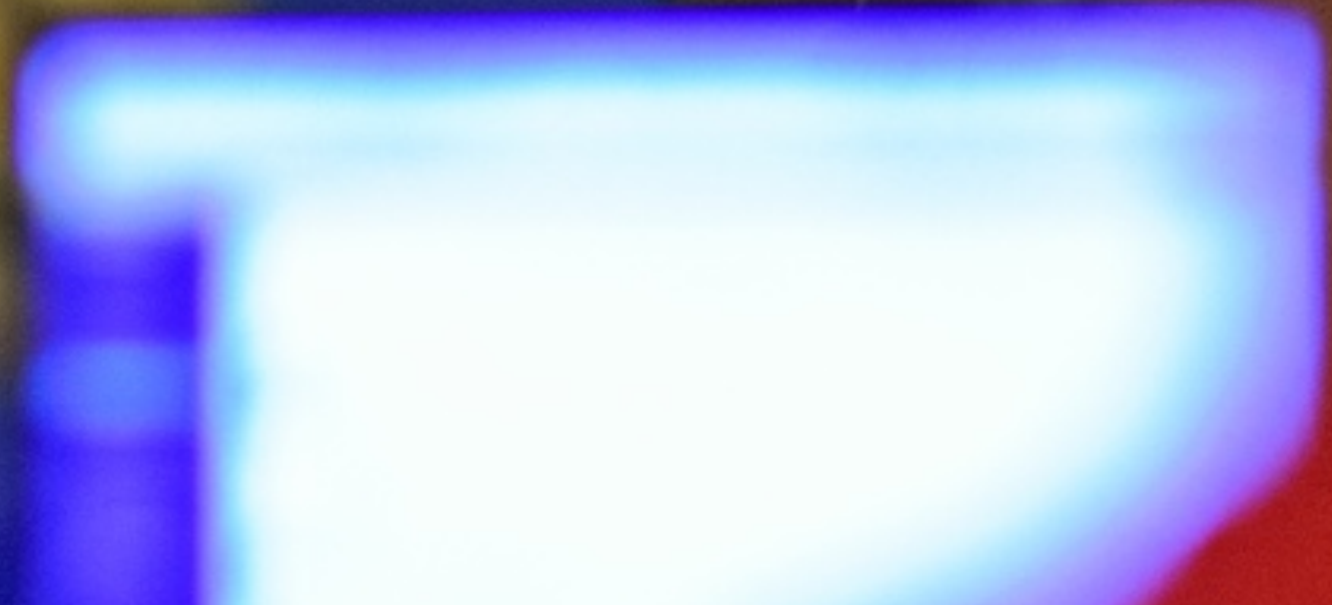


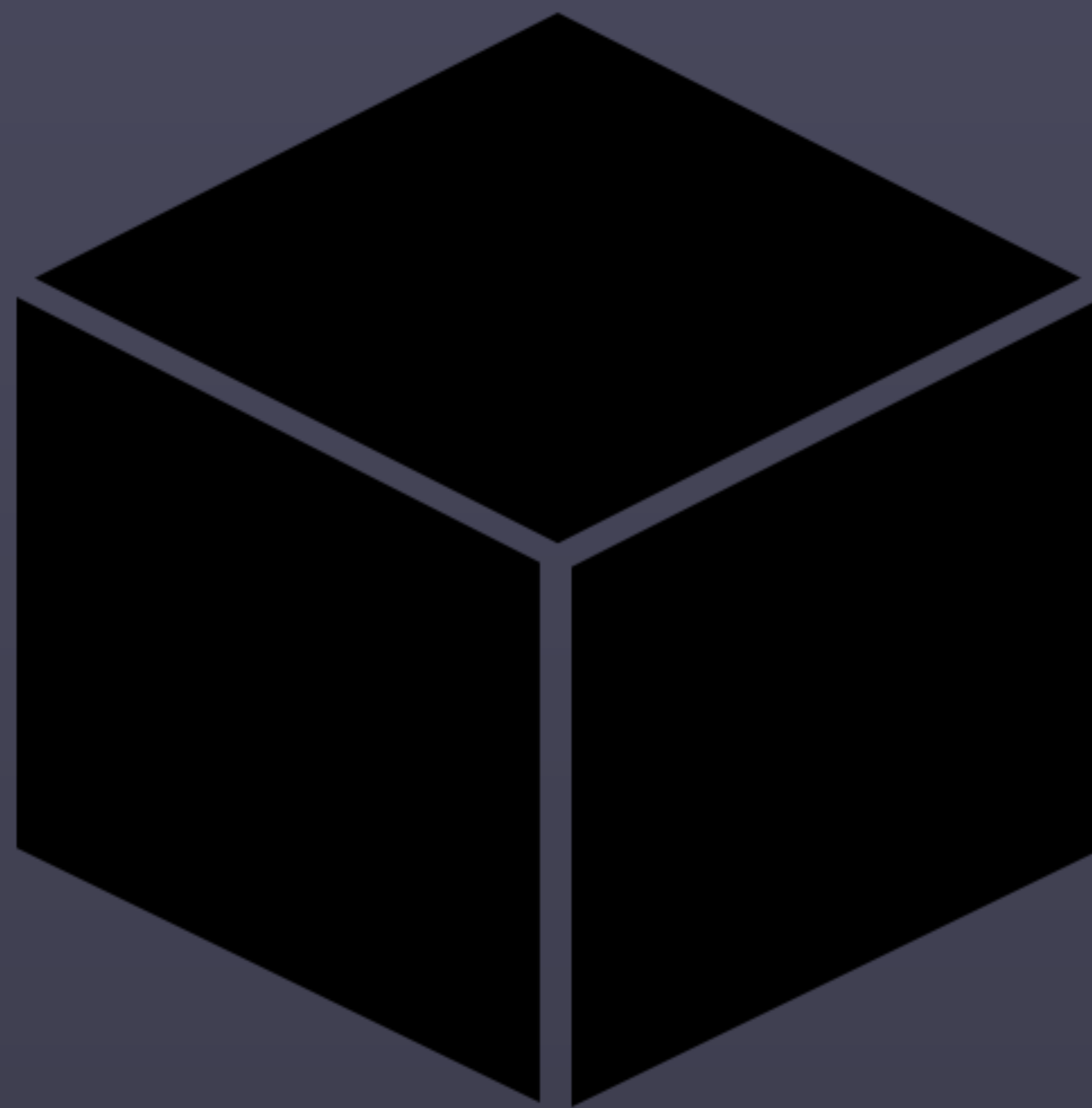
Designing a Modern Swift Network Stack

Philly CocoaHeads • January 10, 2019

Hello



My Client Story

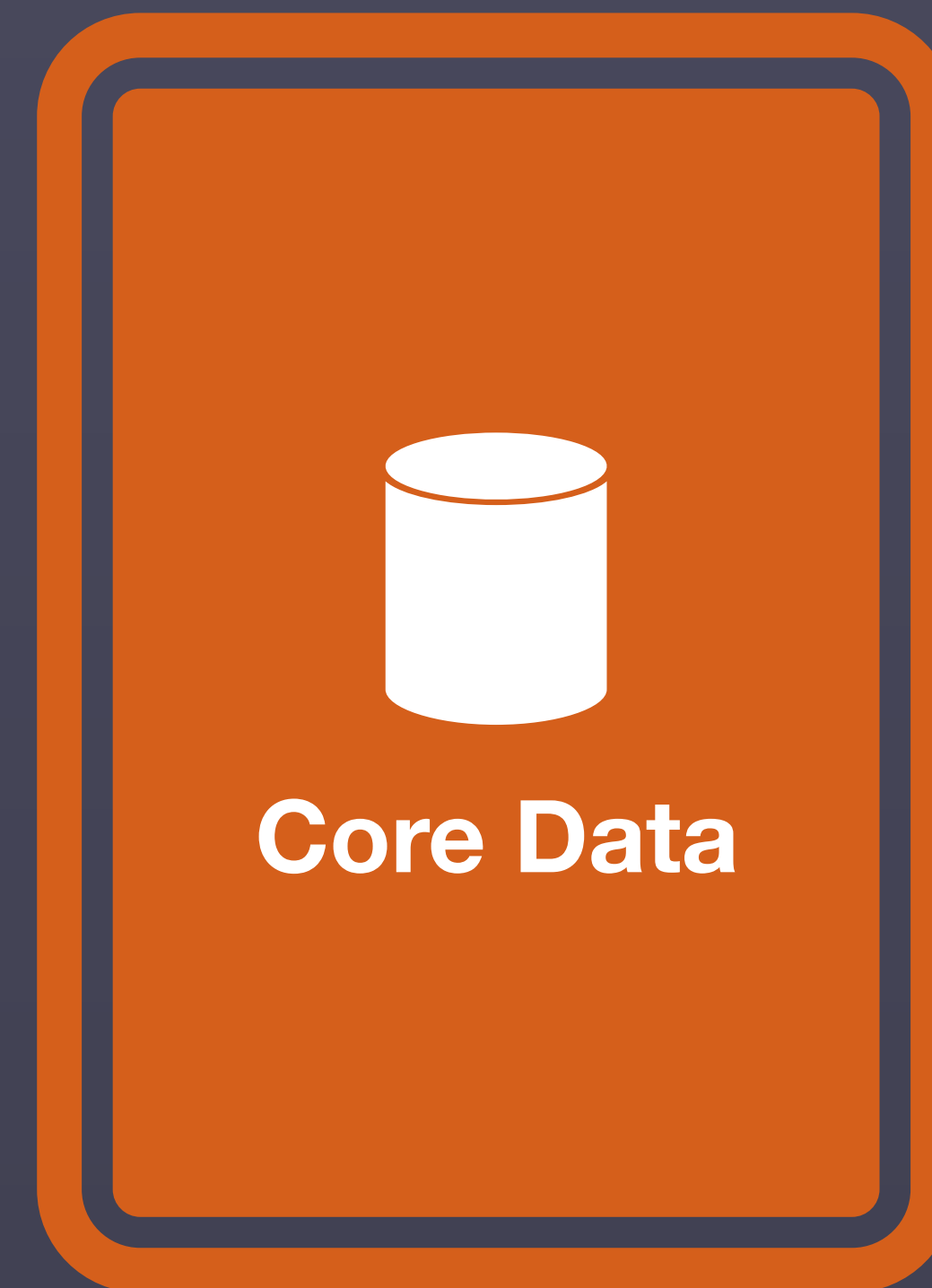




Company API



APIClient + MapModelManager
2000 lines each



App

Problems

- **Cumbersome to Add and Refactor Code**
- **Lack of Documentation**
- **Lack of Unit Testing**
- **OAuth2 Token Renewal Bugs**
- **Lack of Respect for Current User State**

