let account = accounts?.first as? ACAccount {
                self.account = account
                self.getHomeTimeline()
            } else {
                self.showAlert("No Twitter account")
            }
        } else {
            self.showAlert("No account access")
        }
    }
}
```

# 1-5a.タイムラインを表示

```
import UIKit
import Accounts
import Social
import RealmSwift

class TimelineViewController: UITableViewController {

    var timeline: Results<Tweet>?
    var notificationToken: NotificationToken?

    var account: ACAccount?

    override func viewDidLoad() {
        super.viewDidLoad()

        ...
```



kk@realm.io

# 1-5b.タイムラインを表示

```
override func viewDidLoad() {
    super.viewDidLoad()

    let accountStore = ACAccountStore()
    let accountType = accountStore.accountType(withAccountTypeIdentifier: ACAccountTypeIdentifierTwitter)
    accountStore.requestAccessToAccounts(with: accountType, options: nil) { (granted, error) -> Void in
        ...
    }

    tableView.register(UINib(nibName: "TimelineCell", bundle: nil), forCellReuseIdentifier: "timelineCell")
    tableView.rowHeight = 90
    tableView.estimatedRowHeight = 90

    let realm = try! Realm()
    timeline = realm.objects(Tweet.self).sorted(byKeyPath: "createdAt", ascending: false)
    notificationToken = timeline?.addNotificationBlock { [weak self] (change) in
        switch change {
        case .initial(_):
            self?.tableView.reloadData()
        case .update(_, deletions: _, insertions: _, modifications: _):
            self?.tableView.reloadData()
        case .error(_):
            return
        }
    }
}
```



kk@realm.io

# 1-6.タイムラインを表示

```
class TimelineViewController: UITableViewController {

    var timeline: Results<Tweet>?
    var notificationToken: NotificationToken?

    var account: ACAccount?

    override func viewDidLoad() {
        ...
    }

    func getHomeTimeline() {
        ...
    }

    func showAlert(_ message: String) {
        ...
    }

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return timeline?.count ?? 0
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "timelineCell", for: indexPath) as! TimelineCell

        let tweet = timeline![indexPath.row]

        cell.nameLabel.text = tweet.name
        cell.tweetTextView.text = tweet.text

        URLSession.shared.dataTask(with: URLRequest(url: URL(string: tweet.iconURL)!)) { (data, response, error) -> Void in
            if let _ = error {
                return
            }

            DispatchQueue.main.async {
                let image = UIImage(data: data!)!
                cell.iconView.image = image
            }
        }.resume()

        return cell
    }

    override func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return UITableViewAutomaticDimension
    }

}
```



kk@realm.io

# タイムラインを表示

```
import UIKit
import Accounts
import Social
import RealmSwift

class TimelineViewController: UITableViewController {

    var timeline: Results<Tweet>?
    var notificationToken: NotificationToken?

    var account: ACAccount?

    override func viewDidLoad() {
        super.viewDidLoad()

        let accountStore = ACAccountStore()
        let accountType = accountStore.accountType(withAccountTypeIdentifier: ACAccountTypeIdentifierTwitter)
        accountStore.requestAccessToAccounts(with: accountType, options: nil) { (granted, error) -> Void in
            if granted {
                let accounts = accountStore.accounts(with: accountType)
                if let account = accounts?.first as? ACAccount {
                    self.account = account
                    self.getHomeTimeline()
                } else {
                    self.showAlert("No Twitter account")
                }
            } else {
                self.showAlert("No account access")
            }
        }

        tableView.register(UINib(nibName: "TimelineCell", bundle: nil), forCellReuseIdentifier: "timelineCell")
        tableView.rowHeight = 90
        tableView.estimatedRowHeight = 90

        let realm = try! Realm()
        timeline = realm.objects(Tweet.self).sorted(byKeyPath: "createdAt", ascending: false)
        notificationToken = timeline?.addNotificationBlock { [weak self] (change) in
            switch change {
            case .initial(_):
                self?.tableView.reloadData()
            case .update(_, deletions: _, insertions: _, modifications: _):
                self?.tableView.reloadData()
            case .error(_):
                return
            }
        }
    }

    func getHomeTimeline() {
        let requestURL = URL(string: "https://api.twitter.com/1/statuses/home_timeline.json")
        let request = SLRequest(forServiceType: SLServiceTypeTwitter, requestMethod: .GET, url: requestURL, parameters: nil)
        request?.account = account

        request?.perform { (data, response, error) -> Void in
            if let error = error {
                self.showAlert(error.localizedDescription)
                return
            }

