Dave DeLong
Principal iOS Engineer

Designing Accessible APIs

wework

Accessible API

wework

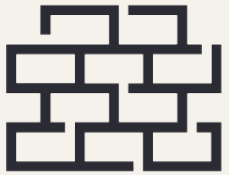
Access-ible API

wework

The Ability to Access

Accessible Apps

- Anyone can access, regardless of:
  - Physical capacity
  - Language
  - Age



Accessible APIs

- Anyone can access, regardless of:
  - Experience
  - Skill level
  - Language

wework

Be Kind
To Yourself

wework

API Lasts Forever

- Expect API to be used in 10+ years
- What is the *minimum* API required?
- What existing code & types can you re-use?

```
typealias TimeRange = Range<Time>

// Times always have a "seconds" value of :00
struct Time: Comparable {
    let hour: Int
    let minute: Int
}

let timeRange: TimeRange = ...
```

wework

API Lasts Forever

- Expect API to be used in 10+ years
- What is the *minimum* API required?
- What existing code & types can you re-use?

```
let timeRange: DateInterval = ...
```

wework

API Lasts Forever

- Expect API to be used in 10+ years
- What is the *minimum* API required?
- What existing code & types can you re-use?
- Avoid “clever” code

```
let timeRange: DateInterval = ...
```

wework

Everyone knows that debugging is twice as hard as writing a program in the first place.

So if you're as clever as you can be when you write it, how will you ever debug it?

Brian Kernighan

Co-Author, “The C Programming Language”

wework

Duplication is far cheaper

than the wrong abstraction

Sandi Metz

https://www.sandimetz.com/blog/2016/1/20/the-wrong-abstraction

wework

Prevent Future Mistakes

- Regressions make your life hard
  - Tests prevent regressions
  - Test as much as you can

```
func test_WhenConnecting_RetrieveCredentials() {
    let e = expectation()
    credentialRetrievalCallback = { e.fulfill() }
    userConnection.requestCurrentUser { _ in }
    wait(for: e)
}
```

wework

Prevent Future Mistakes

- Regressions make your life hard
  - Tests prevent regressions
  - Test as much as you can

```
func test_WhenConnecting_RetrieveCredentials() {
    let e = expectation()
    credentialRetrievalCallback = { e.fulfill() }
    userConnection.requestCurrentUser { _ in }
    wait(for: e)
}
```

- Assert expected behavior
  - dispatchPrecondition()
  - assert()
  - precondition()

```
private func startCollectingGarbage(_ timer: Timer) {
    assert(timer == _timer)
    dispatchPrecondition(condition: .onQueue(.main))

    timer.invalidate()
    queue.async { self.collectGarbage() }
}
```

wework

Be Kind to Yourself

- API lasts forever
- Prevent future mistakes

wework

02

Be Kind
To Users

wework

Remove More Work Than You Add

- "One-click Install"
- Follow platform conventions
  - Naming, Architecture, Style, ...
- Use common types and patterns

```
typealias RequestHandler<T> = (Result<T, RequestError>) -> Void

public func requestCurrentUser(
    completion: @escaping RequestHandler<User>) -> Cancellable {
    …
}
```

wework

Build For the Common Case

- Understand what 90% of clients want to do
- Make common usage the default
- Provide freedom *from* choice
- Be flexible
- Documentation

```
/// Retrieve all values that match a provided `Filter`.
///
/// - Parameters:
///   - filter: The `Filter<T>` describing the constraints to be used when fetching
///     matching values. If a filter is specified for a type that does not support
///     filtering, then the completion handler will be immediately invoked with
///     an `.invalidRequest` error code.
///   - completion: A completion handler that will be invoked with either the array of
///     retrieved values, or a `RequestError` describing the first failure encountered
///     when retrieving the values.
/// - Returns: A `Cancellable` via which the entire operation may be prematurely stopped.
@discardableResult
func requestAll<T: ModelType>(matching filter: Filter<T>?, completion: @escaping RequestHandler<Array<T>>) -> Cancellable {
```

wework

Be Opinionated

- Provide a higher-level abstraction
- Focus on your domain; say “no” to out-of-scope requests
- Be opinionated about that abstraction
- Avoid opinions on everything else
- Disallow unwanted code

```
public func requestCurrentUser(completion: @escaping RequestHandler<User>) -> Cancellable {
    let r = Request.requestForCurrentUser
    return self.execute(request: r, completion: completion)
}

internal func execute<R>(request: Request<R>,
                         completion: @escaping RequestHandler<R>) -> Cancellable {
    return requestClient.requestSingle(request, completion: completion)
}
```

wework

Don’t Force Unwanted Decisions

- Avoid transitive dependencies
- Deprecate; don’t remove
- Update deprecated implementations
- Require “opting-in” to new features
- Don’t crash

```
guard let url = URL(string: request.urlString) else {
    delegateQueue.addOperation {
        let error = HTTPError(code: .invalidRequest, request: request)
        completion(.failure(error))
    }
    return
}
```

wework

Be Kind to Users

- Remove more work than you add
- Build for the common case
- Be opinionated
- Don’t force unwanted decisions

wework

Accessibility

Empathy for yourself

Empathy for others

wework

Thank You

Dave DeLong

Principal iOS Engineer

dave.delong@wework.com

wework