What is a Backend?

- Server and database external to the mobile device
- Located on remote servers set up by developers
- Provides app information to users
- Allows for:
  - Reuse of data across user-base
  - Security controlled by the developers
  - More reliable storage
When to Offload

- When should you offload to the backend?
  - Information is regularly updated
  - Systems are regularly updated
  - Calculations require better hardware
  - User data collected for analytics etc
  - “Cloud” model is extremely popular right now
Network

- Computers communicate via sockets (ports)
- Port provides connection into computer
- Thousands of ports available on a computer
- Ports can be turned off for security
- Predefined ports have specific usages
  - Port 80 (http/insecure)
  - Port 443 (https/secure)
TCP/IP

- Transfer Control Protocol / Internet Protocol
- Dominant network protocol
- Two addressing protocols, IPv4 and IPv6
- App must be able to talk to both networks:
  - NSURLConnection
  - URLSession
Networking Frameworks

- NSURLConnection is older framework
- URLSession replaces NSURLConnection
  - Recommended networking framework
- AFNetworking / AlamoFire are third-party frameworks
  - Popular and stable
  - Built on top of Connection and Session
- AlamoFire is Swift version
URLSession

- Provides API for downloading/uploading content over the Internet
- Built-in support for authentication and execution of background tasks
- Supports data, file, ftp, http and https URL schemes
- Support for proxy servers
- Allows canceling, restarting, resuming and suspending tasks and downloads
Configuration Objects

- URLSessionConfiguration defines behaviors and policies for uploading and downloading
  - **Shared** handles basic requests
  - **Default** allows for incremental data transfer
  - **Ephemeral** does not write caches, cookies, or credentials to persistent store
  - **Background** allows download/upload as a background task
Choosing a Configuration

- **Shared** has limited customization but allows for easy URL fetching
  - Does not create a configuration object
  - Accesses the property directly

- **Default** is similar to *shared* before customizing but can obtain data incrementally
  - Creates default configuration object
  - Stores credentials in user’s keychain

- **Ephemeral** allows for private sessions
  - Stores all data to RAM
  - Only writes to disk when told to write contents to file

- **Background** sessions can run even when the app is off or suspended
  - isDiscretionary allow session to optimize for performance
Additional Configurations

- Timeout values
- Caching policies
- Security policies
- Background transfers
- HTTP and proxy policies
Session Tasks

- **URLSessionTask** performs actual work for retrieving data

- **Data tasks** send and receive data using **NSData objects**
  - Used for short requests and small amounts of data
  - Uses HTTP GET

- **Upload tasks** send and receive large amount of data
  - Uses HTTP POST and PUT

- **Download tasks** retrieve data in the form of a file
Session Delegates

- A session’s delegate tracks when events occur:
  - Authentication requests from server
  - Data arriving from server
  - Any failures in session
- If no event-tracking features are require, can pass in delegate as nil
Custom Protocol

• Implement basic delegate functionality via a protocol

• Handles errors and return data:

```swift
protocol DataProtocol {
    func responseDataHandler(data: [return type])
    func responseError(message: String)
}
```

• Associated with view controller (like any other protocol)
Session Overview

URLSession

URLSessionConfiguration

Cache Cookies Cred Protocols

Options

Delegate

NSURLSessionTask
GET /foo.html

NSURLSessionTask
GET /bar.png

NSURLSessionTask
POST /foo.bar

(www.raywenderlich.com)
Session Summary

- URLSession: high-level session object
- URLSessionTask: object that contains one or more task objects
- URLSessionDataTask: subclass of sessionTask for direct data retrieval
  - dataTask(with: URLRequest)
  - dataTask(with: URL)
- Tasks begin in suspended state
- Start task by calling resume
URLSessionDemo
App Transport Security

- Handles security between app and web
- Required for app using NSURLConnection or URLSession
- Will reject insecure connections
- Exceptions can be added in Info.plist
  - App Transport Security Settings
  - Be careful how you do this!
Quiz Question!

What is the basic URLSessionConfiguration?

A. Default
B. Ephemeral
C. Background
D. Shared
(AFNetworking/Alamofire)

- Built on top of URLSession and Foundation’s URL Loading System
- Allows for easier HTTP requests and response handling without all the boilerplate code
- Example:

```swift
let request = Alamofire.request(.GET, "http://httpbin.org/get")

request.validate() //Checks response status code

request.response { (request, response, data, error) in
}
```
Network Payloads

- Data to be sent across the network
- Structured to be read on both ends:
  - **JSON** (JavaScript Object Notation)
  - **XML** (eXtensible Markup Language)
XML

- Tags defined by angle brackets
- Content placed within tags
- Tags can be nested

```xml
<element>
  <item>First item</item>
  <item>Second item</item>
</element>
```

- Nested elements are children of the enclosing elements
XML Example

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<data>
  <current_condition>
    <cloudcover>16</cloudcover>
    <humidity>59</humidity>
    <observation_time>09:09 PM</observation_time>
  </current_condition>
</data>
```
Model for objects and arrays

Presents hierarchical structures (like XML)

Easier to structure and parse

Objects are unordered name/value pairs: `{name1: value1, name2: value2}`

Arrays are order collections of values: `[value1, value2]`

Objects and arrays can be values
JSON Example

```json
{
    "data": {
        "current_condition": [
            {
                "cloudcover": "16",
                "humidity": "59",
                "observation_time": "09:09 PM",
            }
        ]
    }
}
```
HTTP

- Hypertext Transfer Protocol
- Nodes contain hypertext
- Hyperlinks connect between nodes
- HTTP allows for the exchange and transfer of hypertext
- HTTP requests submitted to the server
- Server responds with message information and requested content
Messages

- HTTP messages consists of:
  - Initial line
    - Request: METHOD pathToRequestedURI HTTP/x.x
    - Response: HTTP/x.x status_code reason
  - Header information
    - Header-Name: value
    - [https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html](https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html)
  - Message body
Status Codes

- Sent in initial line of responses
- Indicate success or failure of request
- Common status codes:
  - 200 (ok)
  - 404 (not found)
  - 301 (moved permanently)
  - 500 (server error)
Other Status Codes

- 1xx (request received, continuing to process)
- 2xx (request received, understood, accepted and processed)
- 3xx (client must take additional action to complete request)
- 4xx (client error in request)
- 5xx (server error in fulfilling request)
GET and HEAD Methods

- Retrieves information from given server at specified URI
- Does not modify data in any way
- GET retrieves headers and message body
- HEAD retrieves only the headers
POST and PUT Methods

- Sends data to server
- Enclosed entity (block of data) sent with request
- POST method submits data to be processed by resource at URI
  - URI is a program for handing this data
- Response is program’s output
- PUT method submits data to be stored at URI
DELETE

* Removes all representations of the resource identified by URI

* Should flag cached representations of resource as stale

  * That is, *all* representations — even those in cache — should be removed from the client’s view of the server

* Server can implement by deleting data or moving data to an inaccessible location
Quiz Question!

True or False (A or B): The POST and PUT method both send data to a server.
REST

- REpresentational State Transfer
- Architecture and practices to provide interoperability between computers across the Internet
- Compliant systems access and manipulate web resources using predefined operations
- Responses provided via XML, HTML or JSON
- “Best practices” for using HTTP
REST Properties

- Performance
- Scalability
- Simplicity
- Modifiability
- Visibility
- Portability
- Reliability
REST Constraints

- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (optional)
RESTful APIs

- Web service APIs that adhere to REST constraints
- Must have:
  - A base URL
  - Internet media type (MIME type)
  - Standard HTTP methods