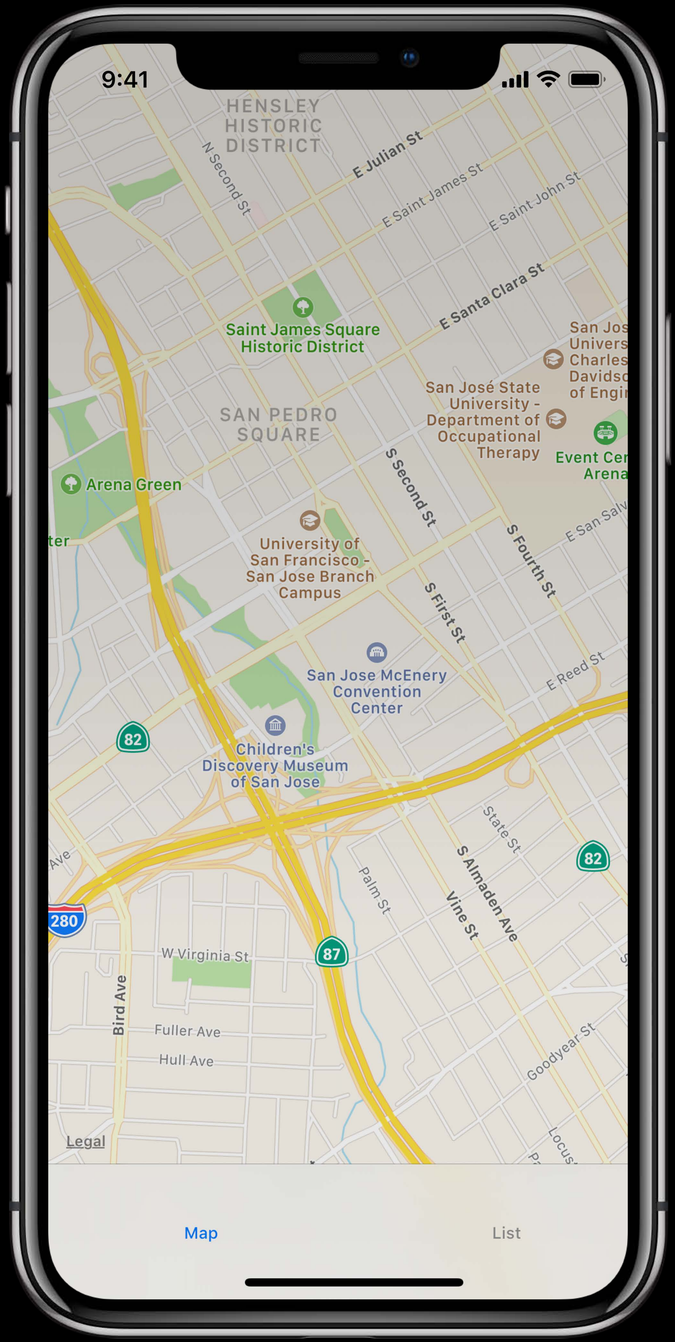
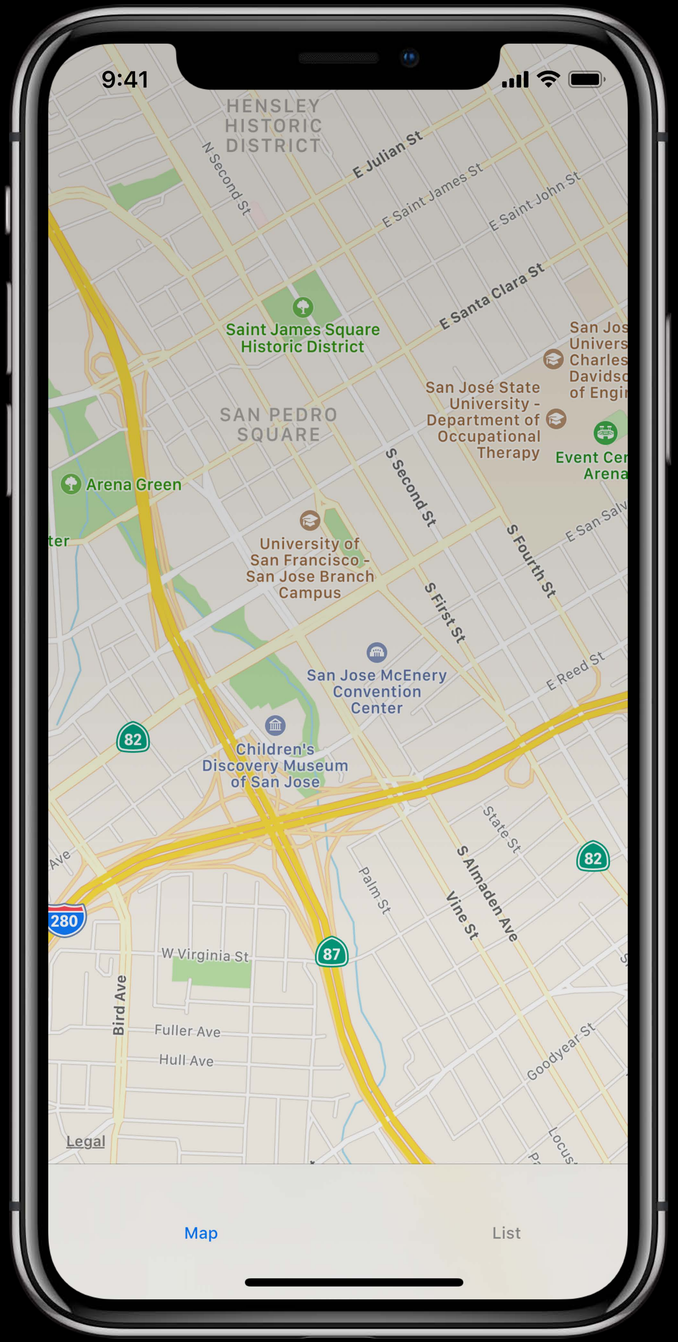
#WWDC18

Testing Tips & Tricks

Session 417

Brian Croom, XCTest Engineer
Stuart Montgomery, XCTest Engineer


9:41
HENSLEY HISTORIC DISTRICT
E Julian St
N Second St
E Saint James St
E Saint John St
E Santa Clara St
Saint James Square Historic District
SAN PEDRO SQUARE
San José State University - Department of Occupational Therapy
Arena Green
S Second St
S Fourth St
S First St
University of San Francisco - San Jose Branch Campus
San Jose McEnery Convention Center
E Reed St
Children's Discovery Museum of San Jose
82
280
87
W Virginia St
Bird Ave
Fuller Ave
Hull Ave
Palm St
Vine St
S Almaden Ave
State St
Goodyear St
Legal
Map
List


9:41
HENSLEY HISTORIC DISTRICT
E Julian St
N Second St
E Saint James St
E Saint John St
E Santa Clara St
Saint James Square Historic District
SAN PEDRO SQUARE
San José State University - Department of Occupational Therapy
Arena Green
S Second St
S Fourth St
S First St
University of San Francisco - San Jose Branch Campus
San Jose McEnery Convention Center
E Reed St
Children's Discovery Museum of San Jose
82
280
87
State St
Palm St
Vine St
S Almaden Ave
W Virginia St
Bird Ave
Fuller Ave
Hull Ave
Goodyear St
Legal
Map
List