Wishlist

- **Break Problem into Smaller, Testable Parts**
- **Less Reliance on Global Data Management**
- **Different Environments (local, stage, production)**
- **Event Logging (For Debugging and Analytics)**
- **Bulletproof OAuth Token Renewal**

Wishlist

- **Simulate Network Responses to Demo App Scenarios**
- **Chain Dependent Network Requests**
- **Group Related Parallel Network Requests**
- **Cancel Network Requests**
- **System to Humanize All Possible Error Messages**

Wishlist

- **Draw Firm Lines for Breaking Down Responsibilities**
- **Utilize System Network Caching instead of Core Data**
- **Keep Logic and Resources as D.R.Y. as possible.**
- **Make Typical Use Cases as Simple as Possible.**
- **Deliver as a Sharable Framework**

The Solution

Small, focused, single responsibility objects that work together.

Request / Response

Request

```
protocol RequestDescribing {  
    var method: HTTPMethod { get }  
    var path: String { get }  
    var queryItems: [URLQueryItem]? { get }  
    var headers: [String: String]? { get }  
    var body: Data? { get }  
    var responseType: ResponseDescribing.Type { get }  
}
```

Places Request

```
struct FetchPlacesRequest: RequestDescribing {
    let method: HTTPMethod = .get
    let path = "/v2/places"
    let queryItems: [URLQueryItem]? = nil
    let headers: [String: String]? = nil
    let body: Data? = nil
    let responseType: ResponseDescribing.Type = FetchPlacesResponse.self
}
```

```
struct FetchClientSecretRequest: RequestDescribing {  
  
    let method: HTTPMethod = .post  
    let path = "/users/retrieve-client"  
    let queryItems: [URLQueryItem]? = nil  
    let headers: [String: String]? = RequestDefaults.JSONApplicationHeader  
    var body: Data? {  
        let values = [  
            "email": email,  
            "password": password],  
        ]  
        return try! JSONEncoder().encode(values)  
    }  
    let responseType: ResponseDescribing.Type = FetchClientSecretResponse.self  
  
    let email: String  
    let password: String  
    init(email: String, password: String) {  
        self.email = email  
        self.password = password  
    }  
}
```

Response

```
protocol ResponseDescribing {  
    var httpResponse: HTTPURLResponse { get }  
    init(data: Data?, httpResponse: HTTPURLResponse) throws  
}
```


Places Response

```
struct FetchPlacesResponse: ResponseDescribing {  
  
    let httpURLResponse: HTTPURLResponse  
    let places: [Place]  
  
    init(data: Data?, httpURLResponse: HTTPURLResponse) throws {  
        // Error Handling Cut For Space  
  
        let response = try JSONDecoder().decode(NetworkResponse.self, from: data)  
        self.httpURLResponse = httpURLResponse  
        self.places = response.data  
    }  
}  
  
private struct NetworkResponse: Codable {  
    let data: [Place]  
}
```

Server Configuration & Server Connection

Server Configuration

```
protocol ServerConfiguration {
    var host: URL { get }
}

struct ProductionConfiguration: ServerConfiguration {
    let host = URL(string: "https://api.example.com")!
}

struct StagingConfiguration: ServerConfiguration {
    let host = URL(string: "https://staging-api.example.com")!
}
```

Server Connection

```
class ServerConnection {  
    let serverConfiguration: ServerConfiguration  
  
    init(configuration: ServerConfiguration) {  
        self.serverConfiguration = configuration  
    }  
  
    func execute(_ request: RequestDescribing, completion: @escaping  
                ((ResponseDescribing?, Error?) -> Void)) { ... }  
}
```

```
// Inside of PlacesViewController

let request = FetchPlacesRequest()
serverConnection?.execute(request, completion: { (response, error) in

    if let error = error {
        // present alert
        return
    }

    guard let fetchPlacesResponse = response as? FetchPlacesResponse else {
        // present alert
        return
    }

    self.places = fetchPlacesResponse.places
    // refresh UI
})
```

What have we gained so far?

- Describe the API across many small files, instead of one large file.
- Can create 1:1 test files per API file.
- Dynamically configure our server connection per environment.
- Small amount of work to add API Key “request signing”.
- Small amount of work to add logging and analytics.

Responsibilities

- **Requests**
 - **Defines the rules for each point of API engagement.**
 - **Can be initialized with attributes that influence a request.**

Responsibilities

- **Responses**
 - **Centralizes the decision making process on what a server response means.**
 - **Owns the deserialization process, turning server returned JSON blobs into a local, native object type.**

Responsibilities

- **ServerConnection**
 - **Processes requests at the request of the view controller.**
 - **Let's the view controller work with local, native business objects (and errors) instead of having to process network JSON responses.**