            do {
                let results = try JSONSerialization.jsonObject(with: data!, options: [])
                if let results = results as? NSDictionary {
                    let errors = results["errors"] as! [[String: AnyObject]]
                    let message = errors.last!["message"] as! String
                    self.showAlert(message)
                    return
                }

                let timeline = results as! [[String: AnyObject]]

                let realm = try! Realm()
                try! realm.write {
                    timeline.forEach { (tweetDictionary) -> () in
                        let tweet = Tweet(tweetDictionary: tweetDictionary)
                        realm.add(tweet)
                    }
                }
            } catch let error as NSError {
                self.showAlert(error.localizedDescription)
            }
        }
    }

    func showAlert(_ message: String) {
        DispatchQueue.main.async {
            let alertController = UIAlertController(title: "Error", message: message, preferredStyle: .alert)
            alertController.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
            self.present(alertController, animated: true, completion: nil)
        }
    }

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return timeline?.count ?? 0
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "timelineCell", for: indexPath) as! TimelineCell

        let tweet = timeline![indexPath.row]

        cell.nameLabel.text = tweet.name
        cell.tweetTextView.text = tweet.text

        URLSession.shared.dataTask(with: URLRequest(url: URL(string: tweet.iconURL)!)) { (data, response, error) -> Void in
            if let _ = error {
                return
            }

            DispatchQueue.main.async {
                let image = UIImage(data: data!)!
                cell.iconView.image = image
            }
        }.resume()

        return cell
    }

    override func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return UITableViewAutomaticDimension
    }
}
```

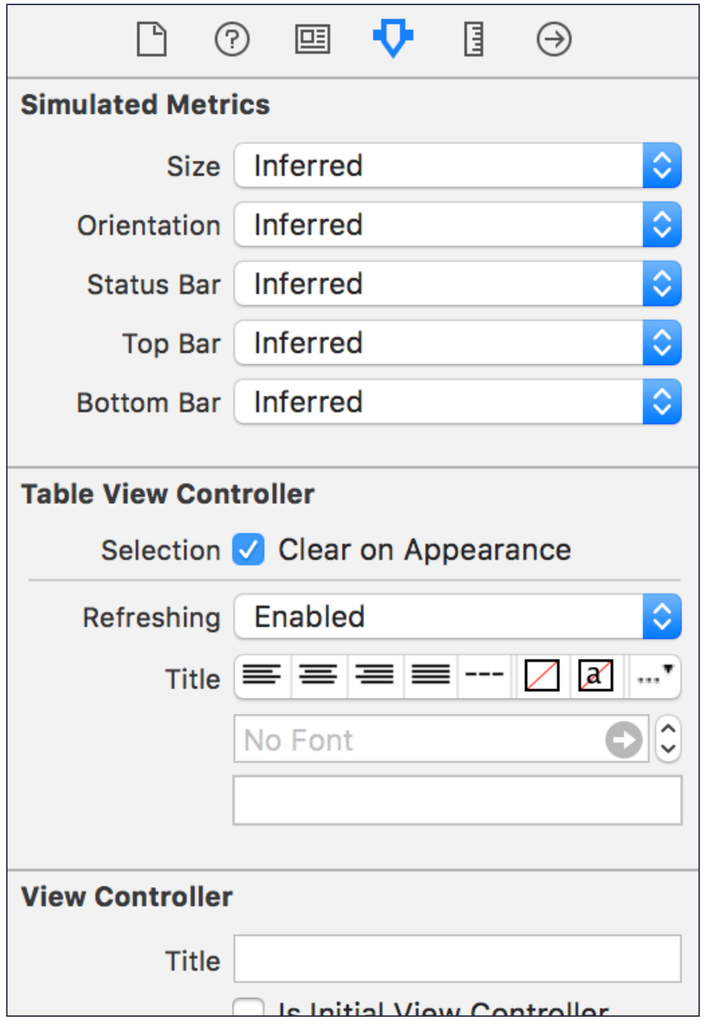


kk@realm.io

「引っ張って更新」が
できるようにする


kk@realm.io

# 2-1a. 引っ張って更新






kk@realm.io

# 2-1b. 引っ張って更新

```
class TimelineViewController: UITableViewController {

    ...