Testing network requests

Working with notifications

Mocking with protocols

Test execution speed

Testing network requests

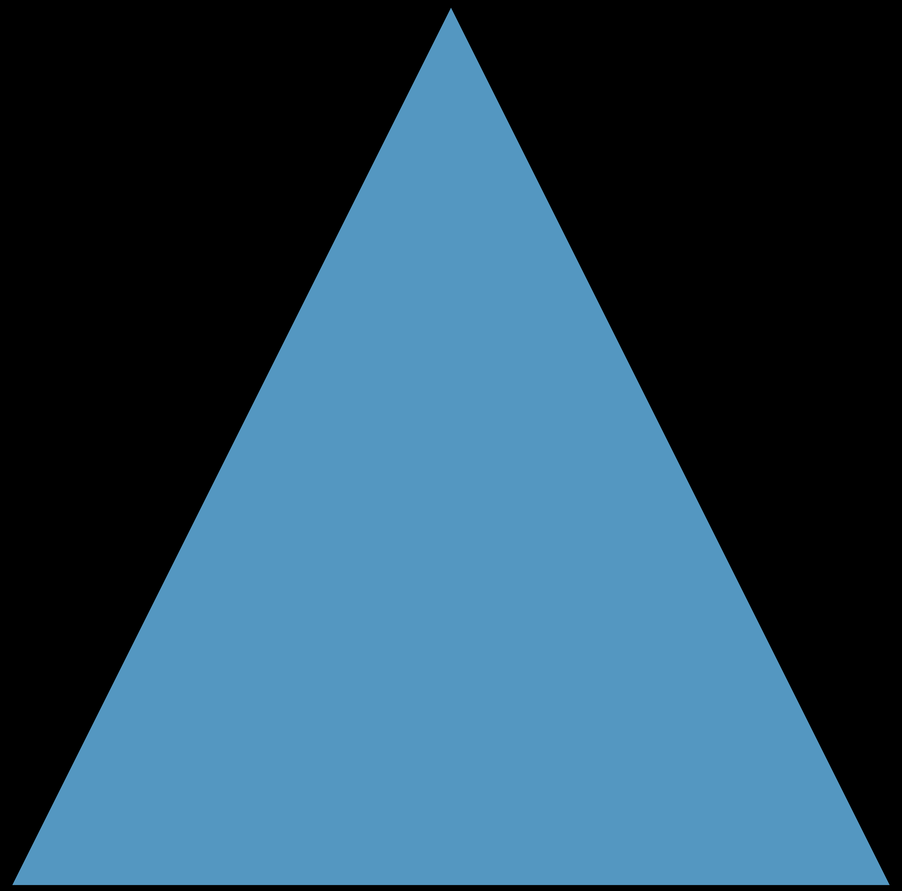
Working with notifications

Mocking with protocols

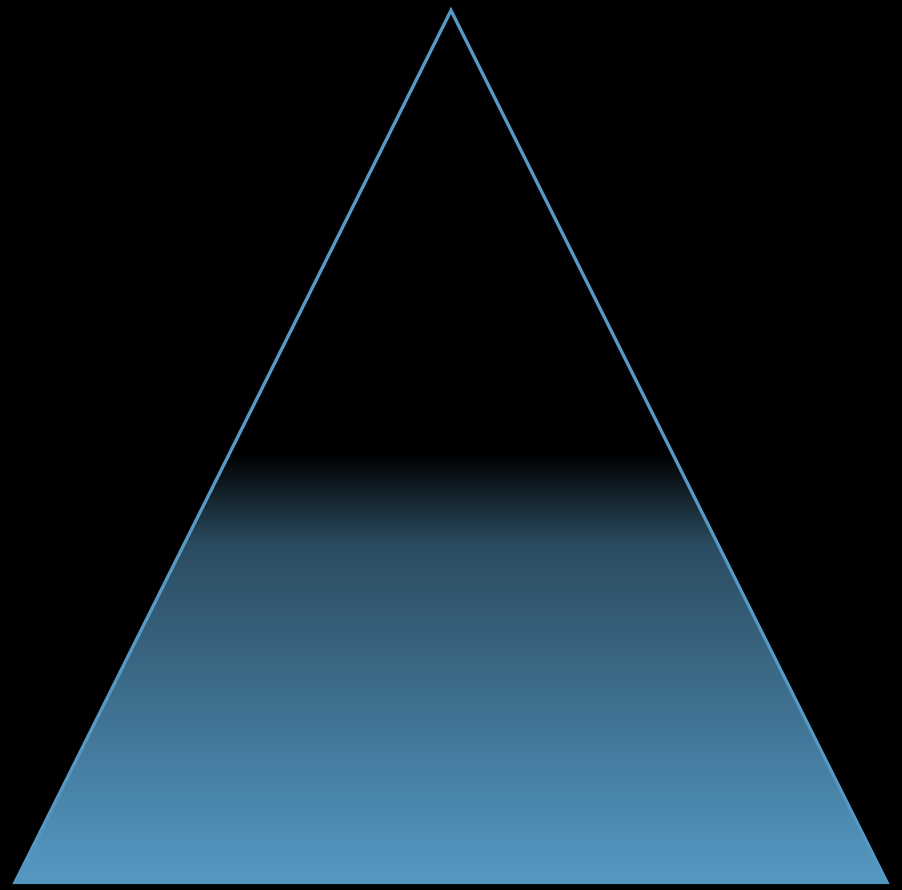
Test execution speed

Pyramid of Tests

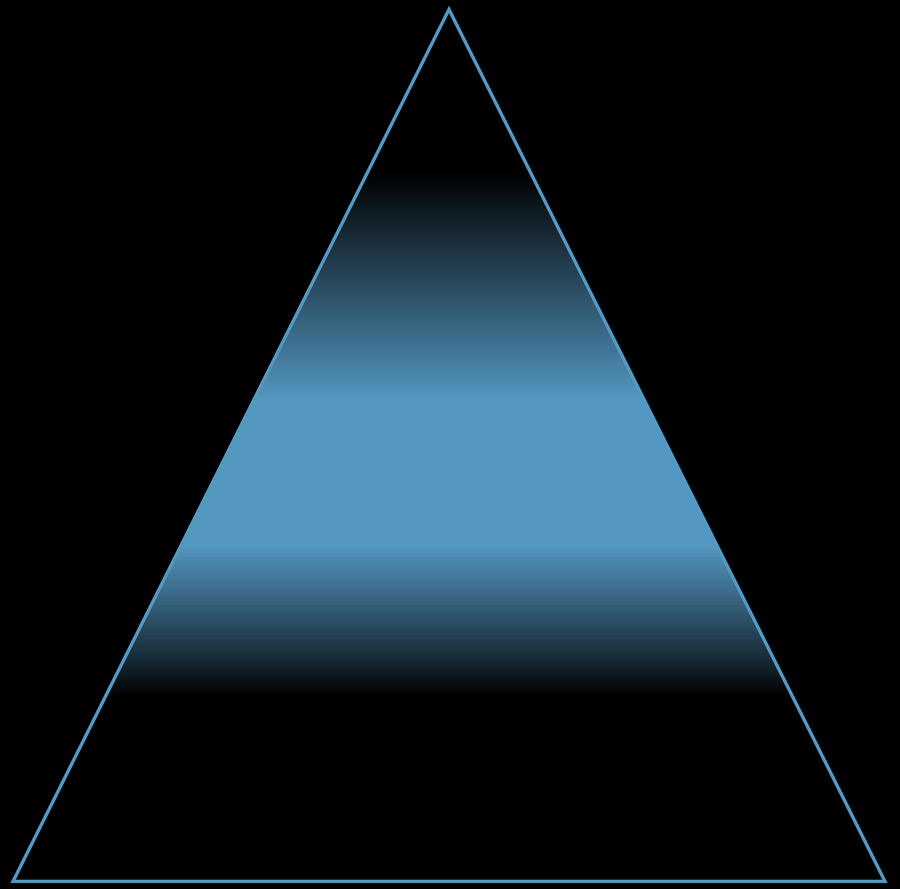
Recap



Pyramid of Tests

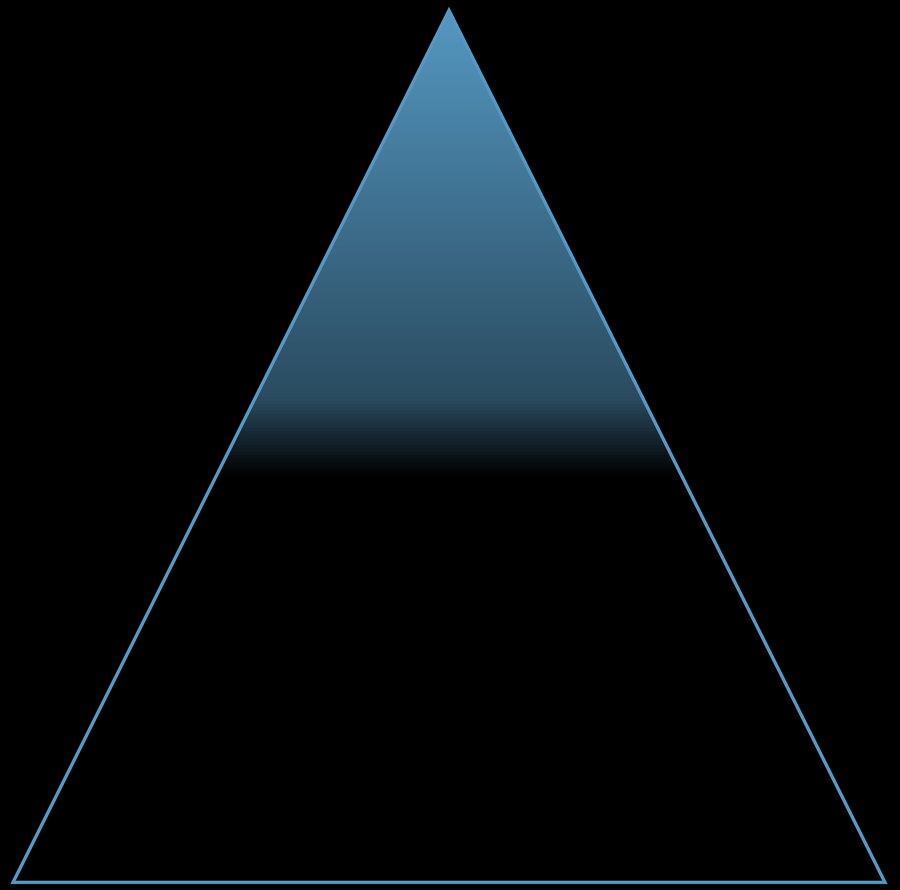


Pyramid of Tests



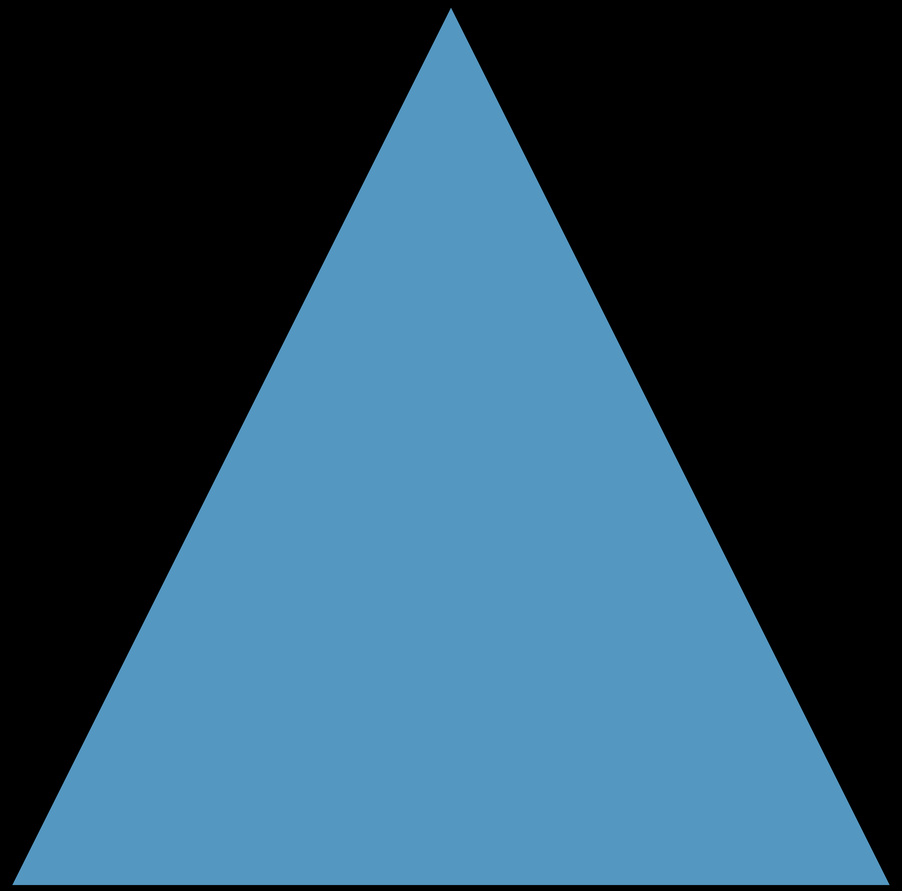
Integration

Pyramid of Tests



End-to-end

Pyramid of Tests

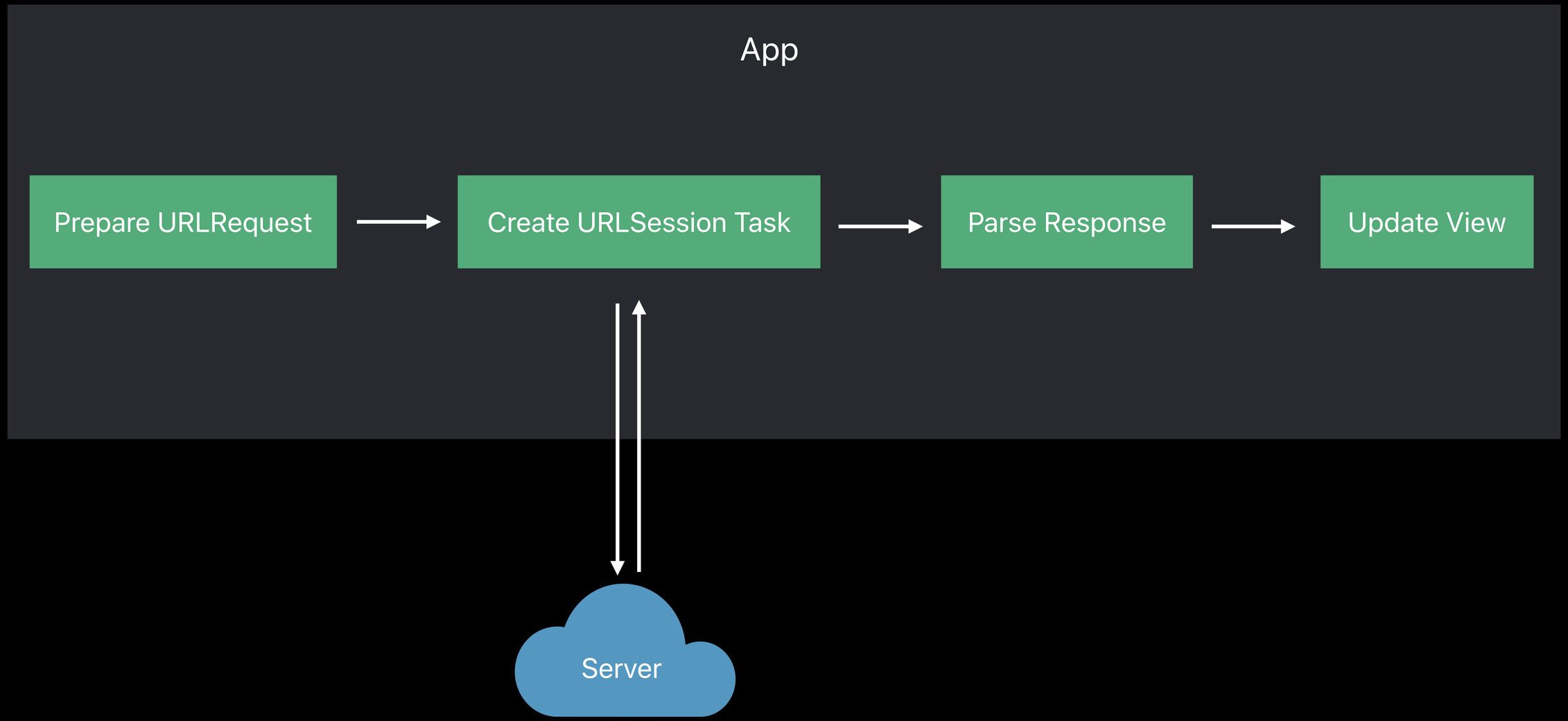


End-to-end

Integration

Unit

Networking Stack



```
func loadData(near coord: CLLocationCoordinate2D) {
    let url = URL(string: "/locations?lat=\(coord.latitude)&long=\(coord.longitude)")!
    URLSession.shared.dataTask(with: url) { data, response, error in
        guard let data = data else { self.handleError(error); return }
        do {
            let values = try JSONDecoder().decode([PointOfInterest].self, from: data)

            DispatchQueue.main.async {
                self.tableValues = values
                self.tableView.reloadData()
            }
        } catch {
            self.handleError(error)
        }
    }.resume()
}
```

```
func loadData(near coord: CLLocationCoordinate2D) {
    let url = URL(string: "/locations?lat=\(coord.latitude)&long=\(coord.longitude)")!
    URLSession.shared.dataTask(with: url) { data, response, error in
        guard let data = data else { self.handleError(error); return }
        do {
            let values = try JSONDecoder().decode([PointOfInterest].self, from: data)

            DispatchQueue.main.async {
                self.tableValues = values
                self.tableView.reloadData()
            }
        } catch {
            self.handleError(error)
        }
    }.resume()
}
```

```
func loadData(near coord: CLLocationCoordinate2D) {
    let url = URL(string: "/locations?lat=\(coord.latitude)&long=\(coord.longitude)")!
    URLSession.shared.dataTask(with: url) { data, response, error in
        guard let data = data else { self.handleError(error); return }
        do {
            let values = try JSONDecoder().decode([PointOfInterest].self, from: data)

            DispatchQueue.main.async {
                self.tableValues = values
                self.tableView.reloadData()
            }
        } catch {
            self.handleError(error)
        }
    }.resume()
}
```

```
func loadData(near coord: CLLocationCoordinate2D) {
    let url = URL(string: "/locations?lat=\(coord.latitude)&long=\(coord.longitude)")!
    URLSession.shared.dataTask(with: url) { data, response, error in
        guard let data = data else { self.handleError(error); return }
        do {
            let values = try JSONDecoder().decode([PointOfInterest].self, from: data)

            DispatchQueue.main.async {
                self.tableValues = values
                self.tableView.reloadData()
            }
        } catch {
            self.handleError(error)
        }
    }.resume()
}
```

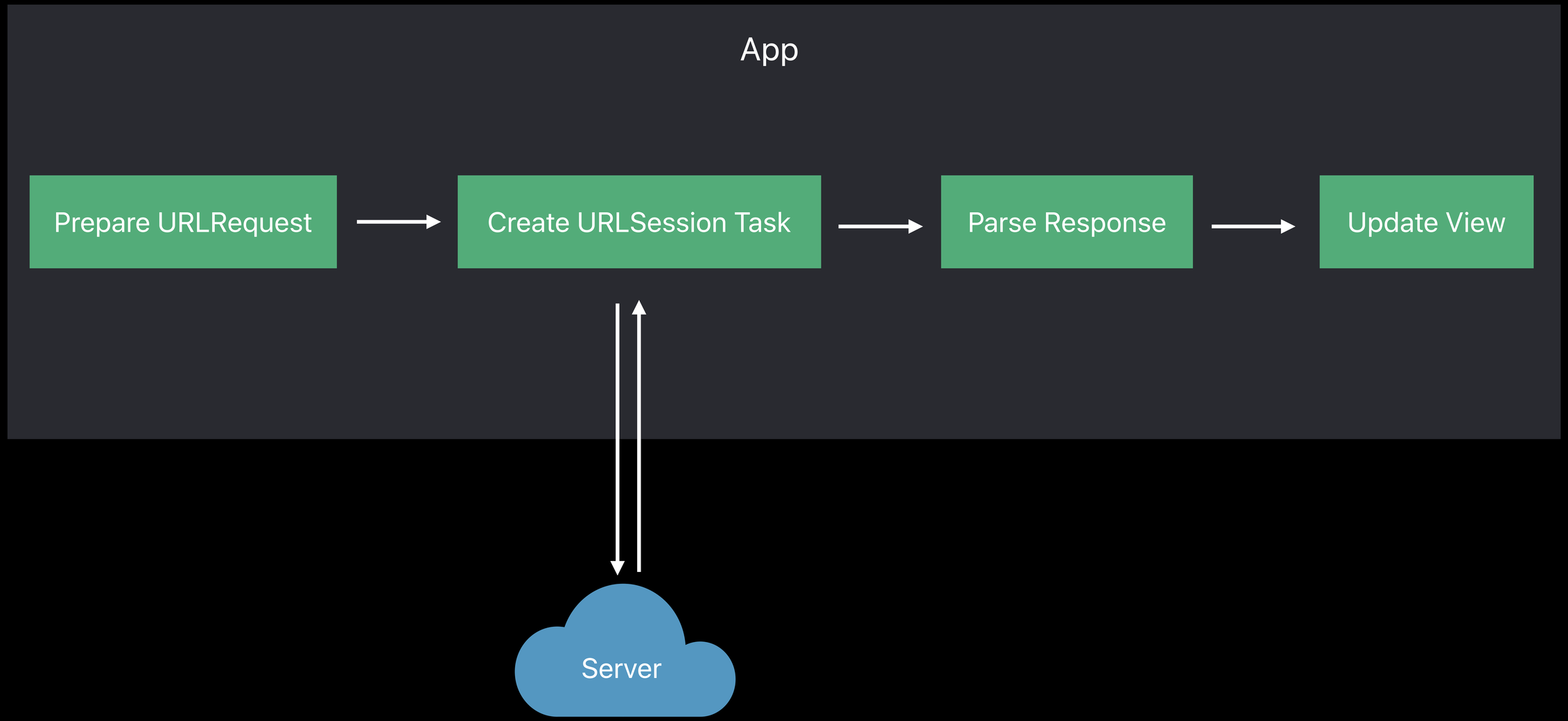
```
func loadData(near coord: CLLocationCoordinate2D) {
    let url = URL(string: "/locations?lat=\(coord.latitude)&long=\(coord.longitude)")!
    URLSession.shared.dataTask(with: url) { data, response, error in
        guard let data = data else { self.handleError(error); return }
        do {
            let values = try JSONDecoder().decode([PointOfInterest].self, from: data)

            DispatchQueue.main.async {
                self.tableValues = values
                self.tableView.reloadData()
            }
        } catch {
            self.handleError(error)
        }
    }.resume()
}
```

```
func loadData(near coord: CLLocationCoordinate2D) {
    let url = URL(string: "/locations?lat=\(coord.latitude)&long=\(coord.longitude)")!
    URLSession.shared.dataTask(with: url) { data, response, error in
        guard let data = data else { self.handleError(error); return }
        do {
            let values = try JSONDecoder().decode([PointOfInterest].self, from: data)

            DispatchQueue.main.async {
                self.tableValues = values
                self.tableView.reloadData()
            }
        } catch {
            self.handleError(error)
        }
    }.resume()
}
```

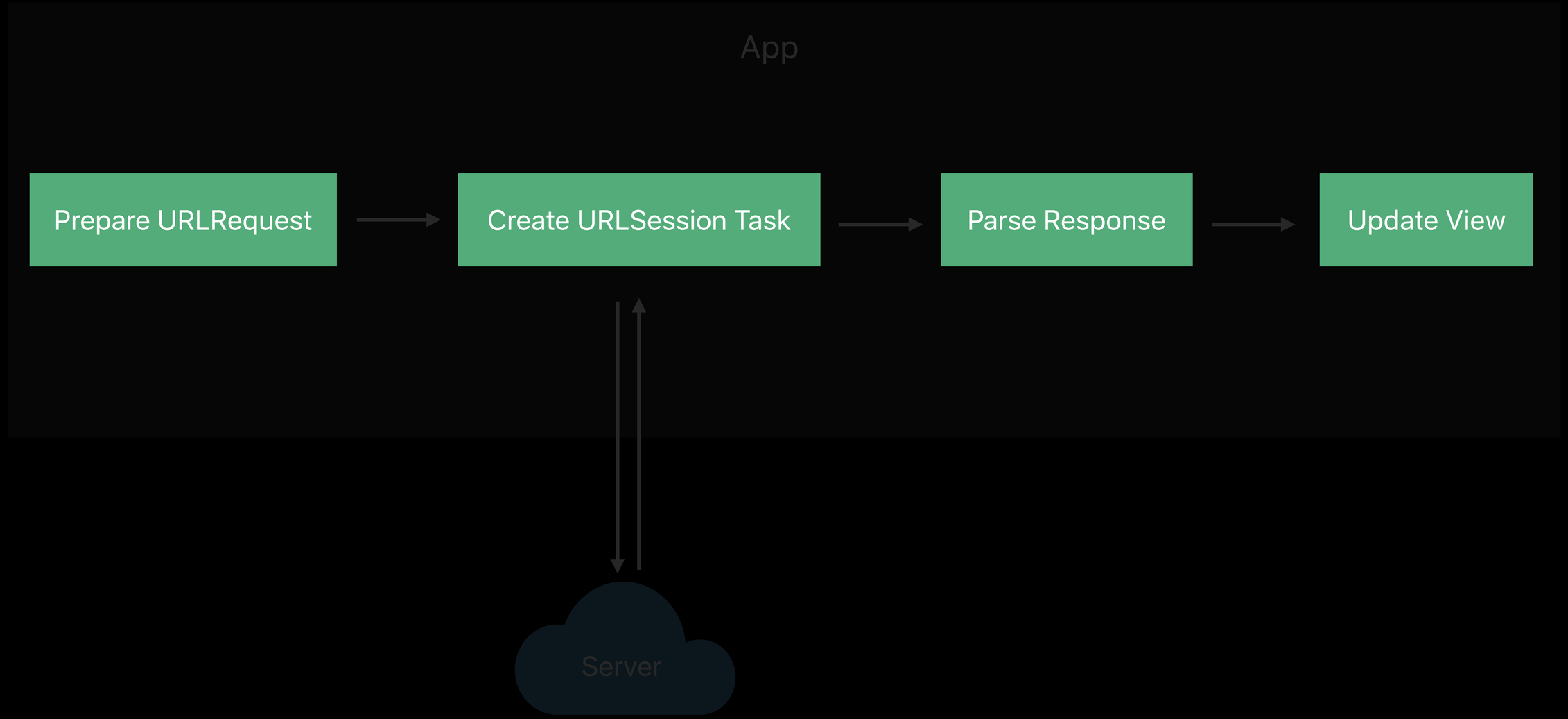
```
func loadData(near coord: CLLocationCoordinate2D) {
    let url = URL(string: "/locations?lat=\(coord.latitude)&long=\(coord.longitude)")!
    URLSession.shared.dataTask(with: url) { data, response, error in
        guard let data = data else { self.handleError(error); return }
        do {
            let values = try JSONDecoder().decode([PointOfInterest].self, from: data)

            DispatchQueue.main.async {
                self.tableValues = values
                self.tableView.reloadData()
            }
        } catch {
            self.handleError(error)
        }
    }.resume()
}
```

```
func loadData(near coord: CLLocationCoordinate2D) {
    let url = URL(string: "/locations?lat=\(coord.latitude)&long=\(coord.longitude)")!
    URLSession.shared.dataTask(with: url) { data, response, error in
        guard let data = data else { self.handleError(error); return }
        do {
            let values = try JSONDecoder().decode([PointOfInterest].self, from: data)

            DispatchQueue.main.async {
                self.tableValues = values
                self.tableView.reloadData()
            }
        } catch {
            self.handleError(error)
        }
    }.resume()
}
```

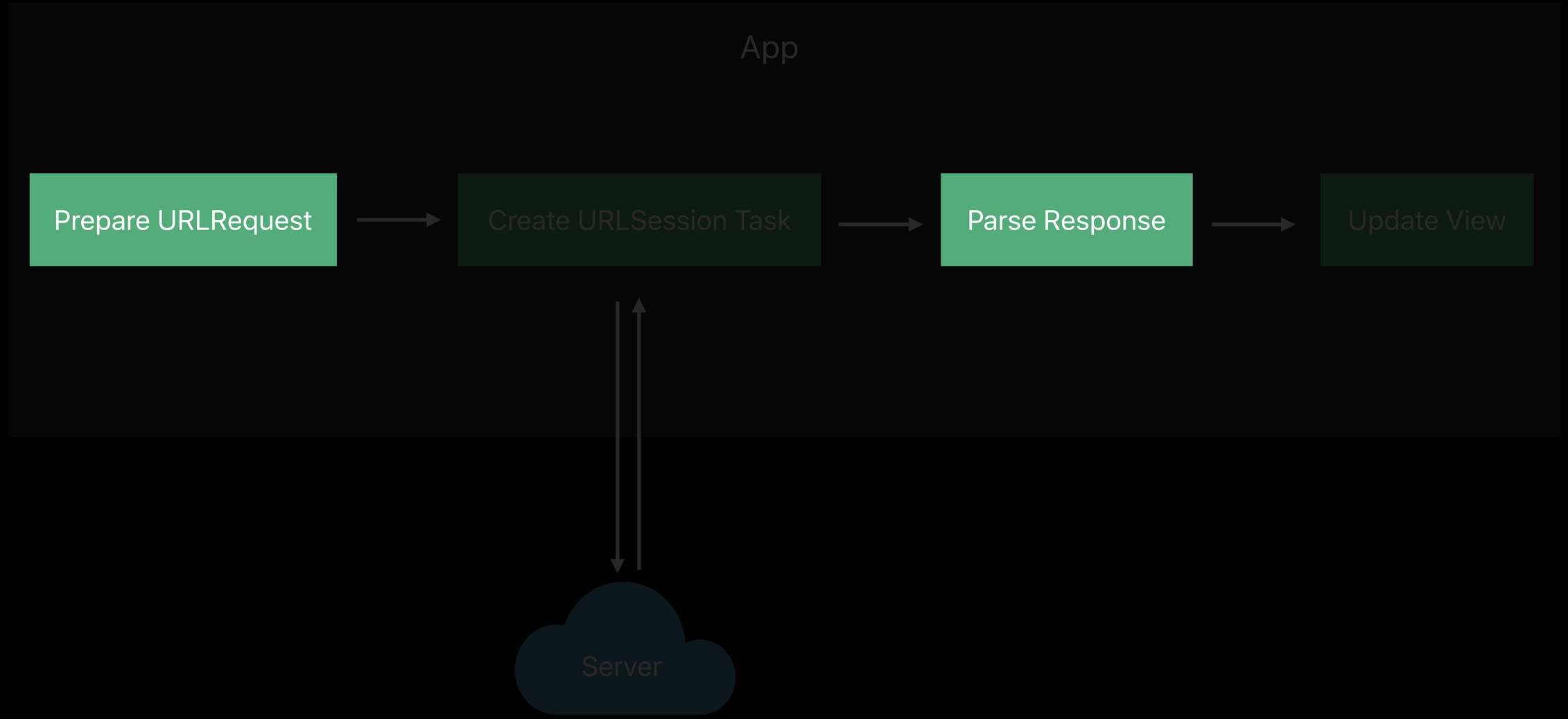
Unit Tests


App
Prepare URLRequest
Create URLSession Task
Parse Response
Update View
Server

Unit Tests


App
Prepare URLRequest
Create URLSession Task
Parse Response
Update View
Server

Unit Tests


App
Prepare URLRequest
Create URLSession Task
Parse Response
Update View
Server

```
struct PointsOfInterestRequest {
    func makeRequest(from coordinate: CLLocationCoordinate2D) throws -> URLRequest {
        guard CLLocationCoordinate2DIsValid(coordinate) else {
            throw RequestError.invalidCoordinate
        }

        var components = URLComponents(string: "https://example.com/locations")!
        components.queryItems = [
            URLQueryItem(name: "lat", value: "\(coordinate.latitude)"),
            URLQueryItem(name: "long", value: "\(coordinate.longitude)")
        ]
        return URLRequest(url: components.url!)
    }
    func parseResponse(data: Data) throws -> [PointOfInterest] {
        return try JSONDecoder().decode([PointOfInterest].self, from: data)
    }
}
```

```
struct PointsOfInterestRequest {
    func makeRequest(from coordinate: CLLocationCoordinate2D) throws -> URLRequest {
        guard CLLocationCoordinate2DIsValid(coordinate) else {
            throw RequestError.invalidCoordinate
        }

        var components = URLComponents(string: "https://example.com/locations")!
        components.queryItems = [
            URLQueryItem(name: "lat", value: "\(coordinate.latitude)"),
            URLQueryItem(name: "long", value: "\(coordinate.longitude)")
        ]
        return URLRequest(url: components.url!)
    }
    func parseResponse(data: Data) throws -> [PointOfInterest] {
        return try JSONDecoder().decode([PointOfInterest].self, from: data)
    }
}
```

