MONASH University

FIT2081
Mobile Application
Development

Networking & APIs

Week 7

Jiazhou 'Joe' Liu

Dr Jiazhou ‘Joe’ Liu

Assistant Lecturer
Embodied Visualisation Group
Department of Human-Centred Computing
Faculty of IT

Teaching Experience:
Data Visualisation
Web Fundamentals
Full-Stack Development
User Interface Design and Usability
Computer/Data Science Project
Research Methods

My research focuses on novel and effective interactions in immersive environments (VR, AR) for visualisation view management and AI in digital health and construction domain

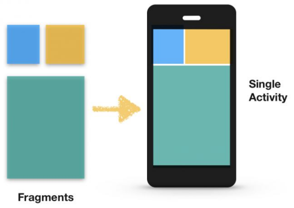
Learning Objectives

- **Fundamental concepts**
  - Understanding HTTP and RESTful Web Services
    - HTTP on Android
  - JSON and JSON parsing
  - What is Retrofit
- **Using Retrofit in Compose projects**
  - Handling permissions (Internet)

Android Development Learning Path

Views and Layouts

✓



Activities and Fragments

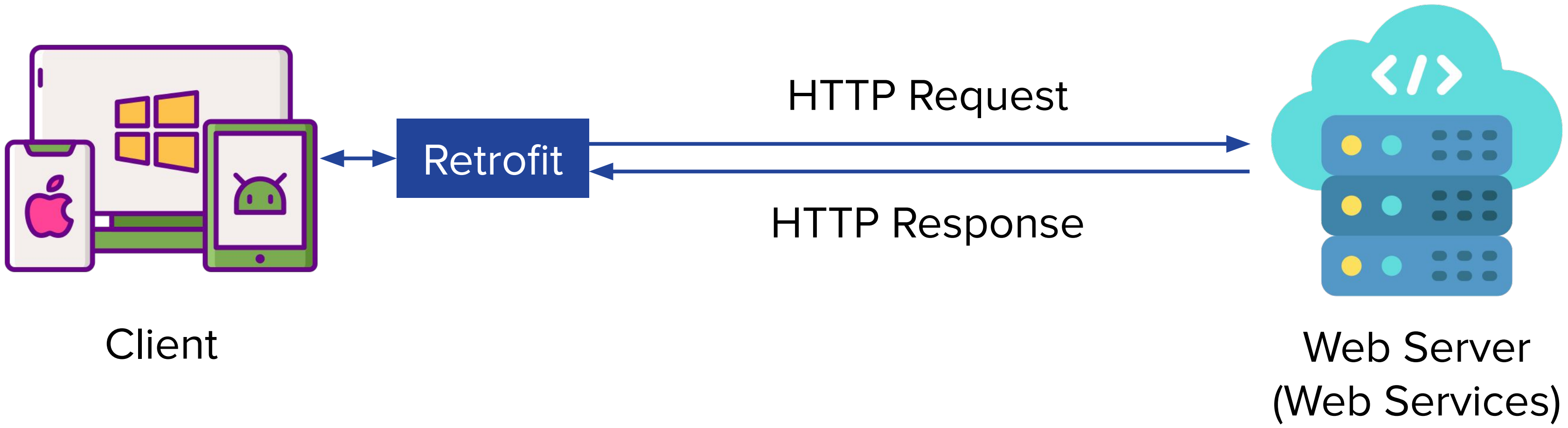
✓

Handle Events

✓

Network Connection

Client, Web Services, HTTP, and Retrofit



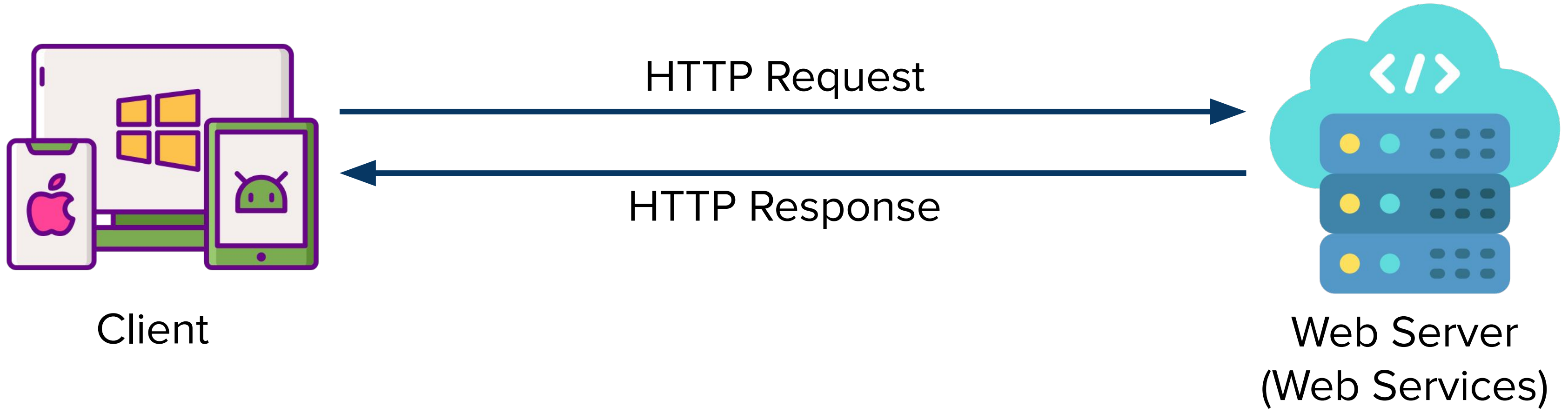
HTTP Fundamentals

Communication between client and server

HTTP Hypertext Transfer Protocol

It is an application-layer protocol for transmitting hypermedia documents, such as HTML.

https://developer.mozilla.org/en-US/docs/Web/HTTP



HTTP Request

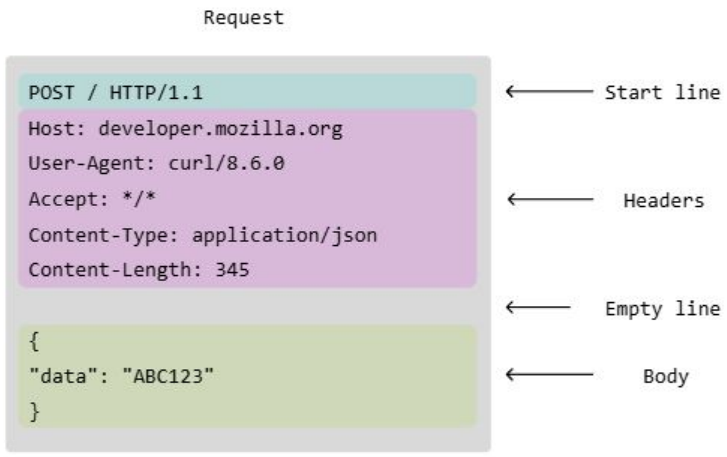
Start Line: HTTP method (HTTP verb), request-target (URL), HTTP version