    override func viewDidLoad() {
        super.viewDidLoad()

        let accountStore = ACAccountStore()
        let accountType = accountStore.accountType(withAccountTypeIdentifier: ACAccountTypeIdentifierTwitter)
        accountStore.requestAccessToAccounts(with: accountType, options: nil) { (granted, error) -> Void in
            ...
        }

        tableView.register(UINib(nibName: "TimelineCell", bundle: nil), forCellReuseIdentifier: "timelineCell")
        tableView.rowHeight = 90
        tableView.estimatedRowHeight = 90

        let realm = try! Realm()
        timeline = realm.objects(Tweet.self).sorted(byKeyPath: "createdAt", ascending: false)
        notificationToken = timeline?.addNotificationBlock { [weak self] (change) in
            ...
        }

        refreshControl?.addTarget(self, action: #selector(TimelineViewController.refresh(_:)), for: .valueChanged)
    }

    func refresh(_ sender: UIRefreshControl) {
        if let _ = self.account {
            getHomeTimeline()
        }
    }

    func getHomeTimeline() {
        ...
    }

    ...

}
```



kk@realm.io

# 2-1c. 引っ張って更新

```
func getHomeTimeline() {
    let requestURL = NSURL(string: "https://api.twitter.com/1/statuses/home_timeline.json")
    let request = SLRequest(forServiceType: SLServiceTypeTwitter, requestMethod: .GET, URL: requestURL, parameters: nil)
    request.account = account

    request.performRequestWithHandler { (data, response, error) -> Void in
        if let error = error {
            self.showAlert(error.localizedDescription)
            return
        }

        do {
            let results = try NSJSONSerialization.JSONObjectWithData(data, options: [])
            if let results = results as? NSDictionary {
                let errors = results["errors"] as! [[String: AnyObject]]
                let message = errors.last!["message"] as! String
                self.showAlert(message)
                return
            }

            let timeline = results as! [[String: AnyObject]]

            let realm = try! Realm()
            try! realm.write {
                timeline.forEach { (tweetDictionary) -> () in
                    let tweet = Tweet(tweetDictionary: tweetDictionary)
                    realm.add(tweet)
                }
            }
        } catch let error as NSError {
            self.showAlert(error.localizedDescription)
        }

        dispatch_async(dispatch_get_main_queue()) {
            self.refreshControl?.endRefreshing()
        }
```



kk@realm.io

# 引っ張って更新

```
import UIKit
import Accounts
import Social
import RealmSwift

class TimelineViewController: UITableViewController {

    var timeline: Results<Tweet>?
    var notificationToken: NotificationToken?

    var account: ACAccount?

    override func viewDidLoad() {
        super.viewDidLoad()

        let accountStore = ACAccountStore()
        let accountType = accountStore.accountType(withAccountTypeIdentifier: ACAccountTypeIdentifierTwitter)
        accountStore.requestAccessToAccounts(with: accountType, options: nil) { (granted, error) -> Void in
            if granted {
                let accounts = accountStore.accounts(with: accountType)
                if let account = accounts?.first as? ACAccount {
                    self.account = account
                    self.getHomeTimeline()
                } else {
                    self.showAlert("No Twitter account")
                }
            } else {
                self.showAlert("No account access")
            }
        }

        tableView.register(UINib(nibName: "TimelineCell", bundle: nil), forCellReuseIdentifier: "timelineCell")
        tableView.rowHeight = 90
        tableView.estimatedRowHeight = 90

        let realm = try! Realm()
        timeline = realm.objects(Tweet.self).sorted(byKeyPath: "createdAt", ascending: false)
        notificationToken = timeline?.addNotificationBlock { [weak self] (change) in
            switch change {
            case .initial(_):
                self?.tableView.reloadData()
            case .update(_, deletions: _, insertions: _, modifications: _):
                self?.tableView.reloadData()
            case .error(_):
                return
            }
        }

        refreshControl?.addTarget(self, action: #selector(TimelineViewController.refresh(_:)), for: .valueChanged)
    }

    func refresh(_ sender: UIRefreshControl) {
        if let _ = self.account {
            getHomeTimeline()
        }
    }

    func getHomeTimeline() {
        let requestURL = URL(string: "https://api.twitter.com/1/statuses/home_timeline.json")
        let request = SLRequest(forServiceType: SLServiceTypeTwitter, requestMethod: .GET, url: requestURL, parameters: nil)
        request?.account = account

        request?.perform { (data, response, error) -> Void in
            if let error = error {
                self.showAlert(error.localizedDescription)
                return
            }

            do {
                let results = try JSONSerialization.jsonObject(with: data!, options: [])
                if let results = results as? NSDictionary {
                    let errors = results["errors"] as! [[String: AnyObject]]
                    let message = errors.last!["message"] as! String
                    self.showAlert(message)
                    return
                }

                let timeline = results as! [[String: AnyObject]]

                let realm = try! Realm()
                try! realm.write {
                    timeline.forEach { (tweetDictionary) -> () in
                        let tweet = Tweet(tweetDictionary: tweetDictionary)
                        realm.add(tweet)
                    }
                }
            } catch let error as NSError {
                self.showAlert(error.localizedDescription)
            }