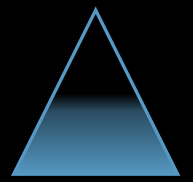
```swift
class PointOfInterestRequestTests: XCTestCase {
    let request = PointsOfInterestRequest()

    func testMakingURLRequest() throws {
        let coordinate = CLLocationCoordinate2D(latitude: 37.3293, longitude: -121.8893)

        let urlRequest = try request.makeRequest(from: coordinate)

        XCTAssertEqual(urlRequest.url?.scheme, "https")
        XCTAssertEqual(urlRequest.url?.host, "example.com")
        XCTAssertEqual(urlRequest.url?.query, "lat=37.3293&long=-121.8893")
    }
}
```

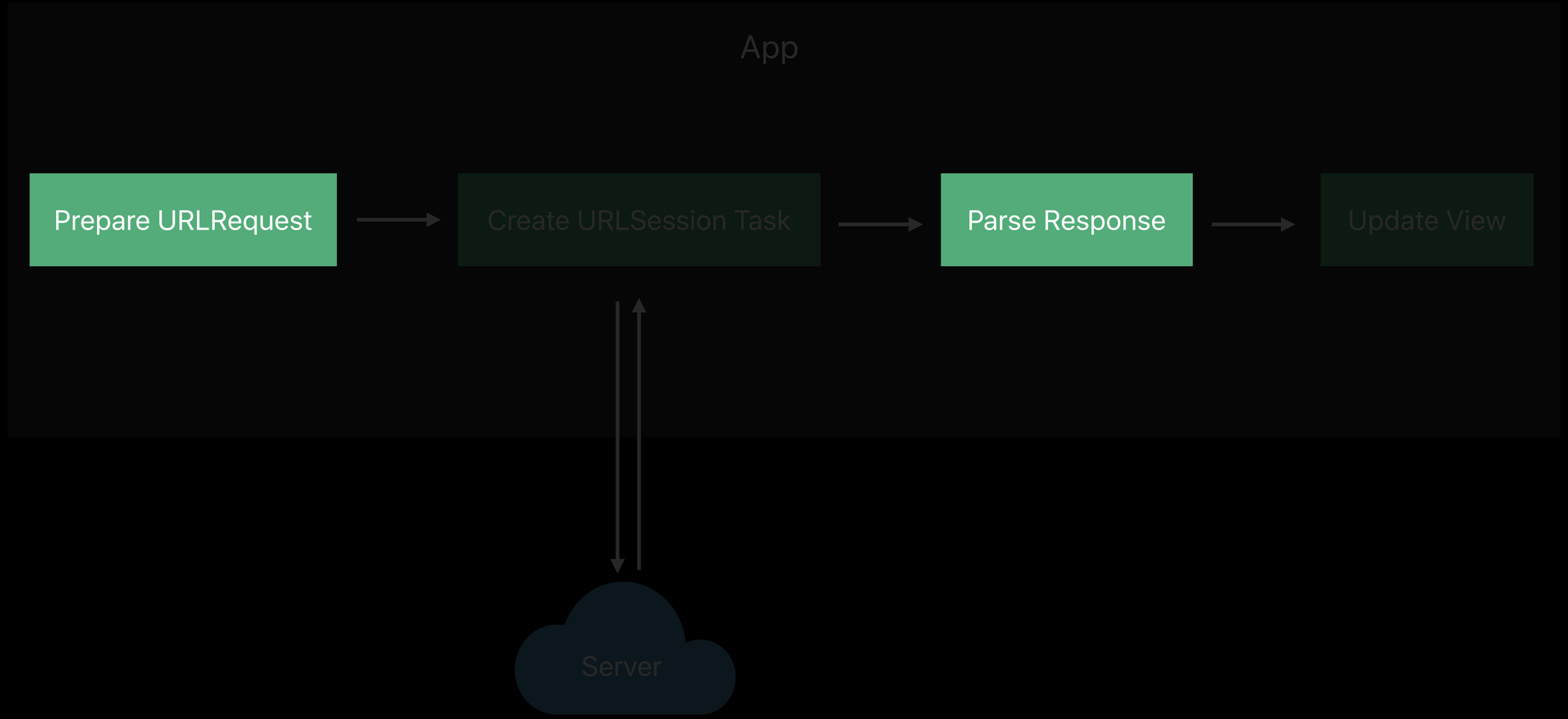
```
class PointOfInterestRequestTests: XCTestCase {
    let request = PointsOfInterestRequest()

    func testMakingURLRequest() throws {
        let coordinate = CLLocationCoordinate2D(latitude: 37.3293, longitude: -121.8893)

        let urlRequest = try request.makeRequest(from: coordinate)

        XCTAssertEqual(urlRequest.url?.scheme, "https")
        XCTAssertEqual(urlRequest.url?.host, "example.com")
        XCTAssertEqual(urlRequest.url?.query, "lat=37.3293&long=-121.8893")
    }
}
```

```
class PointOfInterestRequestTests: XCTestCase {
    let request = PointsOfInterestRequest()

    func testParsingResponse() throws {
        let jsonData = "[{\"name\":\"My Location\"}]".data(using: .utf8)!
        let response = try request.parseResponse(data: jsonData)
        XCTAssertEqual(response, [PointOfInterest(name: "My Location")])
    }
}
```

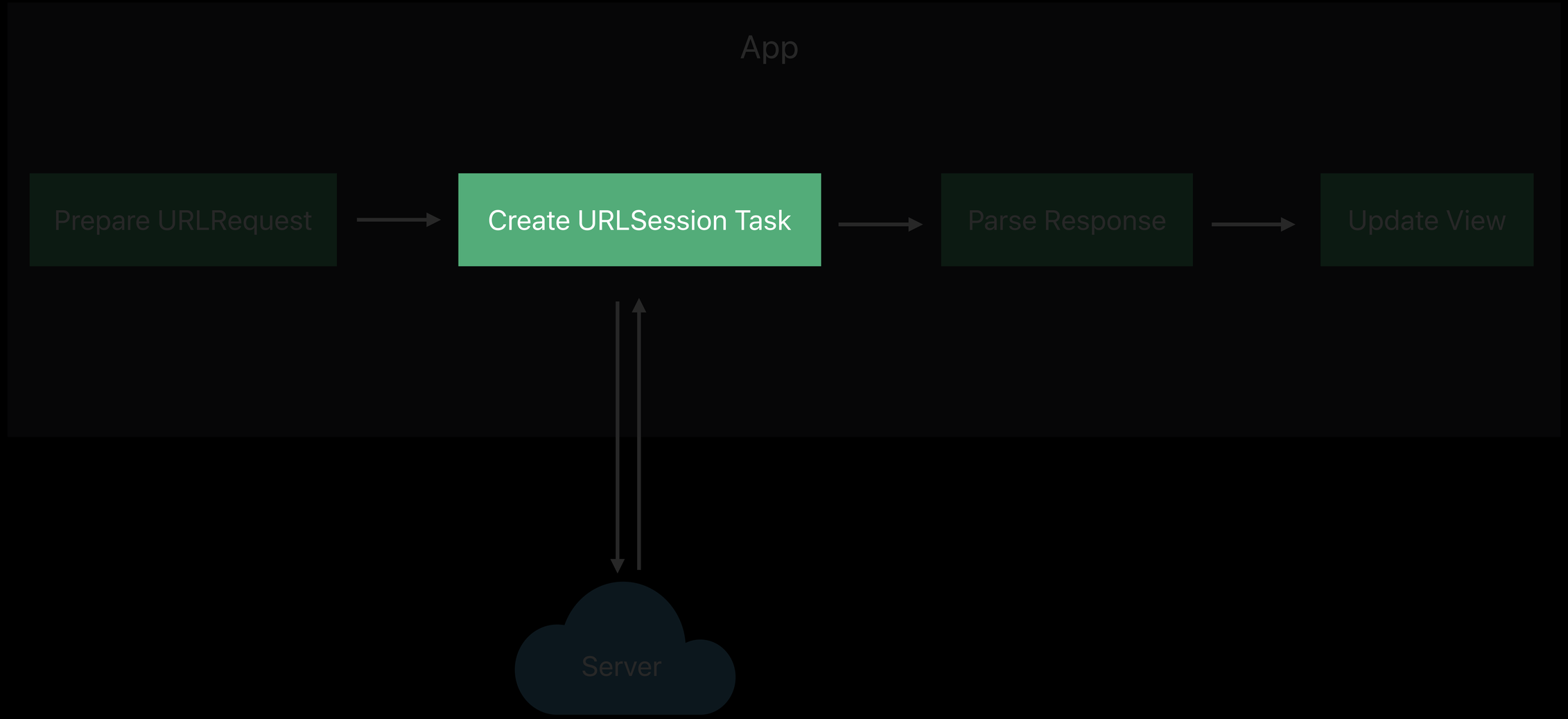
```
class PointOfInterestRequestTests: XCTestCase {
    let request = PointsOfInterestRequest()

    func testParsingResponse() throws {
        let jsonData = "[{\"name\":\"My Location\"}]".data(using: .utf8)!
        let response = try request.parseResponse(data: jsonData)
        XCTAssertEqual(response, [PointOfInterest(name: "My Location")])
    }
}
```

Unit Tests




App
Prepare URLRequest
Create URLSession Task
Parse Response
Update View
Server

Unit Tests


App
Prepare URLRequest
Create URLSession Task
Parse Response
Update View
Server

```swift
protocol APIRequest {
    associatedtype RequestDataType
    associatedtype ResponseDataType

    func makeRequest(from data: RequestDataType) throws -> URLRequest
    func parseResponse(data: Data) throws -> ResponseDataType
}

class APIRequestLoader<T: APIRequest> {
    let apiRequest: T
    let urlSession: URLSession

    init(apiRequest: T, urlSession: URLSession = .shared) {
        self.apiRequest = apiRequest
        self.urlSession = urlSession
    }
}
```

```
protocol APIRequest {
    associatedtype RequestDataType
    associatedtype ResponseDataType

    func makeRequest(from data: RequestDataType) throws -> URLRequest
    func parseResponse(data: Data) throws -> ResponseDataType
}

class APIRequestLoader<T: APIRequest> {
    let apiRequest: T
    let urlSession: URLSession

    init(apiRequest: T, urlSession: URLSession = .shared) {
        self.apiRequest = apiRequest
        self.urlSession = urlSession
    }
}
```

```swift
protocol APIRequest {
    associatedtype RequestDataType
    associatedtype ResponseDataType

    func makeRequest(from data: RequestDataType) throws -> URLRequest
    func parseResponse(data: Data) throws -> ResponseDataType
}

class APIRequestLoader<T: APIRequest> {
    let apiRequest: T
    let urlSession: URLSession

    init(apiRequest: T, urlSession: URLSession = .shared) {
        self.apiRequest = apiRequest
        self.urlSession = urlSession
    }
}
```

```swift
protocol APIRequest {
    associatedtype RequestDataType
    associatedtype ResponseDataType

    func makeRequest(from data: RequestDataType) throws -> URLRequest
    func parseResponse(data: Data) throws -> ResponseDataType
}

class APIRequestLoader<T: APIRequest> {
    let apiRequest: T
    let urlSession: URLSession

    init(apiRequest: T, urlSession: URLSession = .shared) {
        self.apiRequest = apiRequest
        self.urlSession = urlSession
    }
}
```

```swift
protocol APIRequest {
    associatedtype RequestDataType
    associatedtype ResponseDataType

    func makeRequest(from data: RequestDataType) throws -> URLRequest
    func parseResponse(data: Data) throws -> ResponseDataType
}

class APIRequestLoader<T: APIRequest> {
    let apiRequest: T
    let urlSession: URLSession

    init(apiRequest: T, urlSession: URLSession = .shared) {
        self.apiRequest = apiRequest
        self.urlSession = urlSession
    }
}
```

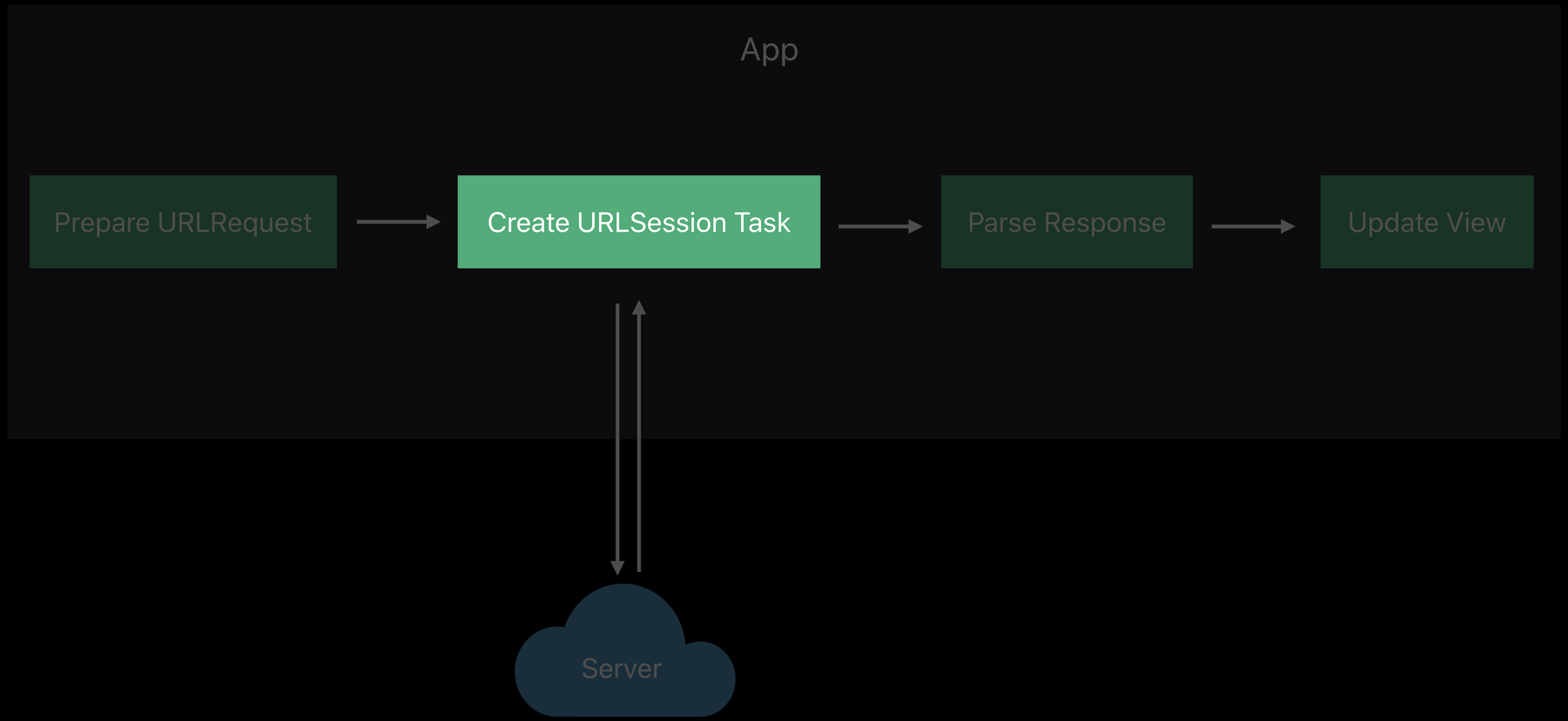
```
class APIRequestLoader<T: APIRequest> {
    func loadAPIRequest(requestData: T.RequestDataType,
                        completionHandler: @escaping (T.ResponseDataType?, Error?) -> Void) {
        do {
            let urlRequest = try apiRequest.makeRequest(from: requestData)
            urlSession.dataTask(with: urlRequest) { data, response, error in
                guard let data = data else { return completionHandler(nil, error) }
                do {
                    let parsedResponse = try self.apiRequest.parseResponse(data: data)
                    completionHandler(parsedResponse, nil)
                } catch {
                    completionHandler(nil, error)
                }
            }.resume()
        } catch { return completionHandler(nil, error) }
    }
}
```

```
class APIRequestLoader<T: APIRequest> {
    func loadAPIRequest(requestData: T.RequestDataType,
                        completionHandler: @escaping (T.ResponseDataType?, Error?) -> Void) {
        do {
            let urlRequest = try apiRequest.makeRequest(from: requestData)
            urlSession.dataTask(with: urlRequest) { data, response, error in
                guard let data = data else { return completionHandler(nil, error) }
                do {
                    let parsedResponse = try self.apiRequest.parseResponse(data: data)
                    completionHandler(parsedResponse, nil)
                } catch {
                    completionHandler(nil, error)
                }
            }.resume()
        } catch { return completionHandler(nil, error) }
    }
}
```