HTTP headers: description of the message – Metadata

Request body: (optional) send data to server

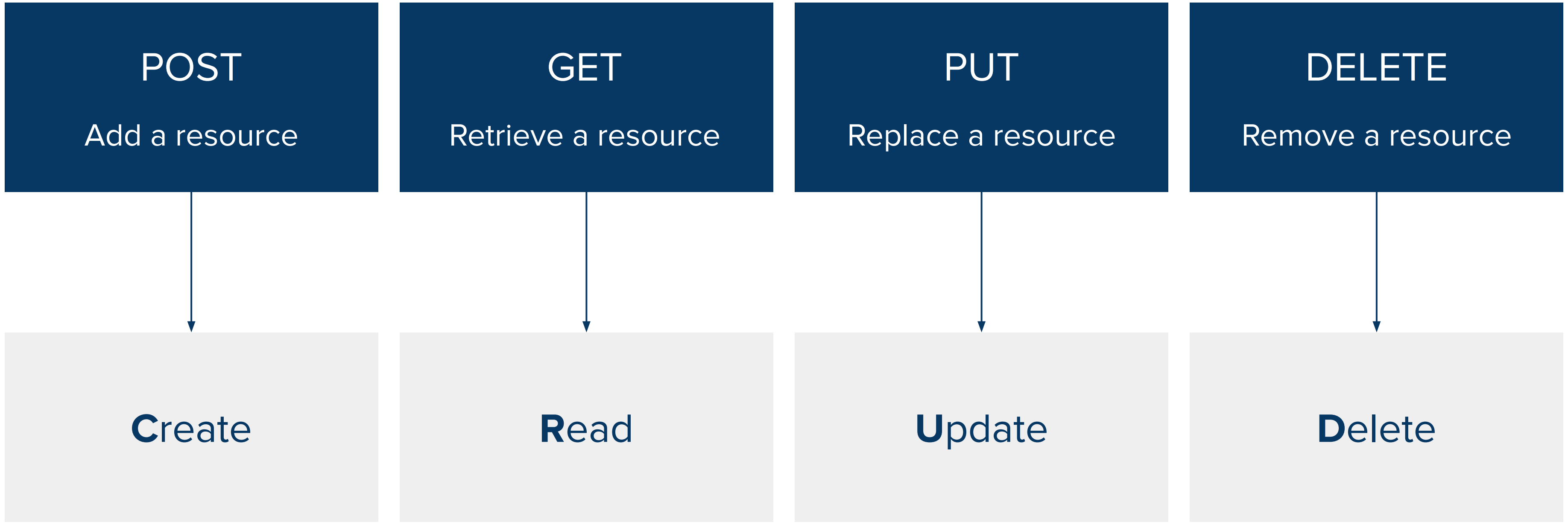
E.g., JSON data for student object

```
{
    “name”: “Joe”,
    “age”: 18
}
```



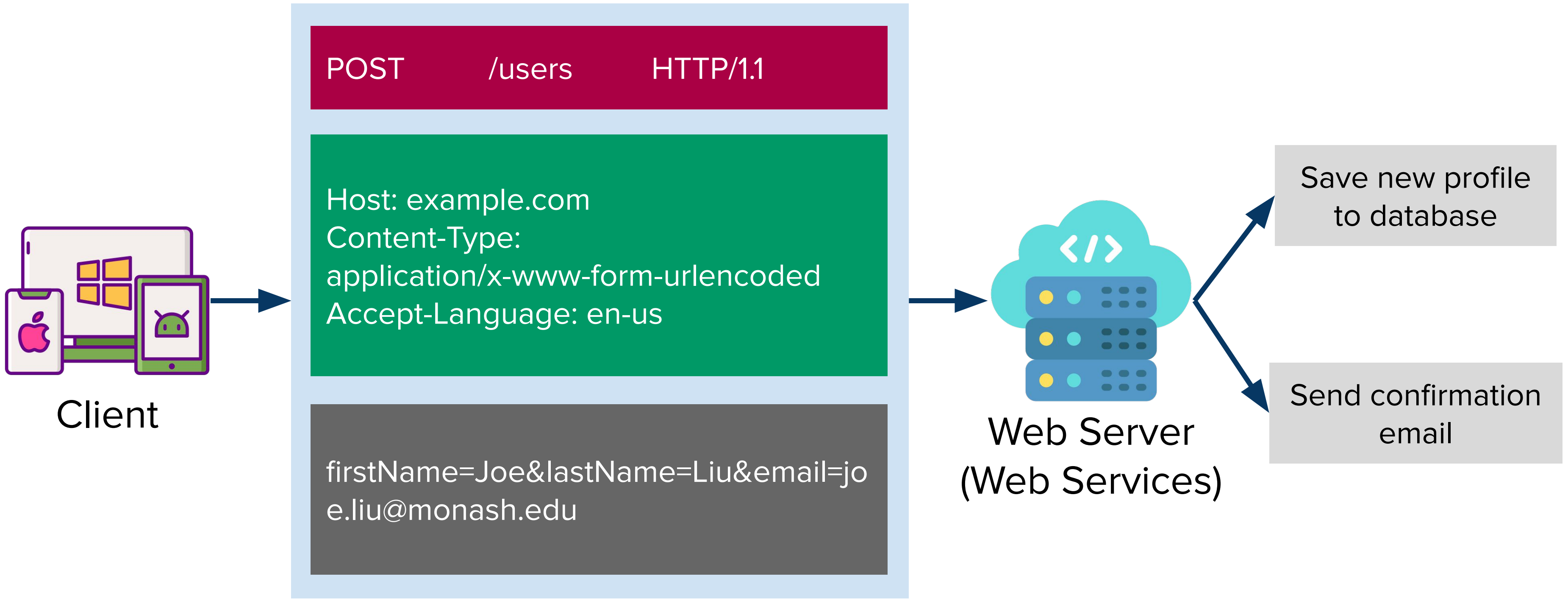
```
POST / HTTP/1.1
Host: developer.mozilla.org
User-Agent: curl/8.6.0
Accept: */*
Content-Type: application/json
Content-Length: 345

{
"data": "ABC123"
}
```