            DispatchQueue.main.async {
                self.refreshControl?.endRefreshing()
            }
        }
    }

    func showAlert(_ message: String) {
        DispatchQueue.main.async {
            let alertController = UIAlertController(title: "Error", message: message, preferredStyle: .alert)
            alertController.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
            self.present(alertController, animated: true, completion: nil)
        }
    }

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return timeline?.count ?? 0
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "timelineCell", for: indexPath) as! TimelineCell

        let tweet = timeline![indexPath.row]

        cell.nameLabel.text = tweet.name
        cell.tweetTextView.text = tweet.text

        URLSession.shared.dataTask(with: URLRequest(url: URL(string: tweet.iconURL)!)) { (data, response, error) -> Void in
            if let _ = error {
                return
            }

            DispatchQueue.main.async {
                let image = UIImage(data: data!)!
                cell.iconView.image = image
            }
        }.resume()

        return cell
    }

    override func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return UITableViewAutomaticDimension
    }

}
```

# プライマリキーで
# 重複を取り除く


kk@realm.io

# 3-1a. プライマリキー

```
class Tweet: Object {
    dynamic var name = ""
    dynamic var text = ""
    dynamic var iconURL = ""
    dynamic var id = ""
    dynamic var createdAt = Date()

    static var dateFormatter: DateFormatter {
        ...
    }

    convenience init(tweetDictionary: [String: AnyObject]) {
        ...
    }

    override class func primaryKey() -> String? {
        return "id"
    }
}
```



kk@realm.io

# 3-1b. プライマリキー

```
func getHomeTimeline() {
    ...

    request?.perform { (data, response, error) -> Void in
        if let error = error {
            ...
        }

        do {
            let results = try JSONSerialization.jsonObject(with: data!, options: [])
            if let results = results as? NSDictionary {
                ...
            }

            let timeline = results as! [[String: AnyObject]]

            let realm = try! Realm()
            try! realm.write {
                timeline.forEach { (tweetDictionary) -> () in
                    let tweet = Tweet(tweetDictionary: tweetDictionary)
                    realm.add(tweet, update: true)
                }
            }
        } catch let error as NSError {
            ...
        }

        ...
    }
}
```

# 選択したTweetを
# Likeできるようにする


kk@realm.io

# 4-1. Likeする

```
class Tweet: Object {
    dynamic var name = ""
    dynamic var text = ""
    dynamic var iconURL = ""
    dynamic var id = ""
    dynamic var createdAt = Date()

    dynamic var favorited = false

    static var dateFormatter: DateFormatter {
        ...
    }

    convenience init(tweetDictionary: [String: AnyObject]) {
        self.init()
        let user = tweetDictionary["user"] as! [String: AnyObject]
        name = user["name"] as! String
        text = tweetDictionary["text"] as! String

        iconURL = user["profile_image_url_https"] as! String

        id = tweetDictionary["id_str"] as! String
        createdAt = Tweet.dateFormatter.date(from: tweetDictionary["created_at"] as! String)!

        favorited = tweetDictionary["favorited"] as! Bool
    }

    override class func primaryKey() -> String? {
        return "id"
    }
}
```



kk@realm.io

# 4-2. Likeする

```
func getHomeTimeline() {
    ...
}

func postFavorite(_ id: String) {
    let realm = try! Realm()
    guard let tweet = realm.object(ofType: Tweet.self, forPrimaryKey: id) else {
        return
    }

    let requestURL: URL
    if tweet.favorited {
        requestURL = URL(string: "https://api.twitter.com/1.1/favorites/destroy.json")!
    } else {
        requestURL = URL(string: "https://api.twitter.com/1.1/favorites/create.json")!
    }

    let request = SLRequest(forServiceType: SLServiceTypeTwitter, requestMethod: .POST, url: requestURL, parameters: ["id": id])
    request?.account = account

    request?.perform { (data, response, error) -> Void in
        if let error = error {
            self.showAlert(error.localizedDescription)
            return
        }

        let realm = try! Realm()
        if let tweet = realm.object(ofType: Tweet.self, forPrimaryKey: id) {
            try! realm.write {
                tweet.favorited = !tweet.favorited
            }
        }
    }
}

func showAlert(_ message: String) {
    ...
}
```



kk@realm.io

# 4-3. Likeする

```
class TimelineViewController: UITableViewController {

    ...