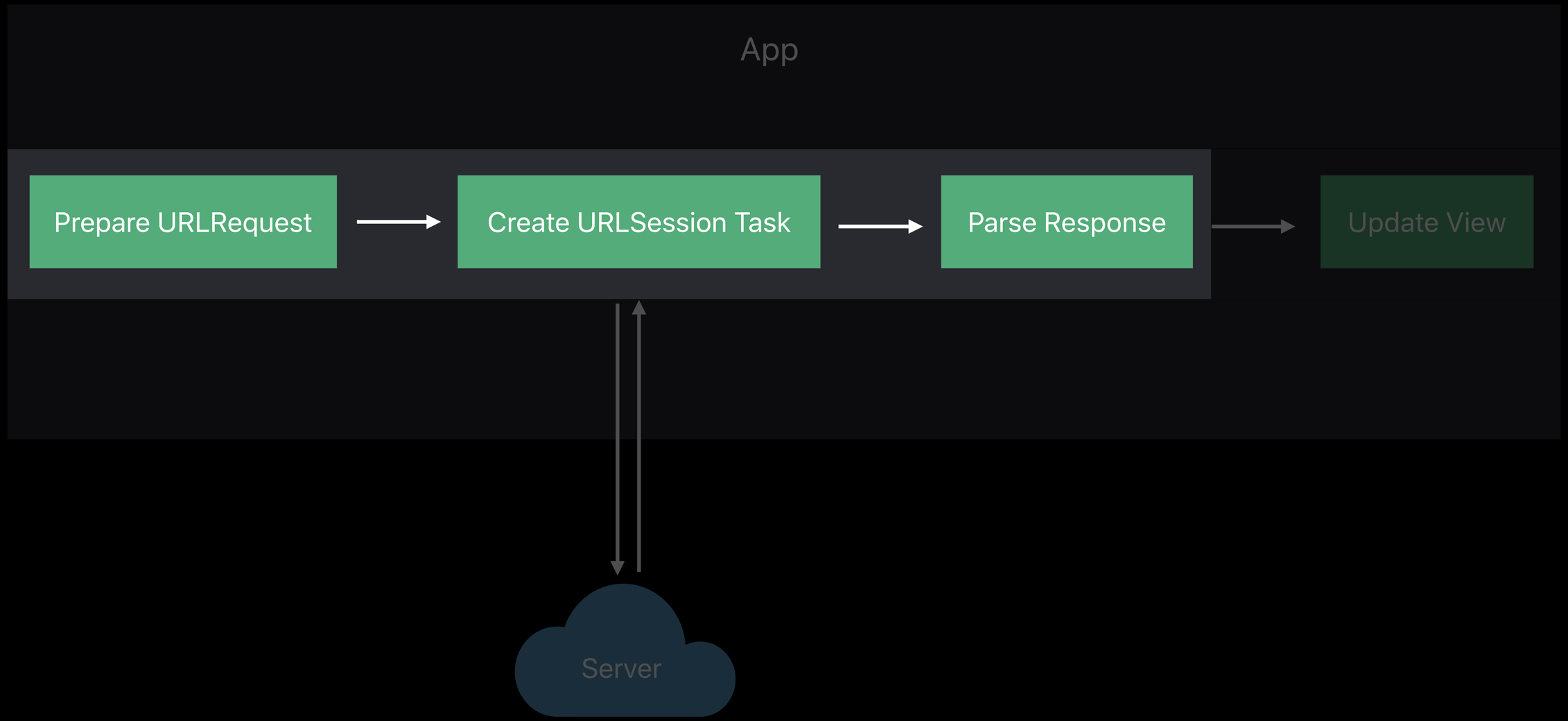
```
class APIRequestLoader<T: APIRequest> {
    func loadAPIRequest(requestData: T.RequestDataType,
                        completionHandler: @escaping (T.ResponseDataType?, Error?) -> Void) {
        do {
            let urlRequest = try apiRequest.makeRequest(from: requestData)
            urlSession.dataTask(with: urlRequest) { data, response, error in
                guard let data = data else { return completionHandler(nil, error) }
                do {
                    let parsedResponse = try self.apiRequest.parseResponse(data: data)
                    completionHandler(parsedResponse, nil)
                } catch {
                    completionHandler(nil, error)
                }
            }.resume()
        } catch { return completionHandler(nil, error) }
    }
}
```

```
class APIRequestLoader<T: APIRequest> {
    func loadAPIRequest(requestData: T.RequestDataType,
                        completionHandler: @escaping (T.ResponseDataType?, Error?) -> Void) {
        do {
            let urlRequest = try apiRequest.makeRequest(from: requestData)
            urlSession.dataTask(with: urlRequest) { data, response, error in
                guard let data = data else { return completionHandler(nil, error) }
                do {
                    let parsedResponse = try self.apiRequest.parseResponse(data: data)
                    completionHandler(parsedResponse, nil)
                } catch {
                    completionHandler(nil, error)
                }
            }.resume()
        } catch { return completionHandler(nil, error) }
    }
}
```