Main HTTP Methods



Four basic operations (actions) of persistent storage

HTTP Request Example

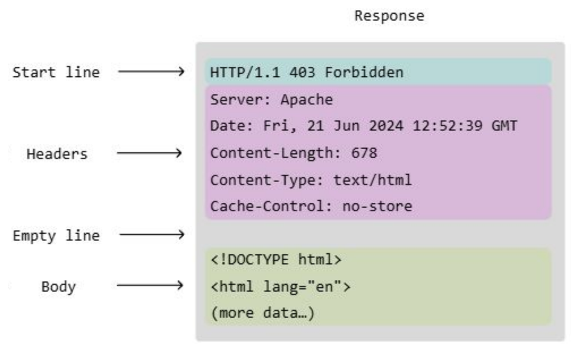


```
POST        /users        HTTP/1.1
```

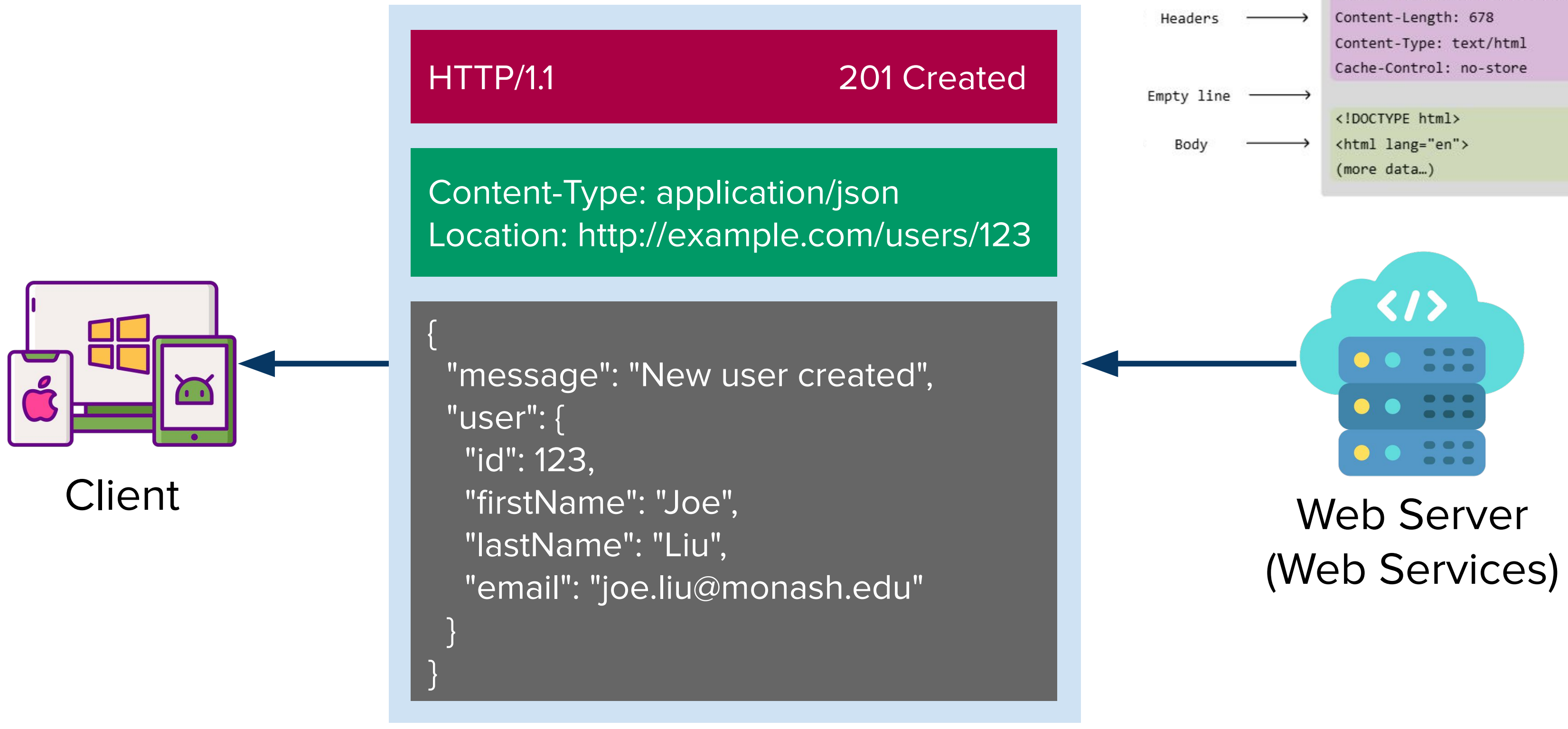
```
Host: example.com
Content-Type:
application/x-www-form-urlencoded
Accept-Language: en-us
```

```
firstName=Joe&lastName=Liu&email=jo
e.liu@monash.edu
```