Error Message Provider

```
// Inside of PlacesViewController

let request = FetchPlacesRequest()
serverConnection?.execute(request, completion: { (response, error) in

    if let error = error {
        // present alert
        return
    }

    guard let fetchPlacesResponse = response as? FetchPlacesResponse else {
        // present alert
        return
    }

    self.places = fetchPlacesResponse.places
    // refresh UI
})
```

```
enum ServerConnectionError: Error {  
    /// Typically refers to an internal error; XRequest expects, XResponse.  
    case unexpectedResponse  
  
    /// Holds server error messages intended for user presentation.  
    case descriptiveServerError(String)  
  
    /// Holds the HTTP Status Code. .descriptiveServerError is  
    preferred over .httpError when possible.  
    case httpError(Int)  
}
```

```
if let error = error {
    let alert = UIAlertController(title: "Could not load places.", error: error)
    self.present(alert, animated: true, completion: nil)
    return
}

extension UIAlertController {

    convenience init(title: String, error: Error) {
        self.init(title: title, message: nil, preferredStyle: .alert)
        self.message = ErrorMessageProvider.errorMessageFor(error)
        self.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
    }

}
```

```
public class ErrorMessageProvider {  
  
    static func errorMessageFor(_ error: Error) -> String {  
        if let serverConnectionError = error as? ServerConnectionError {  
            return errorMessageForServerConnectionError(serverConnectionError)  
        } else {  
            return error.localizedDescription  
        }  
    }  
  
    static func errorMessageForServerConnectionError(_ error: ServerConnectionError)  
-> String {  
        switch error {  
        case .unexpectedResponse:  
            return "Unexpected Response"  
        case .descriptiveServerError(let message):  
            return message  
        case .httpError(let statusCode):  
            return "HTTP Error \(statusCode)"  
        }  
    }  
}
```

Authentication

What is OAuth 2?



Featured Video: [OAuth 2.0 Access Tokens Explained](#)

OAuth 2.0

OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 supersedes the work done on the original OAuth protocol created in 2006. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices. This specification and its extensions are being developed within the [IETF OAuth Working Group](#).



Questions, suggestions and protocol changes should be discussed on the [mailing list](#).

OAuth 2.0 Core

- [OAuth 2.0 Framework](#) - RFC 6749
- [OAuth 2.0 Grant Types](#)
 - [Authorization Code](#)
 - [Implicit](#)
 - [Password](#)
 - [Client Credentials](#)
 - [Device Code](#)
 - [Refresh Token](#)
- [OAuth 2.0 Bearer Tokens](#) - RFC 6750
- [Threat Model and Security Considerations](#) - RFC 6819
- [OAuth 2.0 Security Best Current Practice](#)

Mobile and Other Devices

- [Native Apps](#) - Recommendations for using OAuth 2.0 with native apps
- [PKCE](#) - Proof Key for Code Exchange, better security for native apps
- [Browser-Based Apps](#) - Recommendations for using OAuth 2.0 with browser-based apps (e.g. an SPA)
- [OAuth 2.0 Device Flow](#)

Token and Token Management

- [OAuth 2.0 Token Introspection](#) - RFC 7662, to determine the active state and meta-information of a token
- [OAuth 2.0 Token Revocation](#) - RFC 7009, to signal that a previously obtained token is no longer needed



OAuth 2.0 is the modern standard for securing access to APIs.

Read on for a complete guide to building your own authorization server.

[Learn about OAuth 2.0](#)

[Solve it with Okta](#)

Table of Contents

Background	9 Authorization	18 Token Introspection Endpoint
1 Getting Ready	10 Scope	19 Creating Documentation
2 Accessing Data in an OAuth Server	11 Redirect URLs	20 Terminology Reference
3 Signing in with Google	12 Access Tokens	21 Differences Between OAuth 1 and 2
4 Server-Side Apps	13 Listing Authorizations	22 OpenID Connect
5 Single-Page Apps	14 The Resource Server	23 IndieAuth
6 Mobile and Native Apps	15 OAuth for Native Apps	24 Map of OAuth 2.0 Specs
7 Making Authenticated Requests	16 OAuth for Browserless and Input-Constrained Devices	25 Tools and Libraries
8 Client Registration	17 Protecting Mobile Apps with PKCE	Appendix

Questions, suggestions and protocol changes should be discussed on the [mailing list](#).