Unit Tests

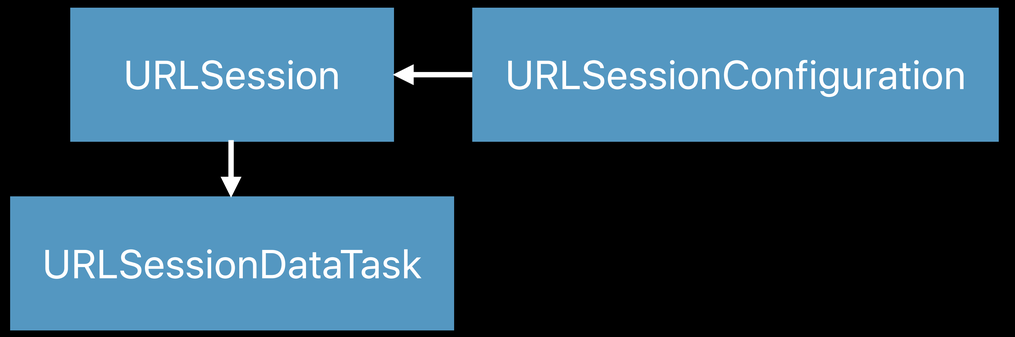

App
Prepare URLRequest
Create URLSession Task
Parse Response
Update View
Server

Integration Tests

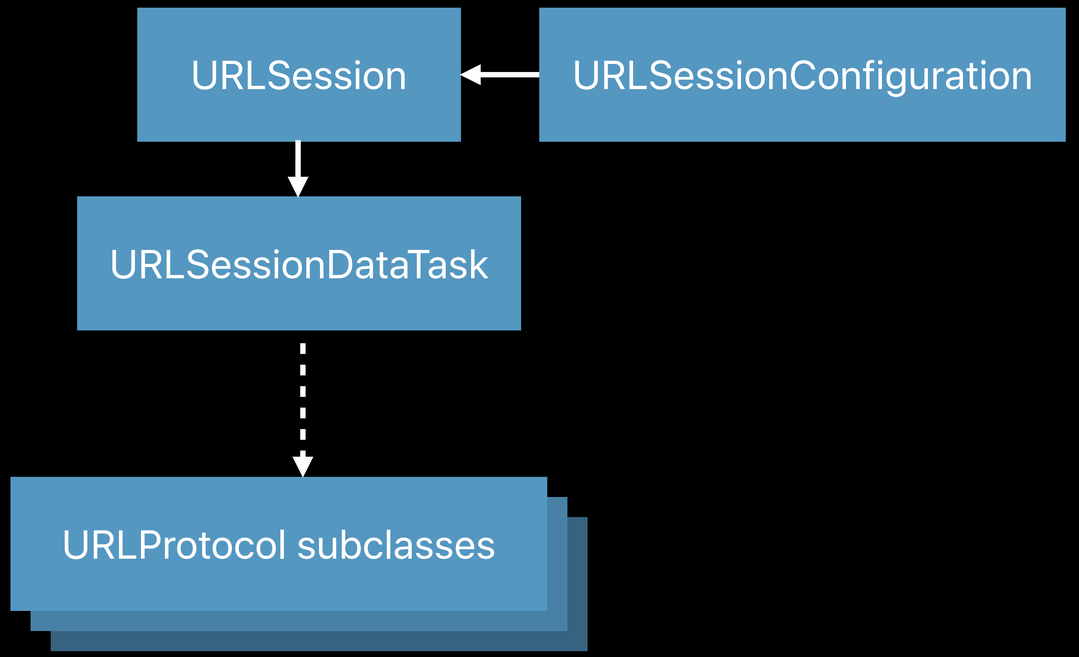


How to Use URLProtocol

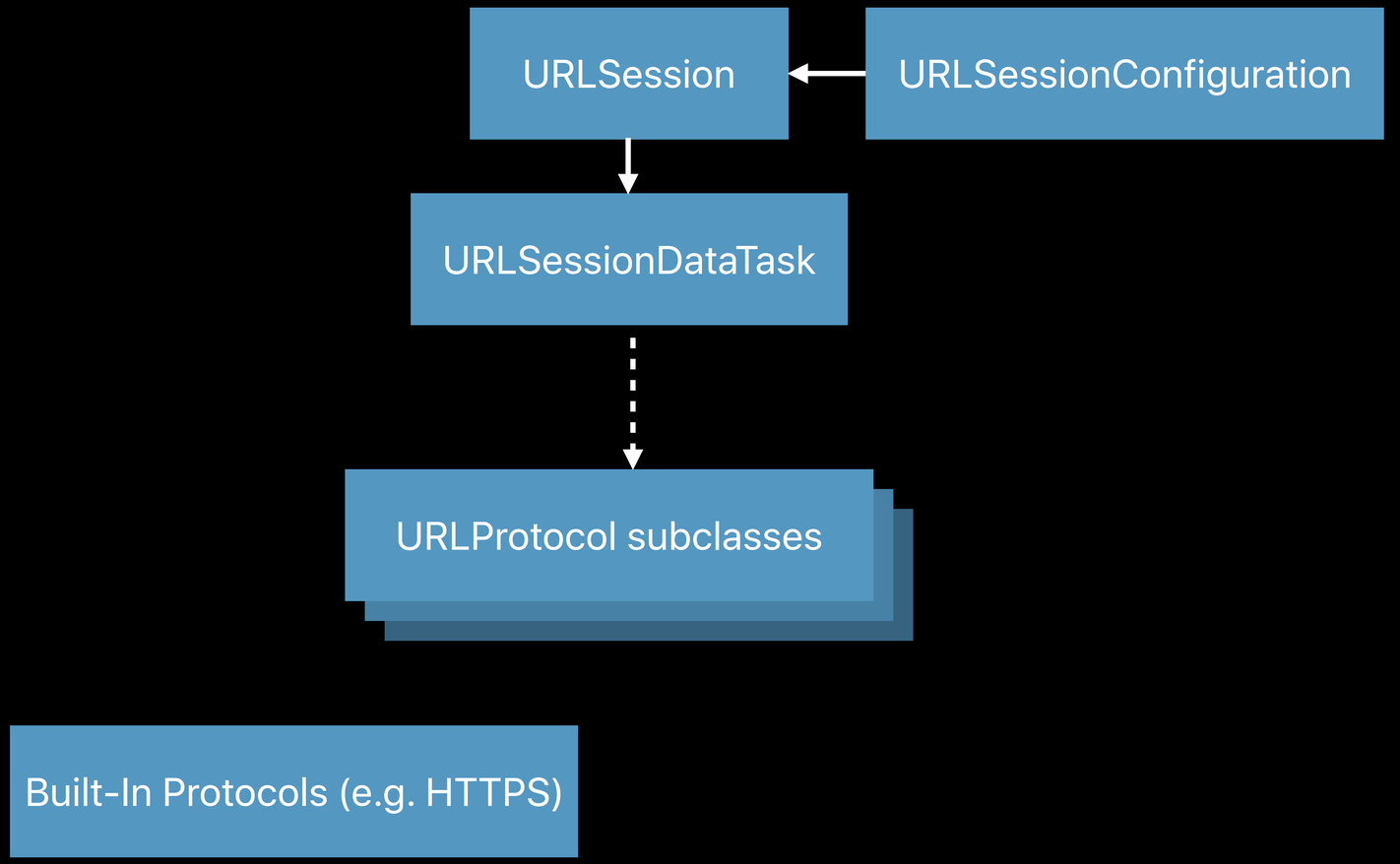
How to Use URLProtocol



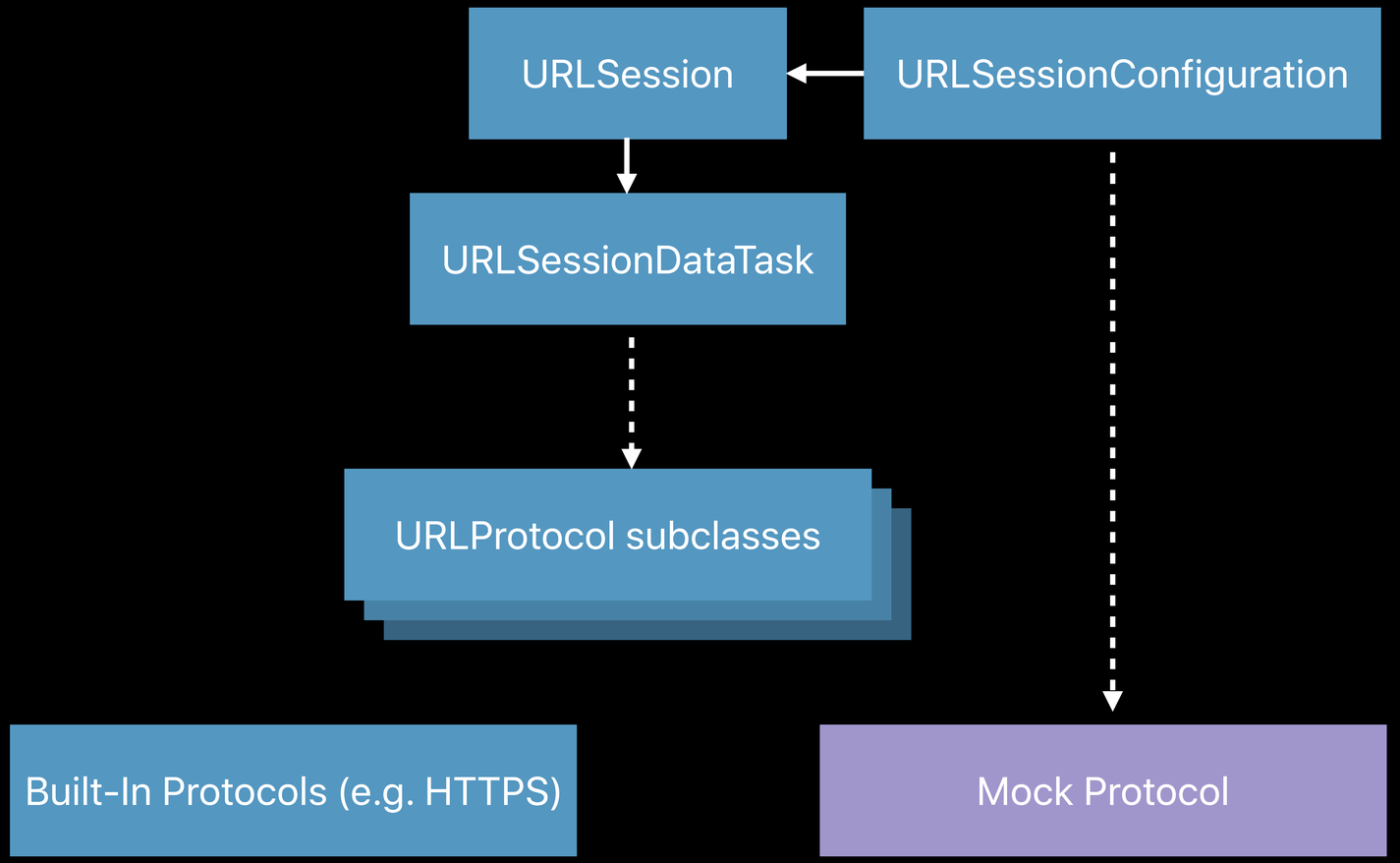
How to Use URLProtocol



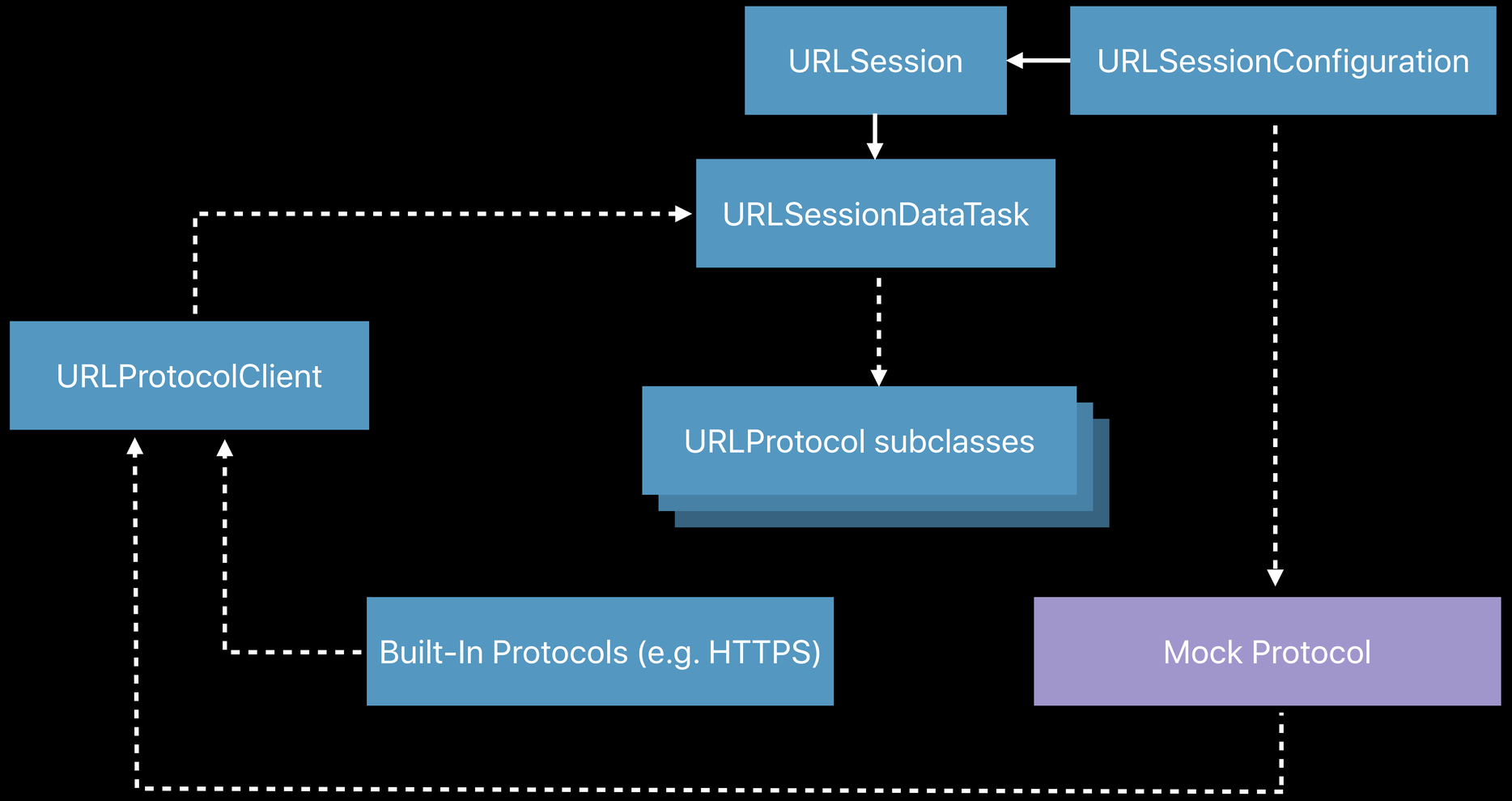
How to Use URLProtocol



How to Use URLProtocol



How to Use URLProtocol



```
class MockURLProtocol: URLProtocol {
    override class func canInit(with request: URLRequest) -> Bool {
        return true
    }

    override class func canonicalRequest(for request: URLRequest) -> URLRequest {
        return request
    }

    override func startLoading() {
        // ...
    }

    override func stopLoading() {
        // ...
    }
}
```

```
class MockURLProtocol: URLProtocol {
    override class func canInit(with request: URLRequest) -> Bool {
        return true
    }

    override class func canonicalRequest(for request: URLRequest) -> URLRequest {
        return request
    }

    override func startLoading() {
        // ...
    }

    override func stopLoading() {
        // ...
    }
}
```

```
class MockURLProtocol: URLProtocol {
    override class func canInit(with request: URLRequest) -> Bool {
        return true
    }

    override class func canonicalRequest(for request: URLRequest) -> URLRequest {
        return request
    }

    override func startLoading() {
        // ...
    }

    override func stopLoading() {
        // ...
    }
}
```

```
class MockURLProtocol: URLProtocol {
    override class func canInit(with request: URLRequest) -> Bool {
        return true
    }

    override class func canonicalRequest(for request: URLRequest) -> URLRequest {
        return request
    }

    override func startLoading() {
        // ...
    }

    override func stopLoading() {
        // ...
    }
}
```

```
class MockURLProtocol: URLProtocol {
    static var requestHandler: ((URLRequest) throws -> (HTTPURLResponse, Data))?
    override func startLoading() {
        guard let handler = MockURLProtocol.requestHandler else {
            XCTFail("Received unexpected request with no handler set")
            return
        }
        do {
            let (response, data) = try handler(request)
            client?.urlProtocol(self, didReceive: response, cacheStoragePolicy: .notAllowed)
            client?.urlProtocol(self, didLoad: data)
            client?.urlProtocolDidFinishLoading(self)
        } catch {
            client?.urlProtocol(self, didFailWithError: error)
        }
    }
}
```

```
class MockURLProtocol: URLProtocol {
    static var requestHandler: ((URLRequest) throws -> (HTTPURLResponse, Data))?
    override func startLoading() {
        guard let handler = MockURLProtocol.requestHandler else {
            XCTFail("Received unexpected request with no handler set")
            return
        }
        do {
            let (response, data) = try handler(request)
            client?.urlProtocol(self, didReceive: response, cacheStoragePolicy: .notAllowed)
            client?.urlProtocol(self, didLoad: data)
            client?.urlProtocolDidFinishLoading(self)
        } catch {
            client?.urlProtocol(self, didFailWithError: error)
        }
    }
}
```

```
class MockURLProtocol: URLProtocol {
    static var requestHandler: ((URLRequest) throws -> (HTTPURLResponse, Data))?
    override func startLoading() {
        guard let handler = MockURLProtocol.requestHandler else {
            XCTFail("Received unexpected request with no handler set")
            return
        }
        do {
            let (response, data) = try handler(request)
            client?.urlProtocol(self, didReceive: response, cacheStoragePolicy: .notAllowed)
            client?.urlProtocol(self, didLoad: data)
            client?.urlProtocolDidFinishLoading(self)
        } catch {
            client?.urlProtocol(self, didFailWithError: error)
        }
    }
}
```

```
class MockURLProtocol: URLProtocol {
    static var requestHandler: ((URLRequest) throws -> (HTTPURLResponse, Data))?
    override func startLoading() {
        guard let handler = MockURLProtocol.requestHandler else {
            XCTFail("Received unexpected request with no handler set")
            return
        }
        do {
            let (response, data) = try handler(request)
            client?.urlProtocol(self, didReceive: response, cacheStoragePolicy: .notAllowed)
            client?.urlProtocol(self, didLoad: data)
            client?.urlProtocolDidFinishLoading(self)
        } catch {
            client?.urlProtocol(self, didFailWithError: error)
        }
    }
}
```

```
class MockURLProtocol: URLProtocol {
    static var requestHandler: ((URLRequest) throws -> (HTTPURLResponse, Data))?
    override func startLoading() {
        guard let handler = MockURLProtocol.requestHandler else {
            XCTFail("Received unexpected request with no handler set")
            return
        }
        do {
            let (response, data) = try handler(request)
            client?.urlProtocol(self, didReceive: response, cacheStoragePolicy: .notAllowed)
            client?.urlProtocol(self, didLoad: data)
            client?.urlProtocolDidFinishLoading(self)
        } catch {
            client?.urlProtocol(self, didFailWithError: error)
        }
    }
}
```

```
class MockURLProtocol: URLProtocol {
    static var requestHandler: ((URLRequest) throws -> (HTTPURLResponse, Data))?
    override func startLoading() {
        guard let handler = MockURLProtocol.requestHandler else {
            XCTFail("Received unexpected request with no handler set")
            return
        }
        do {
            let (response, data) = try handler(request)
            client?.urlProtocol(self, didReceive: response, cacheStoragePolicy: .notAllowed)
            client?.urlProtocol(self, didLoad: data)
            client?.urlProtocolDidFinishLoading(self)
        } catch {
            client?.urlProtocol(self, didFailWithError: error)
        }
    }
}
```

```
class MockURLProtocol: URLProtocol {
    static var requestHandler: ((URLRequest) throws -> (HTTPURLResponse, Data))?
    override func startLoading() {
        guard let handler = MockURLProtocol.requestHandler else {
            XCTFail("Received unexpected request with no handler set")
            return
        }
        do {
            let (response, data) = try handler(request)
            client?.urlProtocol(self, didReceive: response, cacheStoragePolicy: .notAllowed)
            client?.urlProtocol(self, didLoad: data)
            client?.urlProtocolDidFinishLoading(self)
        } catch {
            client?.urlProtocol(self, didFailWithError: error)
        }
    }
}
```

```
class MockURLProtocol: URLProtocol {
    static var requestHandler: ((URLRequest) throws -> (HTTPURLResponse, Data))?
    override func startLoading() {
        guard let handler = MockURLProtocol.requestHandler else {
            XCTFail("Received unexpected request with no handler set")
            return
        }
        do {
            let (response, data) = try handler(request)
            client?.urlProtocol(self, didReceive: response, cacheStoragePolicy: .notAllowed)
            client?.urlProtocol(self, didLoad: data)
            client?.urlProtocolDidFinishLoading(self)
        } catch {
            client?.urlProtocol(self, didFailWithError: error)
        }
    }
}
```

```
class APILoaderTests: XCTestCase {
    var loader: APIRequestLoader<PointsOfInterestRequest>!

    override func setUp() {
        let request = PointsOfInterestRequest()

        let configuration = URLSessionConfiguration.ephemeral
        configuration.protocolClasses = [MockURLProtocol.self]
        let urlSession = URLSession(configuration: configuration)

        loader = APIRequestLoader(apiRequest: request, urlSession: urlSession)
    }
}
```

```
class APILoaderTests: XCTestCase {
    var loader: APIRequestLoader<PointsOfInterestRequest>!

    override func setUp() {
        let request = PointsOfInterestRequest()

        let configuration = URLSessionConfiguration.ephemeral
        configuration.protocolClasses = [MockURLProtocol.self]
        let urlSession = URLSession(configuration: configuration)

        loader = APIRequestLoader(apiRequest: request, urlSession: urlSession)
    }
}
```

```
class APILoaderTests: XCTestCase {
    var loader: APIRequestLoader<PointsOfInterestRequest>!

    override func setUp() {
        let request = PointsOfInterestRequest()

        let configuration = URLSessionConfiguration.ephemeral
        configuration.protocolClasses = [MockURLProtocol.self]
        let urlSession = URLSession(configuration: configuration)

        loader = APIRequestLoader(apiRequest: request, urlSession: urlSession)
    }
}
```

```
class APILoaderTests: XCTestCase {
    func testLoaderSuccess() {
        let inputCoordinate = CLLocationCoordinate2D(latitude: 37.3293, longitude: -121.8893)
        let mockJSONData = "[{\"name\":\"MyPointOfInterest\"}]".data(using: .utf8)!
        MockURLProtocol.requestHandler = { request in
            XCTAssertEqual(request.url?.query?.contains("lat=37.3293"), true)
            return (HTTPURLResponse(), mockJSONData)
        }

        let expectation = XCTestExpectation(description: "response")
        loader.loadAPIRequest(requestData: inputCoordinate) { pointsOfInterest, error in
            XCTAssertEqual(pointsOfInterest, [PointOfInterest(name: "MyPointOfInterest")])
            expectation.fulfill()
        }
        wait(for: [expectation], timeout: 1)
    }
}
```

```
class APILoaderTests: XCTestCase {
    func testLoaderSuccess() {
        let inputCoordinate = CLLocationCoordinate2D(latitude: 37.3293, longitude: -121.8893)
        let mockJSONData = "[{\"name\":\"MyPointOfInterest\"}]".data(using: .utf8)!
        MockURLProtocol.requestHandler = { request in
            XCTAssertEqual(request.url?.query?.contains("lat=37.3293"), true)
            return (HTTPURLResponse(), mockJSONData)
        }

        let expectation = XCTestExpectation(description: "response")
        loader.loadAPIRequest(requestData: inputCoordinate) { pointsOfInterest, error in
            XCTAssertEqual(pointsOfInterest, [PointOfInterest(name: "MyPointOfInterest")])
            expectation.fulfill()
        }
        wait(for: [expectation], timeout: 1)
    }
}
```

```
class APILoaderTests: XCTestCase {
    func testLoaderSuccess() {
        let inputCoordinate = CLLocationCoordinate2D(latitude: 37.3293, longitude: -121.8893)
        let mockJSONData = "[{\"name\":\"MyPointOfInterest\"}]".data(using: .utf8)!
        MockURLProtocol.requestHandler = { request in
            XCTAssertEqual(request.url?.query?.contains("lat=37.3293"), true)
            return (HTTPURLResponse(), mockJSONData)
        }

        let expectation = XCTestExpectation(description: "response")
        loader.loadAPIRequest(requestData: inputCoordinate) { pointsOfInterest, error in
            XCTAssertEqual(pointsOfInterest, [PointOfInterest(name: "MyPointOfInterest")])
            expectation.fulfill()
        }
        wait(for: [expectation], timeout: 1)
    }
}
```

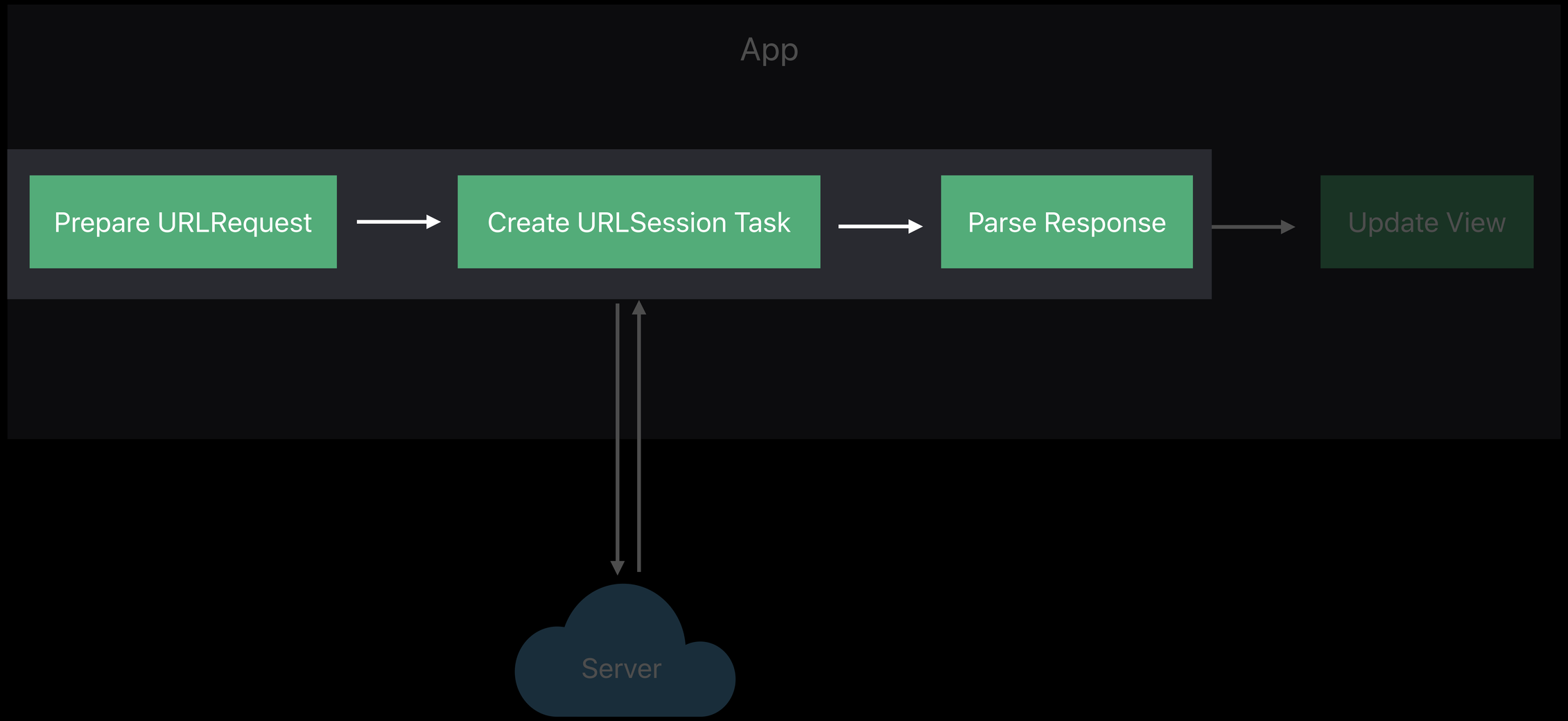
```
class APILoaderTests: XCTestCase {
    func testLoaderSuccess() {
        let inputCoordinate = CLLocationCoordinate2D(latitude: 37.3293, longitude: -121.8893)
        let mockJSONData = "[{\"name\":\"MyPointOfInterest\"}]".data(using: .utf8)!
        MockURLProtocol.requestHandler = { request in
            XCTAssertEqual(request.url?.query?.contains("lat=37.3293"), true)
            return (HTTPURLResponse(), mockJSONData)
        }

        let expectation = XCTestExpectation(description: "response")
        loader.loadAPIRequest(requestData: inputCoordinate) { pointsOfInterest, error in
            XCTAssertEqual(pointsOfInterest, [PointOfInterest(name: "MyPointOfInterest")])
            expectation.fulfill()
        }
        wait(for: [expectation], timeout: 1)
    }
}
```

```
class APILoaderTests: XCTestCase {
    func testLoaderSuccess() {
        let inputCoordinate = CLLocationCoordinate2D(latitude: 37.3293, longitude: -121.8893)
        let mockJSONData = "[{\"name\":\"MyPointOfInterest\"}]".data(using: .utf8)!
        MockURLProtocol.requestHandler = { request in
            XCTAssertEqual(request.url?.query?.contains("lat=37.3293"), true)
            return (HTTPURLResponse(), mockJSONData)
        }

        let expectation = XCTestExpectation(description: "response")
        loader.loadAPIRequest(requestData: inputCoordinate) { pointsOfInterest, error in
            XCTAssertEqual(pointsOfInterest, [PointOfInterest(name: "MyPointOfInterest")])
            expectation.fulfill()
        }
        wait(for: [expectation], timeout: 1)
    }
}
```

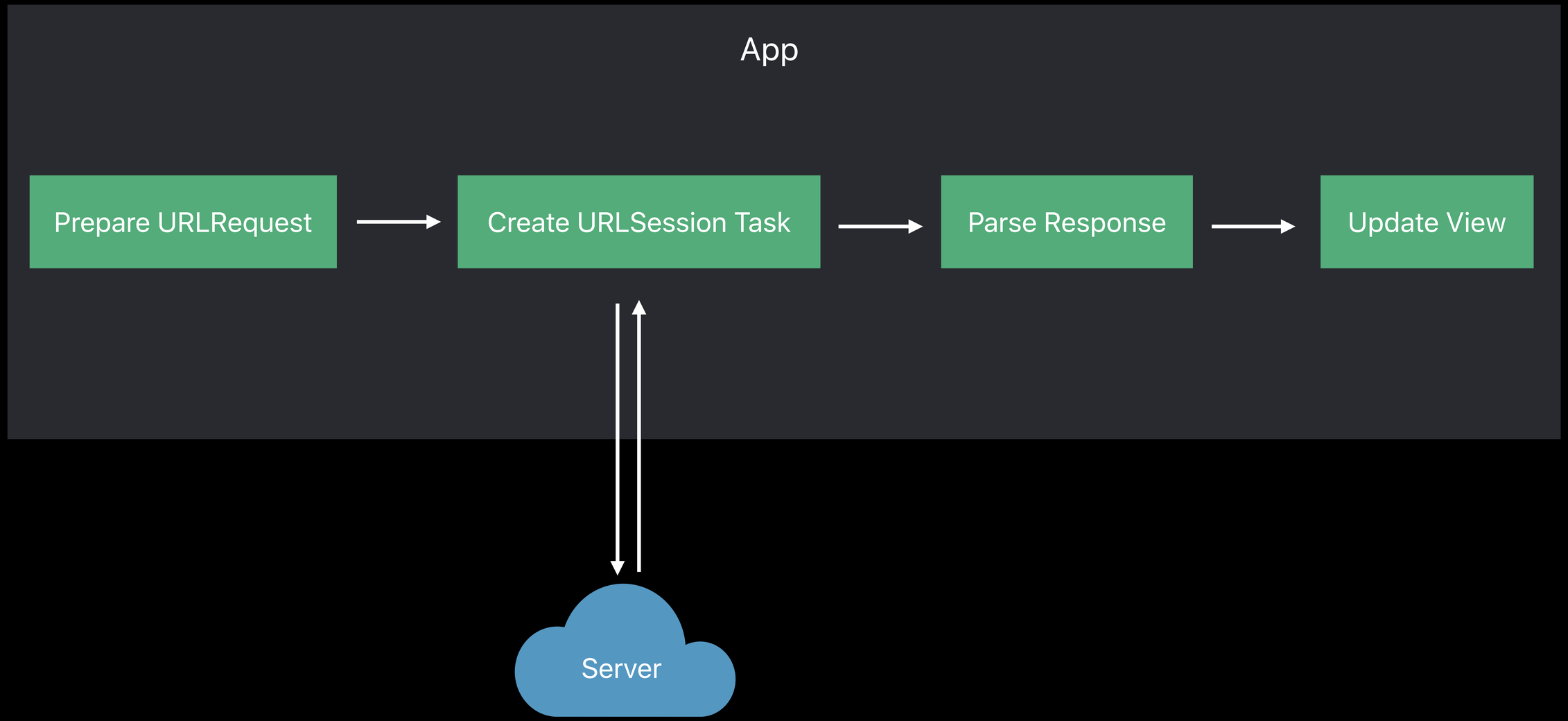
```
class APILoaderTests: XCTestCase {
    func testLoaderSuccess() {
        let inputCoordinate = CLLocationCoordinate2D(latitude: 37.3293, longitude: -121.8893)
        let mockJSONData = "[{\"name\":\"MyPointOfInterest\"}]".data(using: .utf8)!
        MockURLProtocol.requestHandler = { request in
            XCTAssertEqual(request.url?.query?.contains("lat=37.3293"), true)
            return (HTTPURLResponse(), mockJSONData)
        }

        let expectation = XCTestExpectation(description: "response")
        loader.loadAPIRequest(requestData: inputCoordinate) { pointsOfInterest, error in
            XCTAssertEqual(pointsOfInterest, [PointOfInterest(name: "MyPointOfInterest")])
            expectation.fulfill()
        }
        wait(for: [expectation], timeout: 1)
    }
}
```

```
class APILoaderTests: XCTestCase {
    func testLoaderSuccess() {
        let inputCoordinate = CLLocationCoordinate2D(latitude: 37.3293, longitude: -121.8893)
        let mockJSONData = "[{\"name\":\"MyPointOfInterest\"}]".data(using: .utf8)!
        MockURLProtocol.requestHandler = { request in
            XCTAssertEqual(request.url?.query?.contains("lat=37.3293"), true)
            return (HTTPURLResponse(), mockJSONData)
        }

        let expectation = XCTestExpectation(description: "response")
        loader.loadAPIRequest(requestData: inputCoordinate) { pointsOfInterest, error in
            XCTAssertEqual(pointsOfInterest, [PointOfInterest(name: "MyPointOfInterest")])
            expectation.fulfill()
        }
        wait(for: [expectation], timeout: 1)
    }
}
```