    override func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return UITableViewAutomaticDimension
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        let tweet = timeline![indexPath.row]
        postFavorite(tweet.id)

        tableView.deselectRow(at: indexPath, animated: true)
    }

}
```

# 4-4. Likeする

```
import UIKit
import RealmSwift

class FavoritesViewController: UITableViewController {

    var likes: Results<Tweet>?
    var notificationToken: NotificationToken?

    override func viewDidLoad() {
        super.viewDidLoad()

        tableView.register(UINib(nibName: "TimelineCell", bundle: nil), forCellReuseIdentifier: "timelineCell")
        tableView.rowHeight = 90
        tableView.estimatedRowHeight = 90

        let realm = try! Realm()
        likes = realm.objects(Tweet.self).filter("favorited = %@", true).sorted(byKeyPath: "createdAt", ascending: false)
        notificationToken = likes?.addNotificationBlock { [weak self] (change) in
            switch change {
            case .initial(_):
                self?.tableView.reloadData()
            case .update(_, deletions: _, insertions: _, modifications: _):
                self?.tableView.reloadData()
            case .error(_):
                return
            }
        }
    }

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return likes?.count ?? 0
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "timelineCell", for: indexPath) as! TimelineCell

        let tweet = likes![indexPath.row]

        cell.nameLabel.text = tweet.name
        cell.tweetTextView.text = tweet.text

        URLSession.shared.dataTask(with: URLRequest(url: URL(string: tweet.iconURL)!)) { (data, response, error) -> Void in
            if let _ = error {
                return
            }

            DispatchQueue.main.async {
                let image = UIImage(data: data!)!
                cell.iconView.image = image
            }
        }.resume()

        return cell
    }

    override func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return UITableViewAutomaticDimension
    }

}
```

# Where to find us

- 🇯🇵 Realm Japan User Group: facebook.com/groups/realmjp
- 🇯🇵 Twitter: twitter.com/realmJapan
- GitHub: github.com/realm
- 🇯🇵 StackOverflow: ja.stackoverflow.com/questions/tagged/realm
- 🇯🇵 Email: help@realm.io
- 🇯🇵 Slack: slack.realm.io/

# Realm Japan User Group

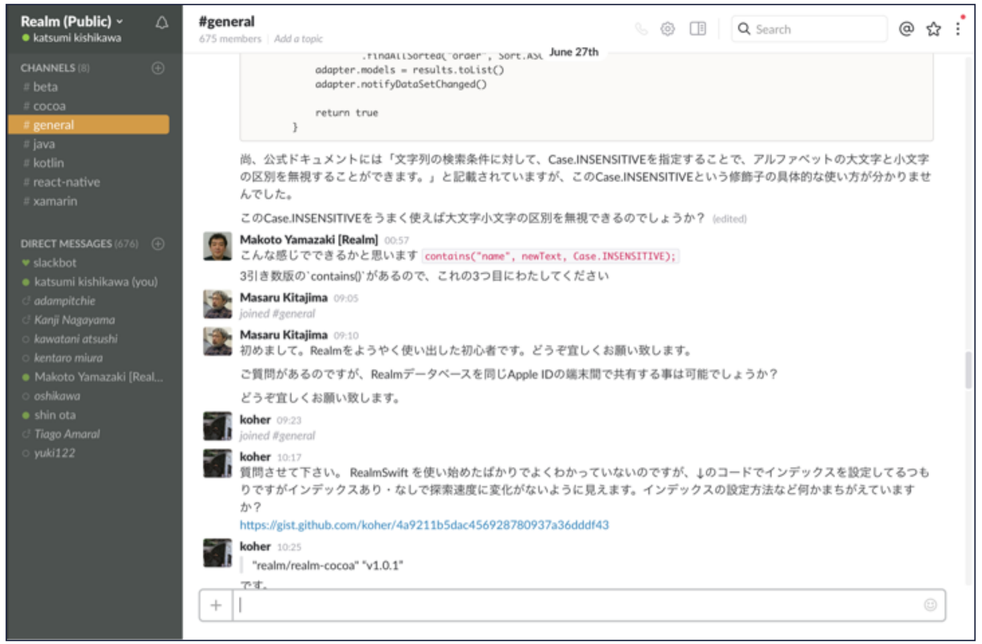
Facebook




kk@realm.io

# Support Chat

Slack






kk@realm.io

# Questions?

Katsuma Kishikawa
kk@realm.io
www.realm.io/jp
@k_katsumi