OAuth 2.0 Core

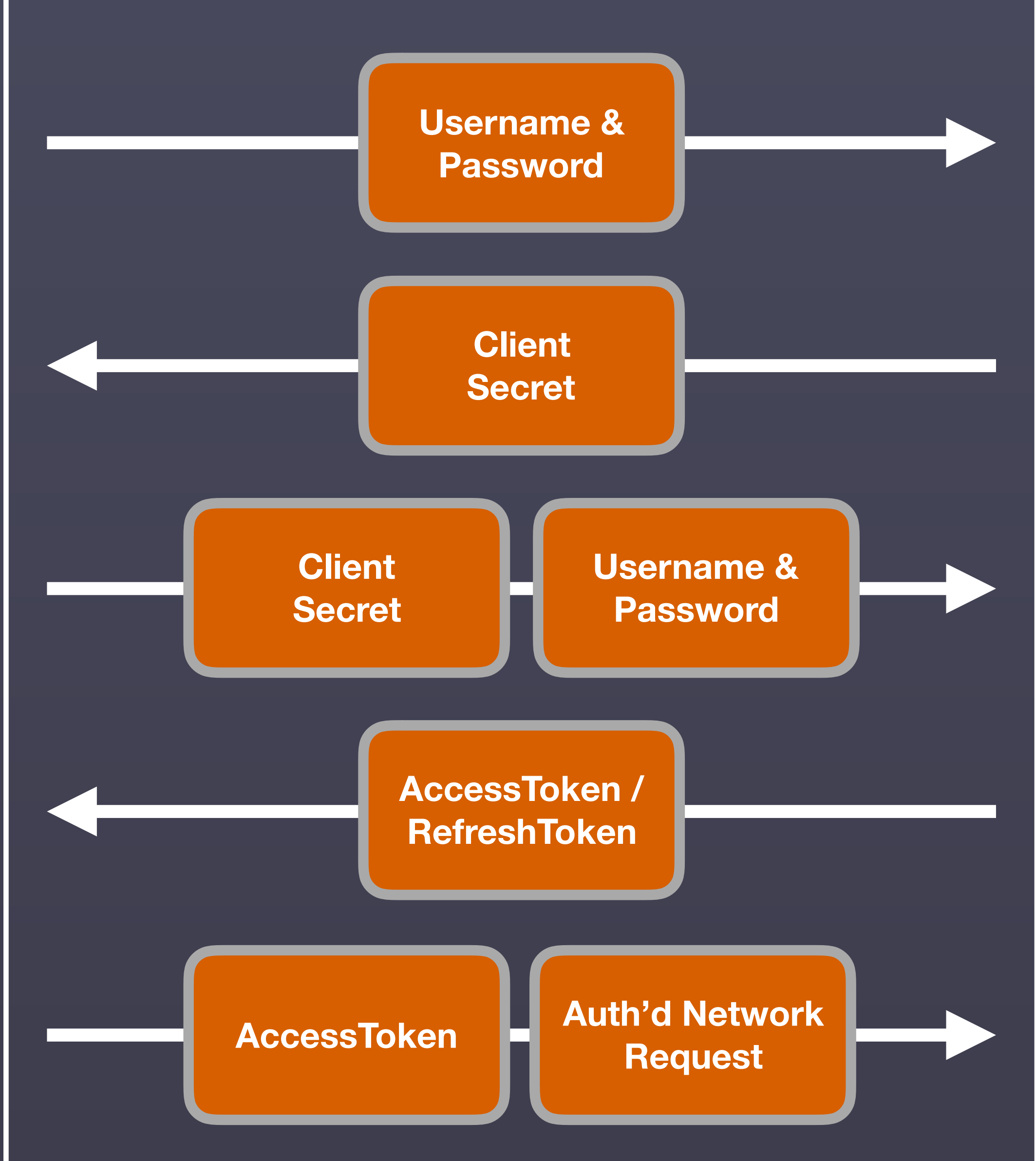
- [OAuth 2.0 Framework](#) - RFC 6749
- [OAuth 2.0 Grant Types](#)
 - [Authorization Code](#)
 - [Implicit](#)
 - [Password](#)
 - [Client Credentials](#)
 - [Device Code](#)
 - [Refresh Token](#)
- [OAuth 2.0 Bearer Tokens](#) - RFC 6750
- [Threat Model and Security Considerations](#) - RFC 6819
- [OAuth 2.0 Security Best Current Practice](#)

Mobile and Other Devices

- [Native Apps](#) - Recommendations for using OAuth 2.0 with native apps
- [PKCE](#) - Proof Key for Code Exchange, better security for native apps
- [Browser-Based Apps](#) - Recommendations for using OAuth 2.0 with browser-based apps (e.g. an SPA)