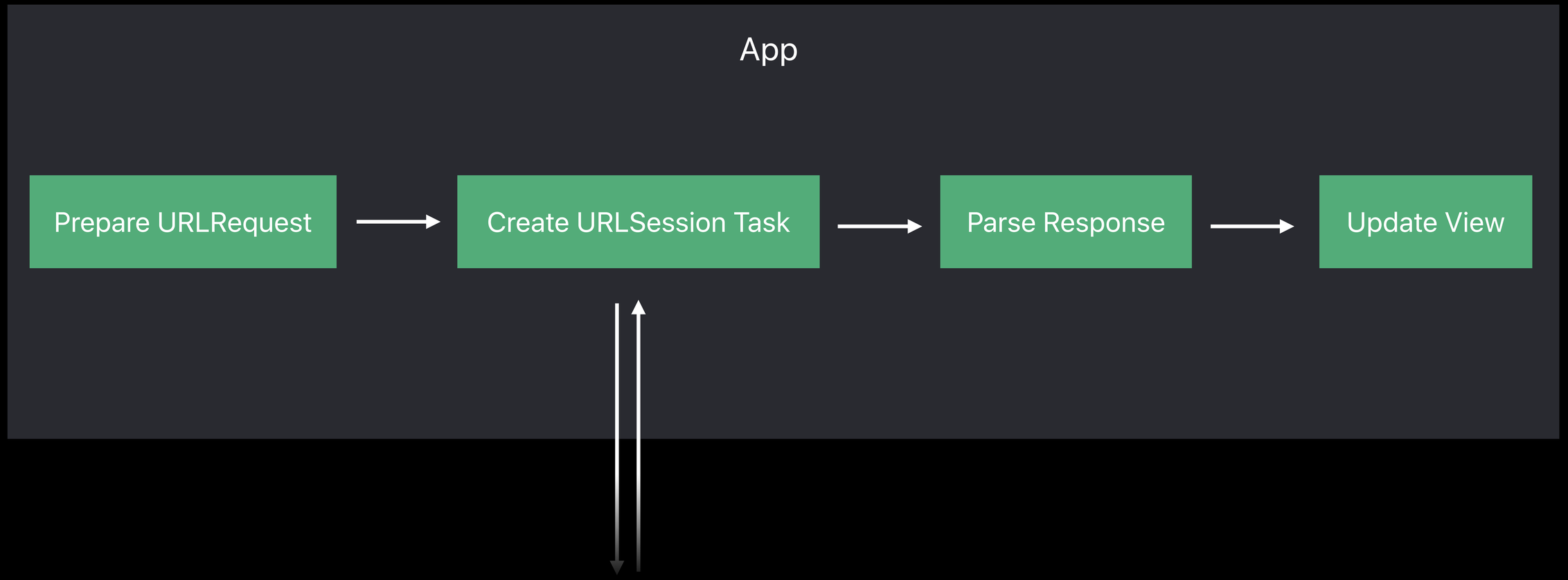
Integration Tests



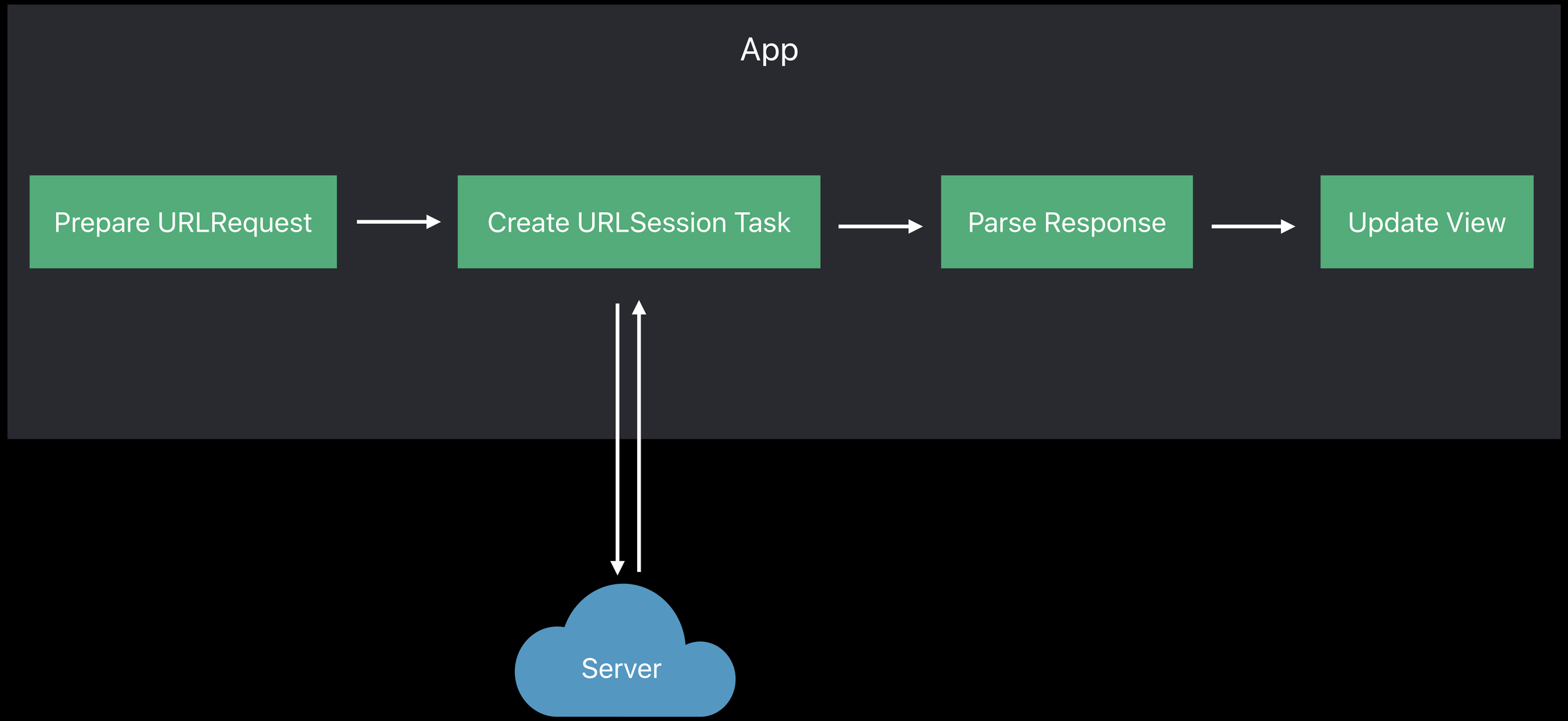
End-to-End Tests



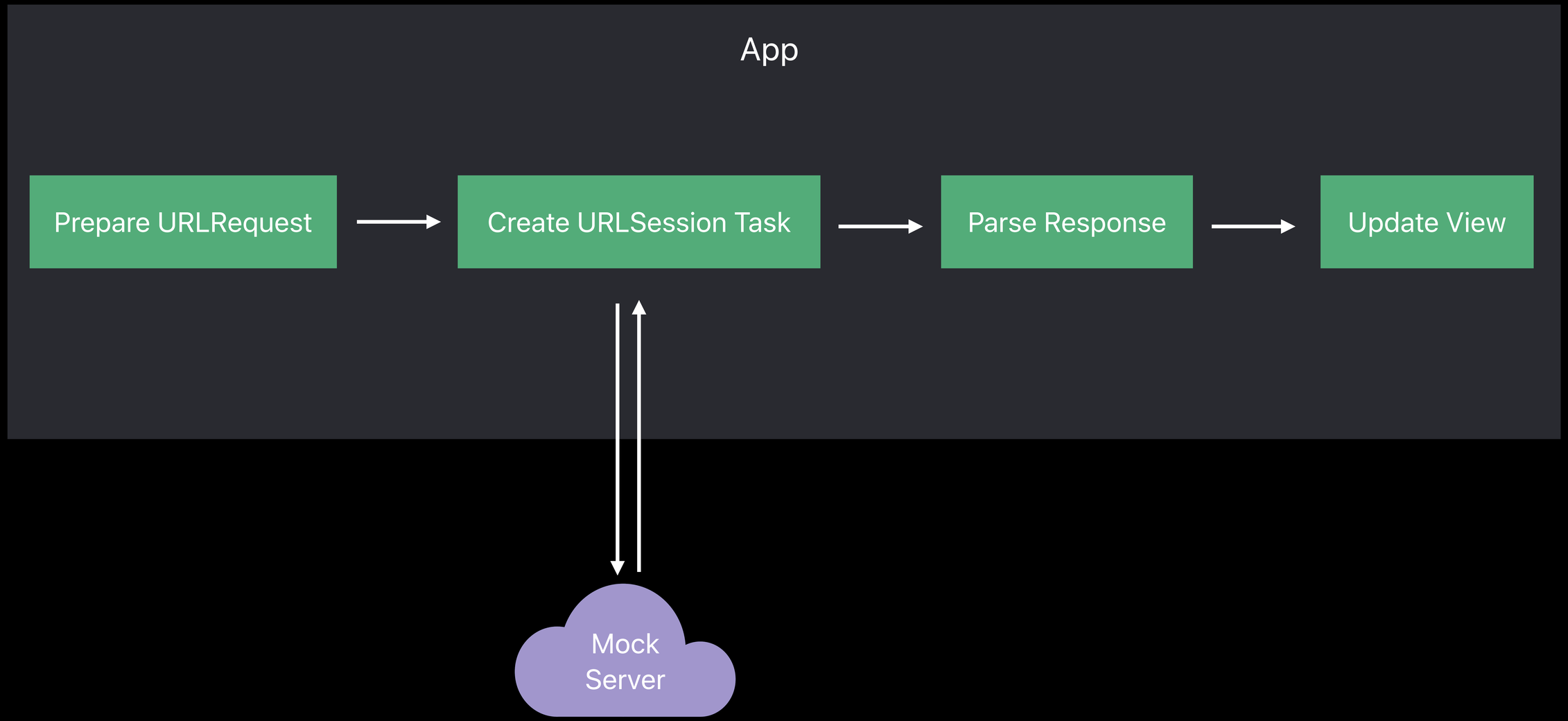
End-to-End Tests



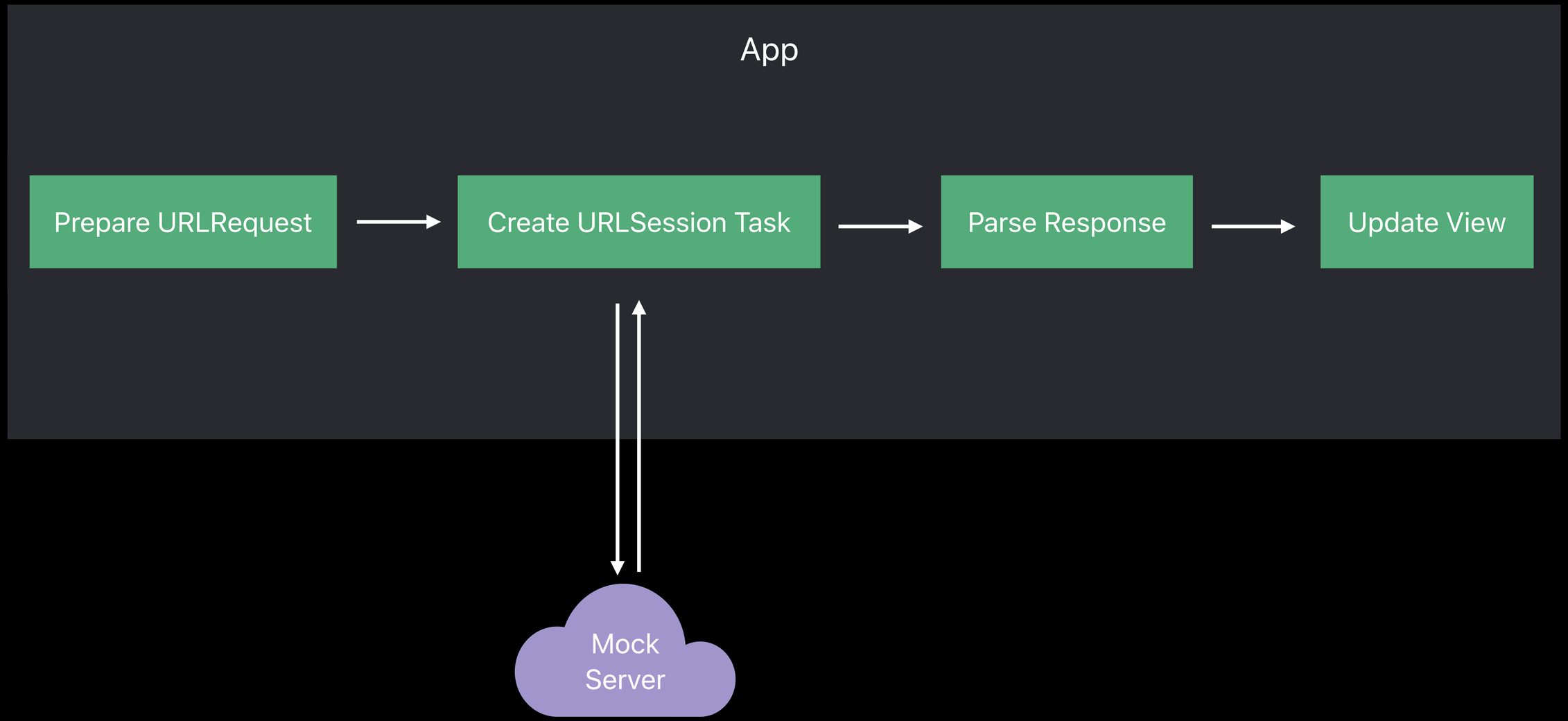
End-to-End Tests



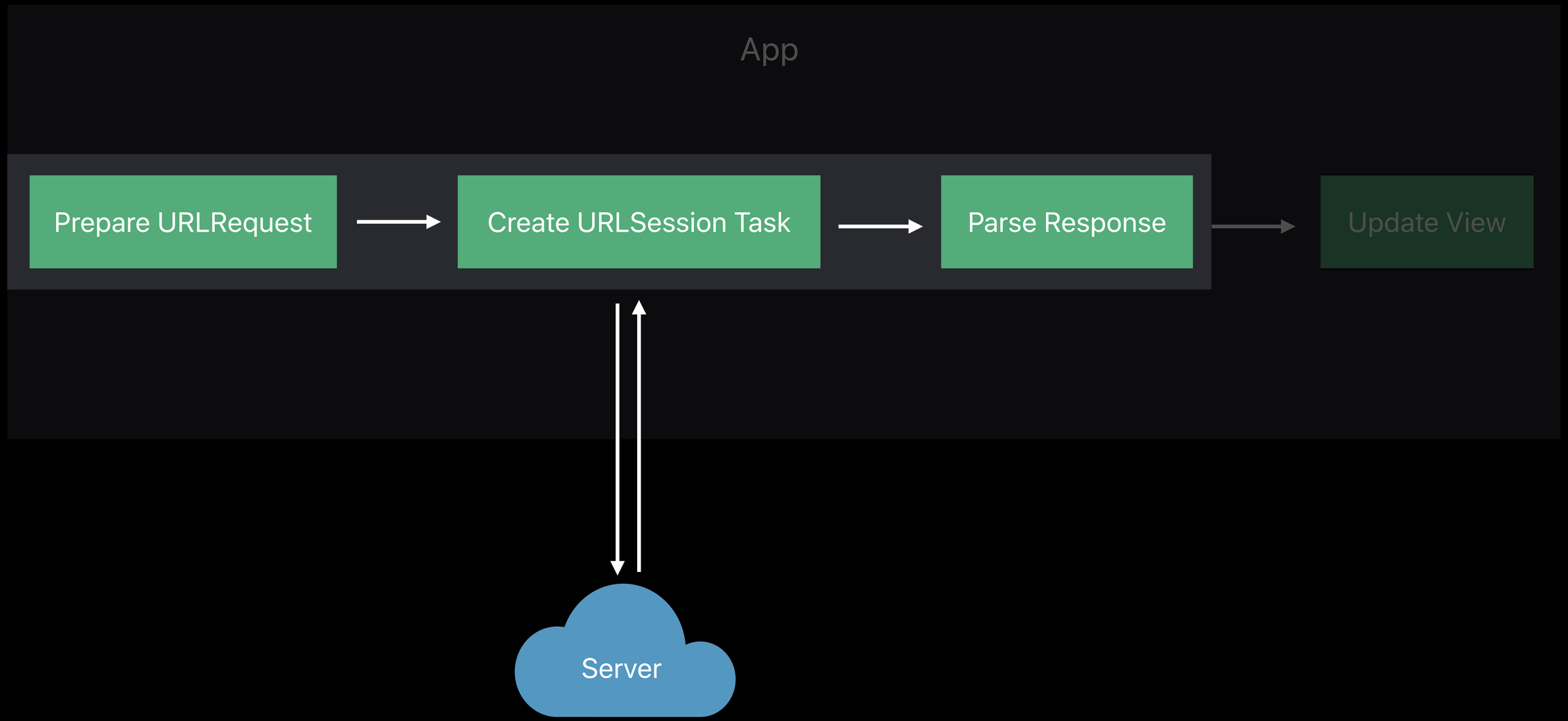
End-to-End Tests



End-to-End Tests



End-to-End Tests



Testing Network Requests

Decompose code for testability

`URLProtocol` as a mocking tool

Tiered testing strategy

Testing network requests

Working with notifications

Mocking with protocols

Test execution speed

Working with Notifications

Test that subject properly observes or posts a `Notification`

Important to test notifications in isolation

Isolation avoids unreliable or “flaky” tests

```
class PointsOfInterestTableViewController {

    var observer: AnyObject?
    init() {
        let name = CurrentLocationProvider.authChangedNotification
        observer = NotificationCenter.default.addObserver(forName: name, object: nil,
                                                          queue: .main) { [weak self] _ in
            self?.handleAuthChanged()
        }
    }

    var didHandleNotification = false
    func handleAuthChanged() {
        didHandleNotification = true
    }

}
```

```
class PointsOfInterestTableViewController {

    var observer: AnyObject?
    init() {
        let name = CurrentLocationProvider.authChangedNotification
        observer = NotificationCenter.default.addObserver(forName: name, object: nil,
                                                          queue: .main) { [weak self] _ in
            self?.handleAuthChanged()
        }
    }

    var didHandleNotification = false
    func handleAuthChanged() {
        didHandleNotification = true
    }

}
```

```
class PointsOfInterestTableViewController {

    var observer: AnyObject?
    init() {
        let name = CurrentLocationProvider.authChangedNotification
        observer = NotificationCenter.default.addObserver(forName: name, object: nil,
                                                          queue: .main) { [weak self] _ in
            self?.handleAuthChanged()
        }
    }

    var didHandleNotification = false
    func handleAuthChanged() {
        didHandleNotification = true
    }

}
```

```
class PointsOfInterestTableViewController {

    var observer: AnyObject?
    init() {
        let name = CurrentLocationProvider.authChangedNotification
        observer = NotificationCenter.default.addObserver(forName: name, object: nil,
                                                          queue: .main) { [weak self] _ in
            self?.handleAuthChanged()
        }
    }

    var didHandleNotification = false
    func handleAuthChanged() {
        didHandleNotification = true
    }

}
```

```
class PointsOfInterestTableViewControllerTests: XCTestCase {

    func testNotification() {
        let observer = PointsOfInterestTableViewController()
        XCTAssertFalse(observer.didHandleNotification)

        // ✕ This notification will be received by all objects in process,
        // could have unintended consequences
        let name = CurrentLocationProvider.authChangedNotification
        NotificationCenter.default.post(name: name, object: nil)

        XCTAssertTrue(observer.didHandleNotification)
    }

}
```

```
class PointsOfInterestTableViewControllerTests: XCTestCase {

    func testNotification() {
        let observer = PointsOfInterestTableViewController()
        XCTAssertFalse(observer.didHandleNotification)

        // ✕ This notification will be received by all objects in process,
        // could have unintended consequences
        let name = CurrentLocationProvider.authChangedNotification
        NotificationCenter.default.post(name: name, object: nil)

        XCTAssertTrue(observer.didHandleNotification)
    }

}
```

Testing Notification Observers

How to use

- Create separate `NotificationCenter`, instead of `.default`
- Pass to `init()` and store in a new property
- Replace all uses of `NotificationCenter.default` with new property

Limits scope of tests by avoiding external effects

```
class PointsOfInterestTableViewController {

    var observer: AnyObject?
    init() {

        let name = CurrentLocationProvider.authChangedNotification
        observer = NotificationCenter.default.addObserver(forName: name, object: nil,
                                                          queue: .main) { [weak self] _ in
            self?.handleAuthChanged()
        }
    }

    var didHandleNotification = false
    func handleAuthChanged() {
        didHandleNotification = true
    }
```

```
class PointsOfInterestTableViewController {
    let notificationCenter: NotificationCenter
    var observer: AnyObject?
    init(notificationCenter: NotificationCenter) {
        self.notificationCenter = notificationCenter
        let name = CurrentLocationProvider.authChangedNotification
        observer = notificationCenter.addObserver(forName: name, object: nil,
                                                  queue: .main) { [weak self] _ in
            self?.handleAuthChanged()
        }
    }

    var didHandleNotification = false
    func handleAuthChanged() {
        didHandleNotification = true
    }
```

```
class PointsOfInterestTableViewController {
    let notificationCenter: NotificationCenter
    var observer: AnyObject?
    init(notificationCenter: NotificationCenter = .default) {
        self.notificationCenter = notificationCenter
        let name = CurrentLocationProvider.authChangedNotification
        observer = notificationCenter.addObserver(forName: name, object: nil,
                                                  queue: .main) { [weak self] _ in
            self?.handleAuthChanged()
        }
    }

    var didHandleNotification = false
    func handleAuthChanged() {
        didHandleNotification = true
    }
```

```
class PointsOfInterestTableViewControllerTests: XCTestCase {

    func testNotification() {

        let observer = PointsOfInterestTableViewController()
        XCTAssertFalse(observer.didHandleNotification)

        // ✕ This notification will be received by all objects in process,
        // could have unintended consequences
        let name = CurrentLocationProvider.authChangedNotification
        NotificationCenter.default.post(name: name, object: nil)

        XCTAssertTrue(observer.didHandleNotification)
    }

}
```

```
class PointsOfInterestTableViewControllerTests: XCTestCase {

    func testNotification() {
        let notificationCenter = NotificationCenter()
        let observer = PointsOfInterestTableViewController(notificationCenter:
                                                           notificationCenter)

        XCTAssertFalse(observer.didHandleNotification)

        // ✅ Notification posted to just this center, isolating the test
        let name = CurrentLocationProvider.authChangedNotification
        notificationCenter.post(name: name, object: nil)

        XCTAssertTrue(observer.didHandleNotification)
    }

}
```

Testing Notification Posters

How to validate that a subject posts a `Notification`?

Use a separate `NotificationCenter` again

Use `XCTNSNotificationExpectation`

```
class CurrentLocationProvider {

    static let authChangedNotification = Notification.Name("AuthChanged")

    func notifyAuthChanged() {
        let name = CurrentLocationProvider.authChangedNotification
        NotificationCenter.default.post(name: name, object: self)
    }

}
```

```
class CurrentLocationProvider {

    static let authChangedNotification = Notification.Name("AuthChanged")

    func notifyAuthChanged() {
        let name = CurrentLocationProvider.authChangedNotification
        NotificationCenter.default.post(name: name, object: self)
    }

}
```

```
class CurrentLocationProviderTests: XCTestCase {
    func testNotifyAuthChanged() {
        let poster = CurrentLocationProvider()
        var observer: AnyObject?
        let expectation = self.expectation(description: "auth changed notification")

        // ✕ Uses default NotificationCenter, not properly isolating test
        let name = CurrentLocationProvider.authChangedNotification
        observer = NotificationCenter.default.addObserver(forName: name, object: poster,
                                                          queue: .main) { _ in
            NotificationCenter.default.removeObserver(observer!)
            expectation.fulfill()
        }

        poster.notifyAuthChanged()
        wait(for: [expectation], timeout: 0)
    }
```

```
class CurrentLocationProviderTests: XCTestCase {
    func testNotifyAuthChanged() {
        let poster = CurrentLocationProvider()
        var observer: AnyObject?
        let expectation = self.expectation(description: "auth changed notification")

        // ✕ Uses default NotificationCenter, not properly isolating test
        let name = CurrentLocationProvider.authChangedNotification
        observer = NotificationCenter.default.addObserver(forName: name, object: poster,
                                                          queue: .main) { _ in
            NotificationCenter.default.removeObserver(observer!)
            expectation.fulfill()
        }

        poster.notifyAuthChanged()
        wait(for: [expectation], timeout: 0)
    }
```

```
class CurrentLocationProviderTests: XCTestCase {
    func testNotifyAuthChanged() {
        let poster = CurrentLocationProvider()

        let name = CurrentLocationProvider.authChangedNotification
        let expectation = XCTNSNotificationExpectation(name: name, object: poster)









        poster.notifyAuthChanged()
        wait(for: [expectation], timeout: 0)
    }
```

```
class CurrentLocationProvider {

    static let authChangedNotification = Notification.Name("AuthChanged")

    func notifyAuthChanged() {
        let name = CurrentLocationProvider.authChangedNotification
        NotificationCenter.default.post(name: name, object: self)
    }

}
```

```
class CurrentLocationProvider {

    static let authChangedNotification = Notification.Name("AuthChanged")

    let notificationCenter: NotificationCenter
    init(notificationCenter: NotificationCenter = .default) {
        self.notificationCenter = notificationCenter
    }

    func notifyAuthChanged() {
        let name = CurrentLocationProvider.authChangedNotification
        notificationCenter.post(name: name, object: self)
    }

}
```

```swift
class CurrentLocationProviderTests: XCTestCase {

    func testNotifyAuthChanged() {

        let poster = CurrentLocationProvider()


        let name = CurrentLocationProvider.authChangedNotification
        let expectation = XCTNSNotificationExpectation(name: name, object: poster)


        poster.notifyAuthChanged()
        wait(for: [expectation], timeout: 0)
    }

}
```

```
class CurrentLocationProviderTests: XCTestCase {

    func testNotifyAuthChanged() {
        let notificationCenter = NotificationCenter()
        let poster = CurrentLocationProvider(notificationCenter: notificationCenter)

        // ✅ Notification only sent to this specific center, isolating test
        let name = CurrentLocationProvider.authChangedNotification
        let expectation = XCTNSNotificationExpectation(name: name, object: poster,
                                                       notificationCenter: notificationCenter)

        poster.notifyAuthChanged()
        wait(for: [expectation], timeout: 0)
    }

}
```

```
class CurrentLocationProviderTests: XCTestCase {

    func testNotifyAuthChanged() {
        let notificationCenter = NotificationCenter()
        let poster = CurrentLocationProvider(notificationCenter: notificationCenter)

        // ✅ Notification only sent to this specific center, isolating test
        let name = CurrentLocationProvider.authChangedNotification
        let expectation = XCTNSNotificationExpectation(name: name, object: poster,
                                                       notificationCenter: notificationCenter)

        poster.notifyAuthChanged()
        wait(for: [expectation], timeout: 0)
    }

}
```

```
class CurrentLocationProviderTests: XCTestCase {

    func testNotifyAuthChanged() {
        let notificationCenter = NotificationCenter()
        let poster = CurrentLocationProvider(notificationCenter: notificationCenter)

        // ✅ Notification only sent to this specific center, isolating test
        let name = CurrentLocationProvider.authChangedNotification
        let expectation = XCTNSNotificationExpectation(name: name, object: poster,
                                                       notificationCenter: notificationCenter)

        poster.notifyAuthChanged()
        wait(for: [expectation], timeout: 0)
    }

}
```

```
class CurrentLocationProviderTests: XCTestCase {

    func testNotifyAuthChanged() {
        let notificationCenter = NotificationCenter()
        let poster = CurrentLocationProvider(notificationCenter: notificationCenter)

        // ✅ Notification only sent to this specific center, isolating test
        let name = CurrentLocationProvider.authChangedNotification
        let expectation = XCTNSNotificationExpectation(name: name, object: poster,
                                                       notificationCenter: notificationCenter)

        poster.notifyAuthChanged()
        wait(for: [expectation], timeout: 0)
    }

}
```

Testing network requests

Working with notifications

Mocking with protocols

Test execution speed

Mocking with Protocols

Classes often interact with other classes in app or SDK

Many SDK classes cannot be created directly

Delegate protocols make testing more challenging

Solution: Mock interface of external class using protocol

```swift
import CoreLocation

class CurrentLocationProvider: NSObject {

    let locationManager = CLLocationManager()
    override init() {
        super.init()
        self.locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters
        self.locationManager.delegate = self
    }

    // ...
```

```
class CurrentLocationProvider: NSObject {

    var currentLocationCheckCallback: ((CLLocation) -> Void)?
    func checkCurrentLocation(completion: @escaping (Bool) -> Void) {
        self.currentLocationCheckCallback = { [unowned self] location in
            completion(self.isPointOfInterest(location))
        }
        locationManager.requestLocation()
    }

    func isPointOfInterest(_ location: CLLocation) -> Bool {
        // Perform check...
    }

}
```

```
class CurrentLocationProvider: NSObject {

    var currentLocationCheckCallback: ((CLLocation) -> Void)?
    func checkCurrentLocation(completion: @escaping (Bool) -> Void) {
        self.currentLocationCheckCallback = { [unowned self] location in
            completion(self.isPointOfInterest(location))
        }
        locationManager.requestLocation()
    }

    func isPointOfInterest(_ location: CLLocation) -> Bool {
        // Perform check...
    }

}
```

```
extension CurrentLocationProvider: CLLocationManagerDelegate {
    func locationManager(_ manager: CLLocationManager, didUpdateLocations locs: [CLLocation]){
        guard let location = locs.first else { return }
        self.currentLocationCheckCallback?(location)
        self.currentLocationCheckCallback = nil
    }
}
```

```
class CurrentLocationProviderTests: XCTestCase {
    func testCheckCurrentLocation() {
        let provider = CurrentLocationProvider()

        XCTAssertNotEqual(provider.locationManager.desiredAccuracy, 0)
        XCTAssertNotNil(provider.locationManager.delegate)

        let completionExpectation = expectation(description: "completion")
        provider.checkCurrentLocation { isPointOfInterest in
            XCTAssertTrue(isPointOfInterest)
            completionExpectation.fulfill()
        }

        // ✕ No way to mock the current location or confirm requestLocation() was called
        wait(for: [completionExpectation], timeout: 1)
    }
}
```

```
class CurrentLocationProviderTests: XCTestCase {
    func testCheckCurrentLocation() {
        let provider = CurrentLocationProvider()

        XCTAssertNotEqual(provider.locationManager.desiredAccuracy, 0)
        XCTAssertNotNil(provider.locationManager.delegate)

        let completionExpectation = expectation(description: "completion")
        provider.checkCurrentLocation { isPointOfInterest in
            XCTAssertTrue(isPointOfInterest)
            completionExpectation.fulfill()
        }

        // ✕ No way to mock the current location or confirm requestLocation() was called
        wait(for: [completionExpectation], timeout: 1)
    }
}
```

```
class CurrentLocationProviderTests: XCTestCase {
    func testCheckCurrentLocation() {
        let provider = CurrentLocationProvider()

        XCTAssertNotEqual(provider.locationManager.desiredAccuracy, 0)
        XCTAssertNotNil(provider.locationManager.delegate)

        let completionExpectation = expectation(description: "completion")
        provider.checkCurrentLocation { isPointOfInterest in
            XCTAssertTrue(isPointOfInterest)
            completionExpectation.fulfill()
        }

        // ✕ No way to mock the current location or confirm requestLocation() was called
        wait(for: [completionExpectation], timeout: 1)
    }
}
```

```
class CurrentLocationProviderTests: XCTestCase {
    func testCheckCurrentLocation() {
        let provider = CurrentLocationProvider()

        XCTAssertNotEqual(provider.locationManager.desiredAccuracy, 0)
        XCTAssertNotNil(provider.locationManager.delegate)

        let completionExpectation = expectation(description: "completion")
        provider.checkCurrentLocation { isPointOfInterest in
            XCTAssertTrue(isPointOfInterest)
            completionExpectation.fulfill()
        }

        // ✕ No way to mock the current location or confirm requestLocation() was called
        wait(for: [completionExpectation], timeout: 1)
    }
}
```

```
class CurrentLocationProviderTests: XCTestCase {
    func testCheckCurrentLocation() {
        let provider = CurrentLocationProvider()

        XCTAssertNotEqual(provider.locationManager.desiredAccuracy, 0)
        XCTAssertNotNil(provider.locationManager.delegate)

        let completionExpectation = expectation(description: "completion")
        provider.checkCurrentLocation { isPointOfInterest in
            XCTAssertTrue(isPointOfInterest)
            completionExpectation.fulfill()
        }

        // ✕ No way to mock the current location or confirm requestLocation() was called
        wait(for: [completionExpectation], timeout: 1)
    }
}
```

Mocking Using a Subclass

Subclassing the external class in tests and overriding methods

Can work, but risky

Some SDK classes cannot be subclassed

Easy to forget to override methods

Mocking Using a Subclass

Subclassing the external class in tests and overriding methods

Can work, but risky

Some SDK classes cannot be subclassed

Easy to forget to override methods

```
class CurrentLocationProvider: NSObject {
    let locationManager = CLLocationManager()
    override init() {
        super.init()
        self.locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters
        self.locationManager.delegate = self
    }
}
```

```
class CurrentLocationProvider: NSObject {
    let locationManager = CLLocationManager()
    override init() {
        super.init()
        self.locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters
        self.locationManager.delegate = self
    }
}
```

```
protocol LocationFetcher {
    var delegate: CLLocationManagerDelegate? { get set }
    var desiredAccuracy: CLLocationAccuracy { get set }
    func requestLocation()
}
extension CLLocationManager: LocationFetcher {}
```

```
class CurrentLocationProvider: NSObject {
    var locationFetcher: LocationFetcher
    init(locationFetcher: LocationFetcher) {
        self.locationFetcher = locationFetcher
        super.init()
        self.locationFetcher.desiredAccuracy = kCLLocationAccuracyHundredMeters
        self.locationFetcher.delegate = self
    }
}

protocol LocationFetcher {
    var delegate: CLLocationManagerDelegate? { get set }
    var desiredAccuracy: CLLocationAccuracy { get set }
    func requestLocation()
}
extension CLLocationManager: LocationFetcher {}
```

```
class CurrentLocationProvider: NSObject {
    var locationFetcher: LocationFetcher
    init(locationFetcher: LocationFetcher = CLLocationManager()) {
        self.locationFetcher = locationFetcher
        super.init()
        self.locationFetcher.desiredAccuracy = kCLLocationAccuracyHundredMeters
        self.locationFetcher.delegate = self
    }
}

protocol LocationFetcher {
    var delegate: CLLocationManagerDelegate? { get set }
    var desiredAccuracy: CLLocationAccuracy { get set }
    func requestLocation()
}
extension CLLocationManager: LocationFetcher {}
```

```
class CurrentLocationProvider: NSObject {

    var currentLocationCheckCallback: ((CLLocation) -> Void)?
    func checkCurrentLocation(completion: @escaping (Bool) -> Void) {
        self.currentLocationCheckCallback = { [unowned self] location in
            completion(self.isPointOfInterest(location))
        }
        locationManager.requestLocation()
    }

    func isPointOfInterest(_ location: CLLocation) -> Bool {
        // Perform check...
    }

}
```

```
class CurrentLocationProvider: NSObject {

    var currentLocationCheckCallback: ((CLLocation) -> Void)?
    func checkCurrentLocation(completion: @escaping (Bool) -> Void) {
        self.currentLocationCheckCallback = { [unowned self] location in
            completion(self.isPointOfInterest(location))
        }
        locationFetcher.requestLocation()
    }

    func isPointOfInterest(_ location: CLLocation) -> Bool {
        // Perform check...
    }

}
```

```
extension CurrentLocationProvider: CLLocationManagerDelegate {
    func locationManager(_ manager: CLLocationManager, didUpdateLocations locs: [CLLocation]){
        guard let location = locs.first else { return }
        self.currentLocationCheckCallback?(location)
        self.currentLocationCheckCallback = nil
    }
}
```

```
protocol LocationFetcher {
    var delegate: CLLocationManagerDelegate? { get set }
    // ...
}
```

```
extension CLLocationManager: LocationFetcher {}
```

```
protocol LocationFetcher {
    var locationFetcherDelegate: LocationFetcherDelegate? { get set }
    // ...
}
protocol LocationFetcherDelegate: class {
    func locationFetcher(_ fetcher: LocationFetcher, didUpdateLocations locs: [CLLocation])
}

extension CLLocationManager: LocationFetcher {}
```

```
protocol LocationFetcher {
    var locationFetcherDelegate: LocationFetcherDelegate? { get set }
    // ...
}
protocol LocationFetcherDelegate: class {
    func locationFetcher(_ fetcher: LocationFetcher, didUpdateLocations locs: [CLLocation])
}

extension CLLocationManager: LocationFetcher {
    var locationFetcherDelegate: LocationFetcherDelegate? {
        get { return delegate as! LocationFetcherDelegate? }
        set { delegate = newValue as! CLLocationManagerDelegate? }
    }
}
```

```
protocol LocationFetcher {
    var locationFetcherDelegate: LocationFetcherDelegate? { get set }
    // ...
}
protocol LocationFetcherDelegate: class {
    func locationFetcher(_ fetcher: LocationFetcher, didUpdateLocations locs: [CLLocation])
}

class CurrentLocationProvider: NSObject {
    var locationFetcher: LocationFetcher
    init(locationFetcher: LocationFetcher = CLLocationManager()) {
        // ...
        self.locationFetcher.delegate = self
    }
}
```

```
protocol LocationFetcher {
    var locationFetcherDelegate: LocationFetcherDelegate? { get set }
    // ...
}
protocol LocationFetcherDelegate: class {
    func locationFetcher(_ fetcher: LocationFetcher, didUpdateLocations locs: [CLLocation])
}

class CurrentLocationProvider: NSObject {
    var locationFetcher: LocationFetcher
    init(locationFetcher: LocationFetcher = CLLocationManager()) {
        // ...
        self.locationFetcher.locationFetcherDelegate = self
    }
}
```

```
extension CurrentLocationProvider: CLLocationManagerDelegate {
    func locationManager(_ manager: CLLocationManager, didUpdateLocations locs: [CLLocation]){
        guard let location = locs.first else { return }
        self.currentLocationCheckCallback?(location)
        self.currentLocationCheckCallback = nil
    }
}
```

```
extension CurrentLocationProvider: LocationFetcherDelegate {
    func locationFetcher(_ fetcher: LocationFetcher, didUpdateLocations locs: [CLLocation]){
        guard let location = locs.first else { return }
        self.currentLocationCheckCallback?(location)
        self.currentLocationCheckCallback = nil
    }
}
```

```
extension CurrentLocationProvider: LocationFetcherDelegate {
    func locationFetcher(_ fetcher: LocationFetcher, didUpdateLocations locs: [CLLocation]){
        guard let location = locs.first else { return }
        self.currentLocationCheckCallback?(location)
        self.currentLocationCheckCallback = nil
    }
}

extension CurrentLocationProvider: CLLocationManagerDelegate {
    func locationManager(_ manager: CLLocationManager, didUpdateLocations locs: [CLLocation]){
        self.locationFetcher(manager, didUpdateLocations: locs)
    }
}
```

```
extension CurrentLocationProvider: LocationFetcherDelegate {
    func locationFetcher(_ fetcher: LocationFetcher, didUpdateLocations locs: [CLLocation]){
        guard let location = locs.first else { return }
        self.currentLocationCheckCallback?(location)
        self.currentLocationCheckCallback = nil
    }
}

extension CurrentLocationProvider: CLLocationManagerDelegate {
    func locationManager(_ manager: CLLocationManager, didUpdateLocations locs: [CLLocation]){
        self.locationFetcher(manager, didUpdateLocations: locs)
    }
}
```

```
class CurrentLocationProviderTests: XCTestCase {
    struct MockLocationFetcher: LocationFetcher {
        weak var locationFetcherDelegate: LocationFetcherDelegate?

        var desiredAccuracy: CLLocationAccuracy = 0

        var handleRequestLocation: (() -> CLLocation)?
        func requestLocation() {
            guard let location = handleRequestLocation?() else { return }
            locationFetcherDelegate?.locationFetcher(self, didUpdateLocations: [location])
        }
    }
```

```
class CurrentLocationProviderTests: XCTestCase {
    struct MockLocationFetcher: LocationFetcher {
        weak var locationFetcherDelegate: LocationFetcherDelegate?

        var desiredAccuracy: CLLocationAccuracy = 0

        var handleRequestLocation: (() -> CLLocation)?
        func requestLocation() {
            guard let location = handleRequestLocation?() else { return }
            locationFetcherDelegate?.locationFetcher(self, didUpdateLocations: [location])
        }
    }
```

```
class CurrentLocationProviderTests: XCTestCase {
    struct MockLocationFetcher: LocationFetcher {
        weak var locationFetcherDelegate: LocationFetcherDelegate?

        var desiredAccuracy: CLLocationAccuracy = 0

        var handleRequestLocation: (() -> CLLocation)?
        func requestLocation() {
            guard let location = handleRequestLocation?() else { return }
            locationFetcherDelegate?.locationFetcher(self, didUpdateLocations: [location])
        }
    }
```

```swift
func testCheckCurrentLocation() {
    var locationFetcher = MockLocationFetcher()
    let requestLocationExpectation = expectation(description: "request location")
    locationFetcher.handleRequestLocation = {
        requestLocationExpectation.fulfill()
        return CLLocation(latitude: 37.3293, longitude: -121.8893)
    }
    let provider = CurrentLocationProvider(locationFetcher: locationFetcher)
    let completionExpectation = expectation(description: "completion")
    provider.checkCurrentLocation { isPointOfInterest in
        XCTAssertTrue(isPointOfInterest)
        completionExpectation.fulfill()
    }

    // ✅ Can mock the current location
    wait(for: [requestLocationExpectation, completionExpectation], timeout: 1)
}
```

```swift
func testCheckCurrentLocation() {
    var locationFetcher = MockLocationFetcher()
    let requestLocationExpectation = expectation(description: "request location")
    locationFetcher.handleRequestLocation = {
        requestLocationExpectation.fulfill()
        return CLLocation(latitude: 37.3293, longitude: -121.8893)
    }
    let provider = CurrentLocationProvider(locationFetcher: locationFetcher)
    let completionExpectation = expectation(description: "completion")
    provider.checkCurrentLocation { isPointOfInterest in
        XCTAssertTrue(isPointOfInterest)
        completionExpectation.fulfill()
    }

    // ✅ Can mock the current location
    wait(for: [requestLocationExpectation, completionExpectation], timeout: 1)
}
```

```swift
func testCheckCurrentLocation() {
    var locationFetcher = MockLocationFetcher()
    let requestLocationExpectation = expectation(description: "request location")
    locationFetcher.handleRequestLocation = {
        requestLocationExpectation.fulfill()
        return CLLocation(latitude: 37.3293, longitude: -121.8893)
    }
    let provider = CurrentLocationProvider(locationFetcher: locationFetcher)
    let completionExpectation = expectation(description: "completion")
    provider.checkCurrentLocation { isPointOfInterest in
        XCTAssertTrue(isPointOfInterest)
        completionExpectation.fulfill()
    }

    // ✅ Can mock the current location
    wait(for: [requestLocationExpectation, completionExpectation], timeout: 1)
}
```

```
func testCheckCurrentLocation() {
    var locationFetcher = MockLocationFetcher()
    let requestLocationExpectation = expectation(description: "request location")
    locationFetcher.handleRequestLocation = {
        requestLocationExpectation.fulfill()
        return CLLocation(latitude: 37.3293, longitude: -121.8893)
    }
    let provider = CurrentLocationProvider(locationFetcher: locationFetcher)
    let completionExpectation = expectation(description: "completion")
    provider.checkCurrentLocation { isPointOfInterest in
        XCTAssertTrue(isPointOfInterest)
        completionExpectation.fulfill()
    }

    // ✅ Can mock the current location
    wait(for: [requestLocationExpectation, completionExpectation], timeout: 1)
}
```

```swift
func testCheckCurrentLocation() {
    var locationFetcher = MockLocationFetcher()
    let requestLocationExpectation = expectation(description: "request location")
    locationFetcher.handleRequestLocation = {
        requestLocationExpectation.fulfill()
        return CLLocation(latitude: 37.3293, longitude: -121.8893)
    }
    let provider = CurrentLocationProvider(locationFetcher: locationFetcher)
    let completionExpectation = expectation(description: "completion")
    provider.checkCurrentLocation { isPointOfInterest in
        XCTAssertTrue(isPointOfInterest)
        completionExpectation.fulfill()
    }

    // ✅ Can mock the current location
    wait(for: [requestLocationExpectation, completionExpectation], timeout: 1)
}
```

```swift
func testCheckCurrentLocation() {
    var locationFetcher = MockLocationFetcher()
    let requestLocationExpectation = expectation(description: "request location")
    locationFetcher.handleRequestLocation = {
        requestLocationExpectation.fulfill()
        return CLLocation(latitude: 37.3293, longitude: -121.8893)
    }
    let provider = CurrentLocationProvider(locationFetcher: locationFetcher)
    let completionExpectation = expectation(description: "completion")
    provider.checkCurrentLocation { isPointOfInterest in
        XCTAssertTrue(isPointOfInterest)
        completionExpectation.fulfill()
    }

    // ✅ Can mock the current location
    wait(for: [requestLocationExpectation, completionExpectation], timeout: 1)
}
```

```swift
func testCheckCurrentLocation() {
    var locationFetcher = MockLocationFetcher()
    let requestLocationExpectation = expectation(description: "request location")
    locationFetcher.handleRequestLocation = {
        requestLocationExpectation.fulfill()
        return CLLocation(latitude: 37.3293, longitude: -121.8893)
    }
    let provider = CurrentLocationProvider(locationFetcher: locationFetcher)
    let completionExpectation = expectation(description: "completion")
    provider.checkCurrentLocation { isPointOfInterest in
        XCTAssertTrue(isPointOfInterest)
        completionExpectation.fulfill()
    }

    // ✅ Can mock the current location
    wait(for: [requestLocationExpectation, completionExpectation], timeout: 1)
}
```