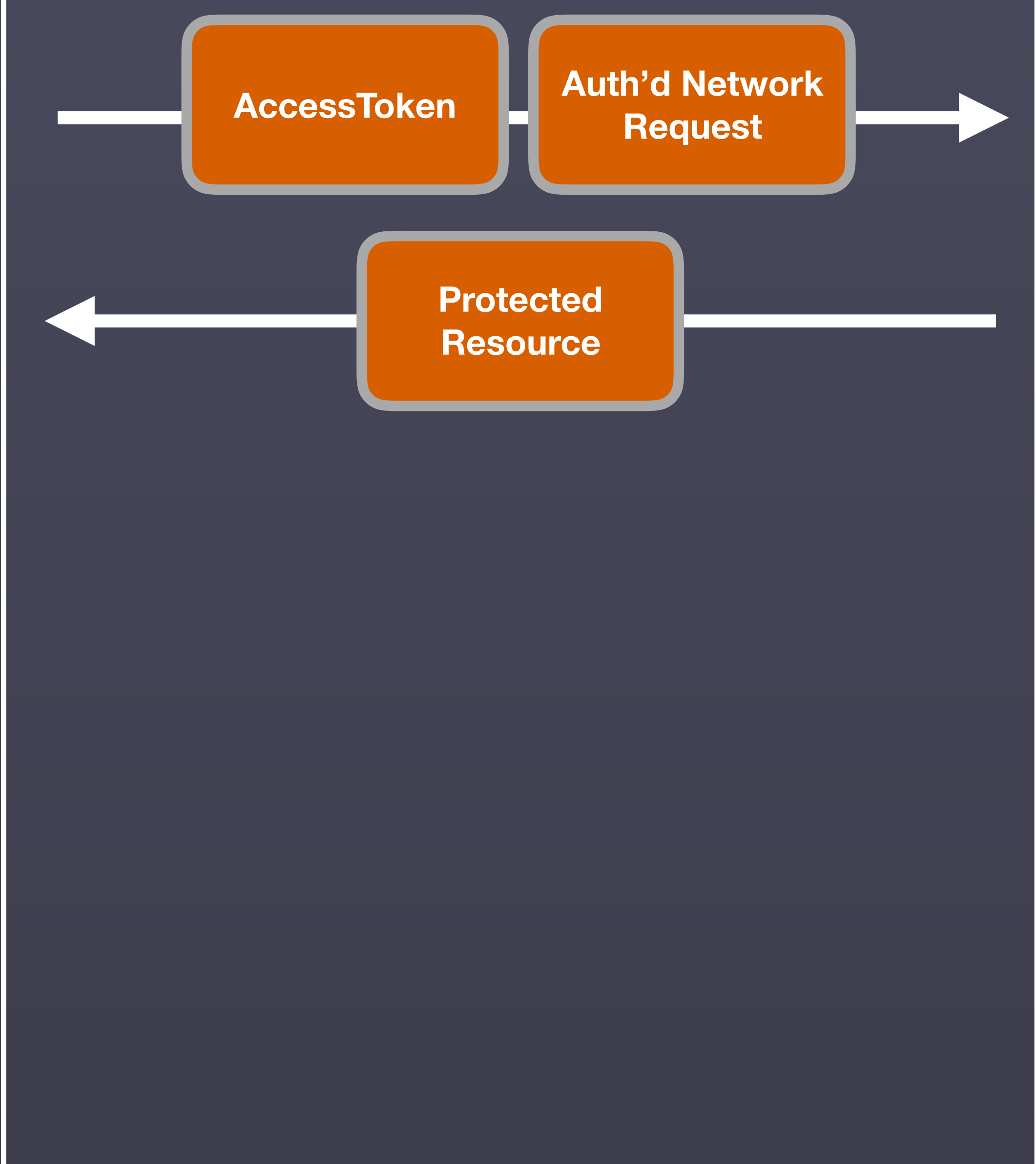
Mobile Client

Server



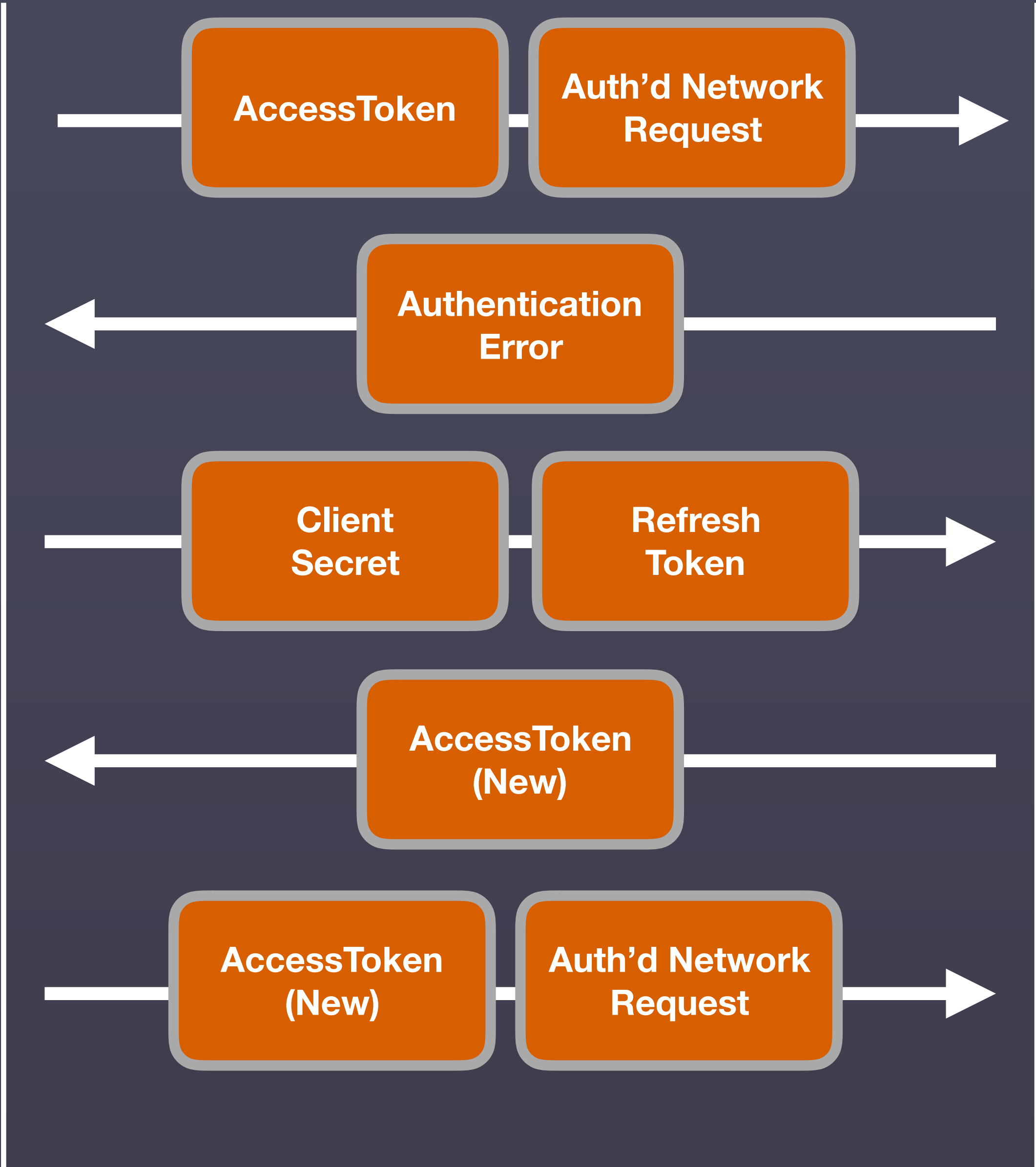
Mobile Client

Server



Mobile Client

Server



Session

Session

```
class Session {  
  let clientSecret: ClientSecret  
  let accessToken: AccessToken  
  
  init(clientSecret: ClientSecret, accessToken: AccessToken) {  
    self.clientSecret = clientSecret  
    self.accessToken = accessToken  
  }  
}
```


Session

```
struct ClientSecret: Equatable {  
    let id: String  
    let secret: String  
}
```

```
struct AccessToken: Codable {  
    let value: String  
    let type: String  
    let expiresAt: Date  
    let refreshToken: String  
}
```

ServerConnection

```
class ServerConnection {  
    private (set) var session: Session? {  
        didSet { postSessionDidChangeNotification() }  
    }  
  
    func login(session: Session) throws  
  
    func logout() throws  
  
}
```

RequestDescribing

```
protocol RequestDescribing {  
    var authenticationRequirement: AuthenticationRequirement { get }  
    // previously described  
}  
  
enum AuthenticationRequirement {  
    case none  
    case accessToken  
}
```

Interface Coordinator

Interface Coordinator

`sessionDidChange()`

**Authentication
Experience**

Window

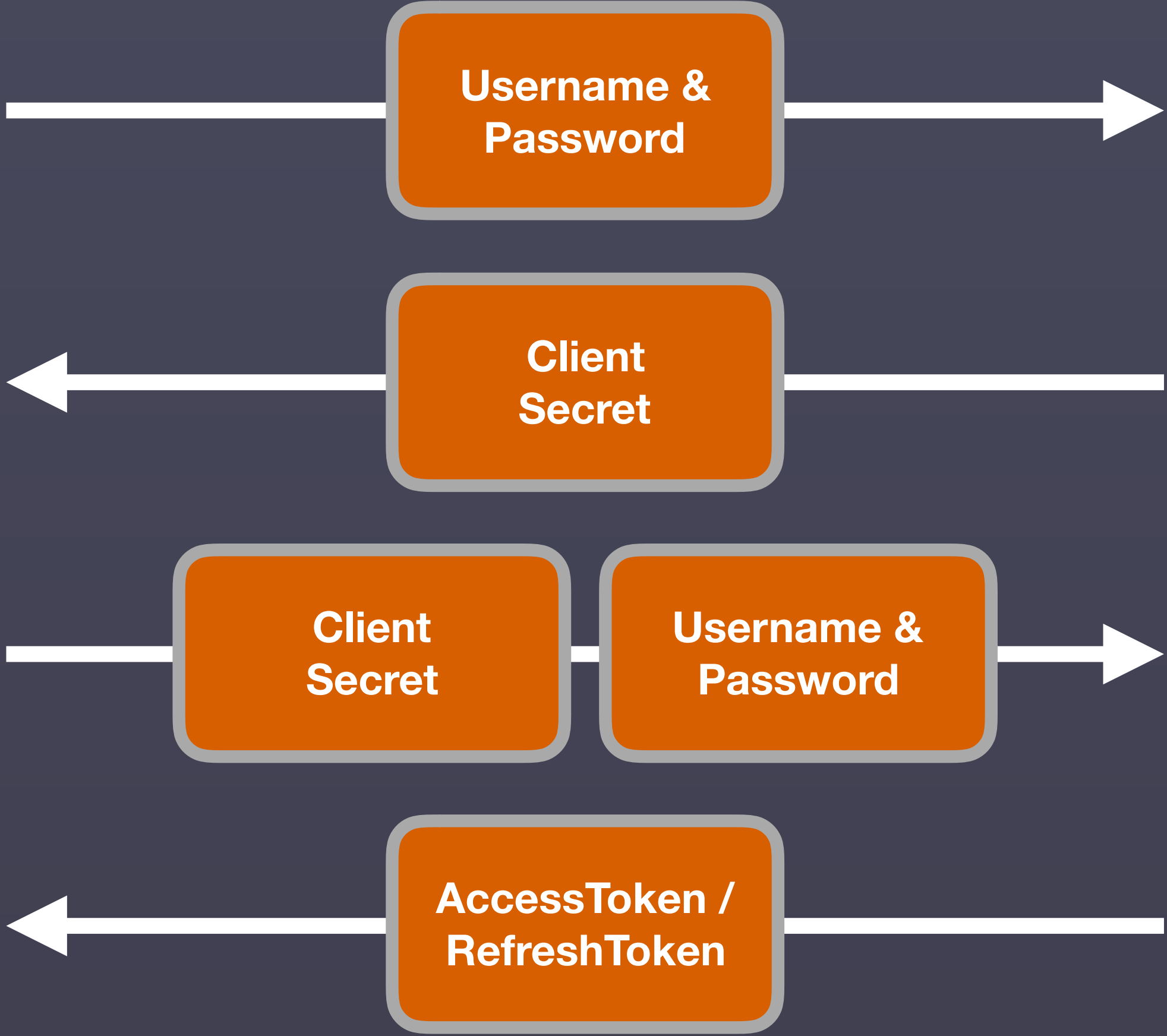
**Main App
Experience**

Making a New Session

Mobile Client

Server

**Dependent
Request
Chain**



```
// Inside of PlacesViewController

let request = FetchPlacesRequest()
serverConnection?.execute(request, completion: { (response, error) in

    if let error = error {
        // present alert
        return
    }

    guard let fetchPlacesResponse = response as? FetchPlacesResponse else {
        // present alert
        return
    }

    self.places = fetchPlacesResponse.places
    // refresh UI
})
```