Mocking with Protocols

Recap

Define a protocol representing the external interface

Define extension on the external class conforming to the protocol

Replace all usage of external class with the protocol

Set the external reference via initializer or a property, using the protocol type

Mocking Delegates with Protocols

Recap

Define delegate protocol with interfaces your code implements

Replace subject type with mock protocol defined earlier

In mock protocol, rename delegate property

In extension on original type, implement mock delegate property and convert

Testing network requests

Working with notifications

Mocking with protocols

Test execution speed

Test Execution Speed

Slow tests hinder developer productivity

Want tests to run fast

Artificial delays should not be necessary with sufficient mocking

```
class FeaturedPlaceManager {

    var currentPlace: Place

    func scheduleNextPlace() {
        // Show next place after 10 seconds
        Timer.scheduledTimer(withTimeInterval: 10, repeats: false) { [weak self] _ in
            self?.showNextPlace()
        }
    }

    func showNextPlace() {
        // Set currentPlace to next place...
    }

}
```

```
class FeaturedPlaceManagerTests: XCTestCase {

    func testScheduleNextPlace() {
        let manager = FeaturedPlaceManager()

        let beforePlace = manager.currentPlace
        manager.scheduleNextPlace()

        // ✕ Slow! Not ideal
        RunLoop.current.run(until: Date(timeIntervalSinceNow: 11))

        XCTAssertNotEqual(manager.currentPlace, beforePlace)
    }

}
```

```
class FeaturedPlaceManager {

    var currentPlace: Place


    func scheduleNextPlace() {
        Timer.scheduledTimer(withTimeInterval: 10, repeats: false) { [weak self] _ in
            self?.showNextPlace()
        }
    }

    func showNextPlace() {
        // Set currentPlace to next place...
    }

}
```

```
class FeaturedPlaceManager {

    var currentPlace: Place
    var interval: TimeInterval = 10

    func scheduleNextPlace() {
        Timer.scheduledTimer(withTimeInterval: interval, repeats: false) { [weak self] _ in
            self?.showNextPlace()
        }
    }

    func showNextPlace() {
        // Set currentPlace to next place...
    }

}
```

```
class FeaturedPlaceManagerTests: XCTestCase {

    func testScheduleNextPlace() {
        let manager = FeaturedPlaceManager()
        manager.interval = 1

        let beforePlace = manager.currentPlace
        manager.scheduleNextPlace()

        // ⚠️ Less slow but still timing-dependent
        RunLoop.current.run(until: Date(timeIntervalSinceNow: 2))

        XCTAssertNotEqual(manager.currentPlace, beforePlace)
    }

}
```

Testing Delayed Actions

Without the delay

How to use

- Identify the “delay” technique (e.g. `Timer`, `DispatchQueue.asyncAfter`)
- Mock this mechanism during tests
- Invoke delayed action immediately

```
class FeaturedPlaceManager {



    func scheduleNextPlace() {
        Timer.scheduledTimer(withTimeInterval: 10, repeats: false) { [weak self] _ in
            self?.showNextPlace()
        }
    }


    func showNextPlace() { /* ... */ }

}
```

```
class FeaturedPlaceManager {



    func scheduleNextPlace() {
        Timer.scheduledTimer(withTimeInterval: 10, repeats: false) { [weak self] _ in
            self?.showNextPlace()
        }
    }


    func showNextPlace() { /* ... */ }

}
```

```
class FeaturedPlaceManager {

    let runLoop = RunLoop.current

    func scheduleNextPlace() {
        let timer = Timer(timeInterval: 10, repeats: false) { [weak self] _ in
            self?.showNextPlace()
        }
        runLoop.add(timer, forMode: .default)
    }

    func showNextPlace() { /* ... */ }

}
```

```swift
protocol TimerScheduler {
    func add(_ timer: Timer, forMode mode: RunLoop.Mode)
}

extension RunLoop: TimerScheduler {}
```

```
class FeaturedPlaceManager {

    let runLoop = RunLoop.current




    func scheduleNextPlace() {
        let timer = Timer(timeInterval: 10, repeats: false) { [weak self] _ in
            self?.showNextPlace()
        }
        runLoop.add(timer, forMode: .default)
    }

    func showNextPlace() { /* ... */ }

}
```

```
class FeaturedPlaceManager {

    let timerScheduler: TimerScheduler
    init(timerScheduler: TimerScheduler) {
        self.timerScheduler = timerScheduler
    }

    func scheduleNextPlace() {
        let timer = Timer(timeInterval: 10, repeats: false) { [weak self] _ in
            self?.showNextPlace()
        }
        timerScheduler.add(timer, forMode: .default)
    }

    func showNextPlace() { /* ... */ }

}
```

```
class FeaturedPlaceManagerTests: XCTestCase {

    struct MockTimerScheduler: TimerScheduler {
        var handleAddTimer: ((_ timer: Timer) -> Void)?

        func add(_ timer: Timer, forMode mode: RunLoop.Mode) {
            handleAddTimer?(timer)
        }
    }

    // ...
```

```
class FeaturedPlaceManagerTests: XCTestCase {
    func testScheduleNextPlace() {
        var timerScheduler = MockTimerScheduler()
        var timerDelay = TimeInterval(0)
        timerScheduler.handleAddTimer = { timer in
            timerDelay = timer.fireDate.timeIntervalSinceNow
            timer.fire()
        }

        let manager = FeaturedPlaceManager(timerScheduler: timerScheduler)
        let beforePlace = manager.currentPlace
        manager.scheduleNextPlace()

        // ✅ No delay!
        XCTAssertEqual(timerDelay, 10, accuracy: 1)
        XCTAssertNotEqual(manager.currentPlace, beforePlace)
    }
```

```swift
class FeaturedPlaceManagerTests: XCTestCase {
    func testScheduleNextPlace() {
        var timerScheduler = MockTimerScheduler()
        var timerDelay = TimeInterval(0)
        timerScheduler.handleAddTimer = { timer in
            timerDelay = timer.fireDate.timeIntervalSinceNow
            timer.fire()
        }

        let manager = FeaturedPlaceManager(timerScheduler: timerScheduler)
        let beforePlace = manager.currentPlace
        manager.scheduleNextPlace()

        // ✅ No delay!
        XCTAssertEqual(timerDelay, 10, accuracy: 1)
        XCTAssertNotEqual(manager.currentPlace, beforePlace)
    }
```

```
class FeaturedPlaceManagerTests: XCTestCase {
    func testScheduleNextPlace() {
        var timerScheduler = MockTimerScheduler()
        var timerDelay = TimeInterval(0)
        timerScheduler.handleAddTimer = { timer in
            timerDelay = timer.fireDate.timeIntervalSinceNow
            timer.fire()
        }

        let manager = FeaturedPlaceManager(timerScheduler: timerScheduler)
        let beforePlace = manager.currentPlace
        manager.scheduleNextPlace()

        // ✅ No delay!
        XCTAssertEqual(timerDelay, 10, accuracy: 1)
        XCTAssertNotEqual(manager.currentPlace, beforePlace)
    }
```

```
class FeaturedPlaceManagerTests: XCTestCase {
    func testScheduleNextPlace() {
        var timerScheduler = MockTimerScheduler()
        var timerDelay = TimeInterval(0)
        timerScheduler.handleAddTimer = { timer in
            timerDelay = timer.fireDate.timeIntervalSinceNow
            timer.fire()
        }

        let manager = FeaturedPlaceManager(timerScheduler: timerScheduler)
        let beforePlace = manager.currentPlace
        manager.scheduleNextPlace()

        // ✅ No delay!
        XCTAssertEqual(timerDelay, 10, accuracy: 1)
        XCTAssertNotEqual(manager.currentPlace, beforePlace)
    }
```

```
class FeaturedPlaceManagerTests: XCTestCase {
    func testScheduleNextPlace() {
        var timerScheduler = MockTimerScheduler()
        var timerDelay = TimeInterval(0)
        timerScheduler.handleAddTimer = { timer in
            timerDelay = timer.fireDate.timeIntervalSinceNow
            timer.fire()
        }

        let manager = FeaturedPlaceManager(timerScheduler: timerScheduler)
        let beforePlace = manager.currentPlace
        manager.scheduleNextPlace()

        // ✅ No delay!
        XCTAssertEqual(timerDelay, 10, accuracy: 1)
        XCTAssertNotEqual(manager.currentPlace, beforePlace)
    }
```

```
class FeaturedPlaceManagerTests: XCTestCase {
    func testScheduleNextPlace() {
        var timerScheduler = MockTimerScheduler()
        var timerDelay = TimeInterval(0)
        timerScheduler.handleAddTimer = { timer in
            timerDelay = timer.fireDate.timeIntervalSinceNow
            timer.fire()
        }

        let manager = FeaturedPlaceManager(timerScheduler: timerScheduler)
        let beforePlace = manager.currentPlace
        manager.scheduleNextPlace()

        // ✅ No delay!
        XCTAssertEqual(timerDelay, 10, accuracy: 1)
        XCTAssertNotEqual(manager.currentPlace, beforePlace)
    }
```

Testing Delayed Actions

Delay is eliminated using this technique

Majority of tests should be direct, without mocking delays

Setting Expectations

Be mindful when using `XCTNSPredicateExpectation`

Slower and best suited for UI tests

Use faster, callback-based expectations in unit tests:

- `XCTestExpectation`
- `XCTNSNotificationExpectation`
- `XCTKVOExpectation`

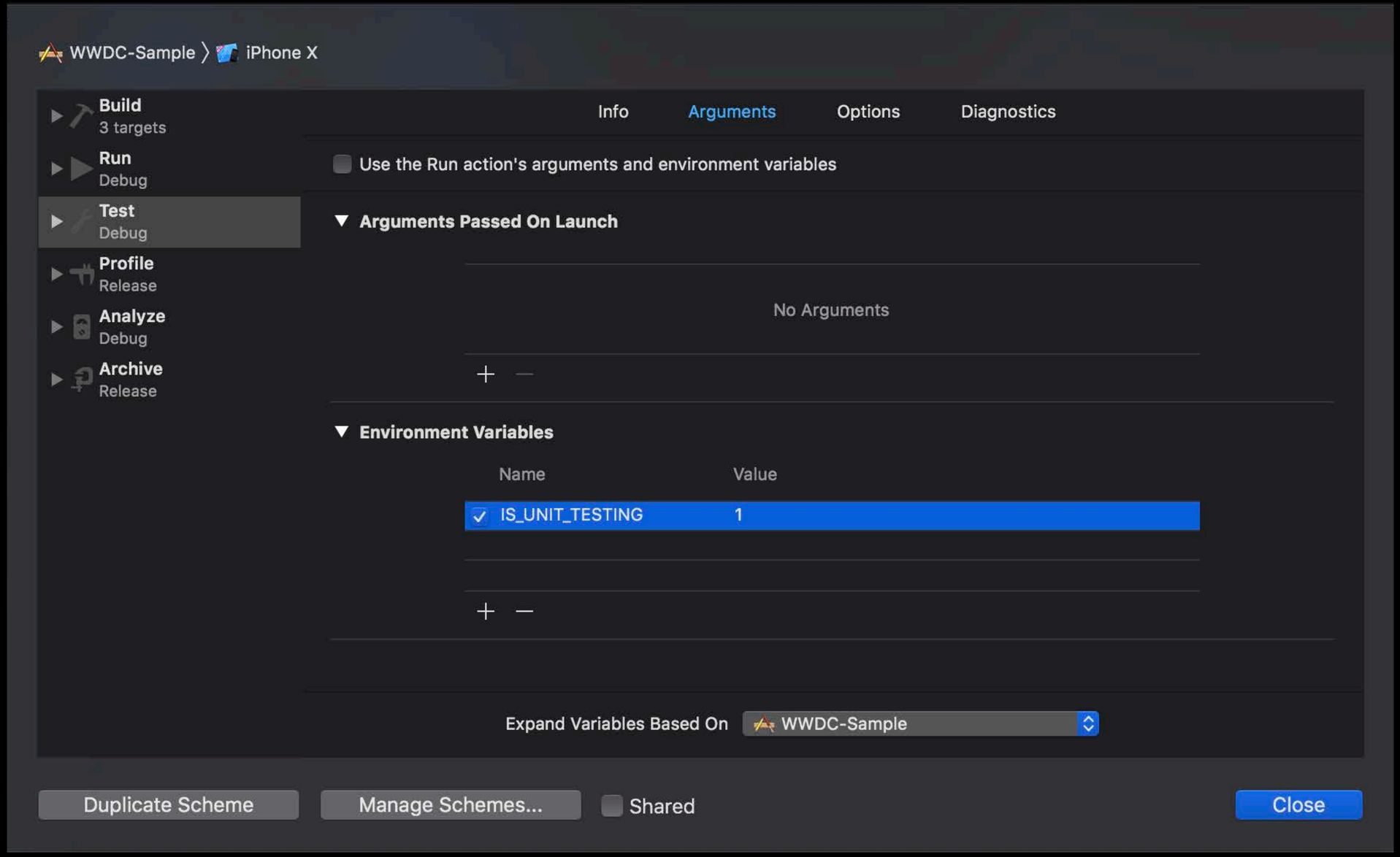
Optimizing App Launch When Testing

Avoid unnecessary work when app is launched as unit test host

Testing begins after ‘app did finish launching’

Set custom scheme environment variables/launch arguments for testing

Optimizing App Launch When Testing



Optimizing App Launch When Testing

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions opts: …) -> Bool
{
    let isUnitTesting = ProcessInfo.processInfo.environment["IS_UNIT_TESTING"] == "1"
    if isUnitTesting == false {
        // Do UI-related setup, which can be skipped when testing
    }
    return true
}
```

Summary

Testing network requests

Working with notifications

Mocking with protocols

Test execution speed

More Information

https://developer.apple.com/wwdc18/417


WWDC18