Request / Response

Job / JobResult

```
let job = LoginJob(email: email, password: password)
serverConnection?.run(job, completion: { (jobResult, error) in
    if let error = error {
        return // Error Handling Cut For Space
    }

    guard let loginJobResult = jobResult as? LoginJobResult else {
        return // Error Handling Cut For Space
    }

    guard let session = loginJobResult.session else {
        return // Error Handling Cut For Space
    }

    // ServerConnection.login(session: session)
})
```

How do we implement?

```
run(_ job: Job)
```

**NSOperation /
NSOperationQueue**

AsyncOperation

```
class AsyncOperation: Operation {  
    // https://gist.github.com/parrots/f1a6ca9c9924905fd1bd12cfb640337a  
}
```

NetworkRequestOperation

```
class NetworkRequestOperation: AsyncOperation {  
    var request: RequestDescribing  
    var result: Result<ResponseDescribing>?  
    var accessToken: AccessToken?  
  
    init(request: RequestDescribing) {  
        self.request = request  
    }  
}  
  
enum Result<T> {  
    case success(T)  
    case failure(Error)  
}
```

Job Protocols

```
protocol Job {
    var resultType: JobResult.Type { get }
    var rootOperation: AsyncOperation { get }
}

protocol JobResult {
    init(operation: Operation) throws
}
```


LoginJob

```
struct LoginJob: Job {  
    let email: String  
    let password: String  
    var resultType: JobResult.Type = LoginJobResult.self  
    var rootOperation: AsyncOperation  
  
    init(email: String, password: String) {  
        self.email = email  
        self.password = password  
        self.rootOperation = LoginJobOperation(email: email, password:  
password)  
    }  
}
```

LoginJobOperation

```
class LoginJobOperation: AsyncOperation {  
  
    let email: String  
    let password: String  
  
    var result: Result<Session>?  
  
    init(email: String, password: String) {  
        // Skipped for space  
    }  
  
    override func main() {  
        // First does an inline NetworkRequestOperation for ClientSecret  
        // Then Build an inline NetworkRequestOperation for AccessToken  
        // Then packages Result, marks us as .finished  
    }  
}
```

LoginJobResult

```
struct LoginJobResult: JobResult {  
    let session: Session?  
  
    init(operation: Operation) throws {  
        // Cast operation as LoginJobOperation  
        // Pull the result out of LoginJobOperation  
  
        switch result {  
        case .failure(let error):  
            throw error  
        case .success(let session):  
            self.session = session  
        }  
    }  
}
```

```
let job = LoginJob(email: email, password: password)
serverConnection?.run(job, completion: { (jobResult, error) in
    if let error = error {
        return // Error Handling Cut For Space
    }

    guard let loginJobResult = jobResult as? LoginJobResult else {
        return // Error Handling Cut For Space
    }

    guard let session = loginJobResult.session else {
        return // Error Handling Cut For Space
    }

    // ServerConnection.login(session: session)
})
```

Job / JobResult



NetworkRequestOperation / AsyncOperation (Subclasses)



Request / Response

What have we gained?

- **Can perform chained requests to generate OAuth tokens.**
- **ServerConnection now owns the Session.**
- **Interface Coordinator now owns ServerConnection.**
 - **Also, listens for Session changes to manage UI.**
- **ViewControllers still have simple work abstraction but they can now run lots of network requests to be fulfilled.**
- **ServerConnection can now mark Authenticated Requests.**

Odds and Ends

ServerConnection.run()

```
func run(  
    _ job: Job,  
    completion: @escaping JobCompletion,  
    completionDispatchQueue: DispatchQueue = DispatchQueue.main)  
-> JobToken
```

```
func cancel(_ requestToken: JobToken)
```


More Odds and Ends

- **OAuth Token Renewal, part of `run(_ job: Job)`**
- **Persisting Log In**
 - **Storing Session in Keychain between app execution**
- **Network Caching**
 - **Increase disk cache sizes**
 - **Honor ETag and cache headers, instead of using Core Data**
- **Integration Testing with Custom URLRequest Protocols**

Regrets

- **Lots of files for a single API endpoint:**
 - **Request, Response, Job, JobResult, JobOperation, Custom AsyncOperation Subclass (complicated needs).**
 - **Can be lessened through Swagger code generation.**
- **Might be able to add-to NetworkRequestOperation to handle collections of Requests.**

Take Aways

- **Separate Design and Code Time.**
- **Draw and Sketch Out Systems Before Attempting To Code.**
- **Build Small, Focused, Single-Responsibility Objects.**
- **Make It Work, Make It Pretty, Make It Testable, Make It Documented.**
- **Iterate And Refactor Aggressively.**
- **Break Big Problems into Small Solutions Over Time.**

Recommended

- **Atlas, an unified approach to mobile development cycle: networking layer**
 - <https://medium.com/iquii/a5ccb064181a>
- **John Sundell: The Lost Art of System Design**
 - <https://www.youtube.com/watch?v=ujOc3a7Hav0>

Thanks

Mike Zornek
mike@mikezornek.com
@zorn (Micro.Blog)

Available for Consulting Projects
http://zornlabs.com