HTTP Response



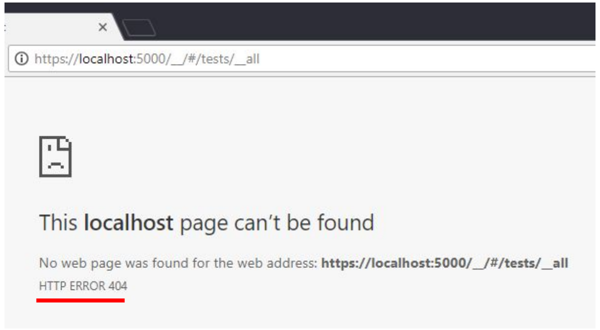
```
HTTP/1.1 403 Forbidden
Server: Apache
Date: Fri, 21 Jun 2024 12:52:39 GMT
Content-Length: 678
Content-Type: text/html
Cache-Control: no-store

<!DOCTYPE html>
<html lang="en">
(more data…)
```



```
HTTP/1.1                201 Created

Content-Type: application/json
Location: http://example.com/users/123

{
  "message": "New user created",
  "user": {
    "id": 123,
    "firstName": "Joe",
    "lastName": "Liu",
    "email": "joe.liu@monash.edu"
  }
}
```

Status Code

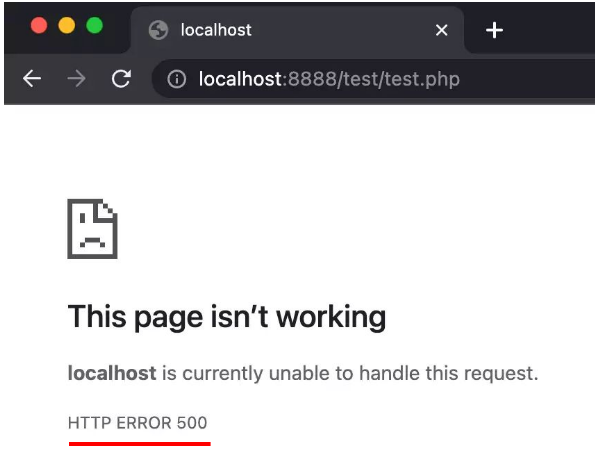
Status Code Range	Meaning
100 – 199	**Informational** responses
200 – 299	**Successful** responses
300 – 399	**Redirection** messages
400 – 499	**Client error** responses
500 – 599	**Server error** responses


https://localhost:5000/__/#/tests/__all

This **localhost** page can't be found

No web page was found for the web address: **https://localhost:5000/__/#/tests/__all**

HTTP ERROR 404


localhost

localhost:8888/test/test.php

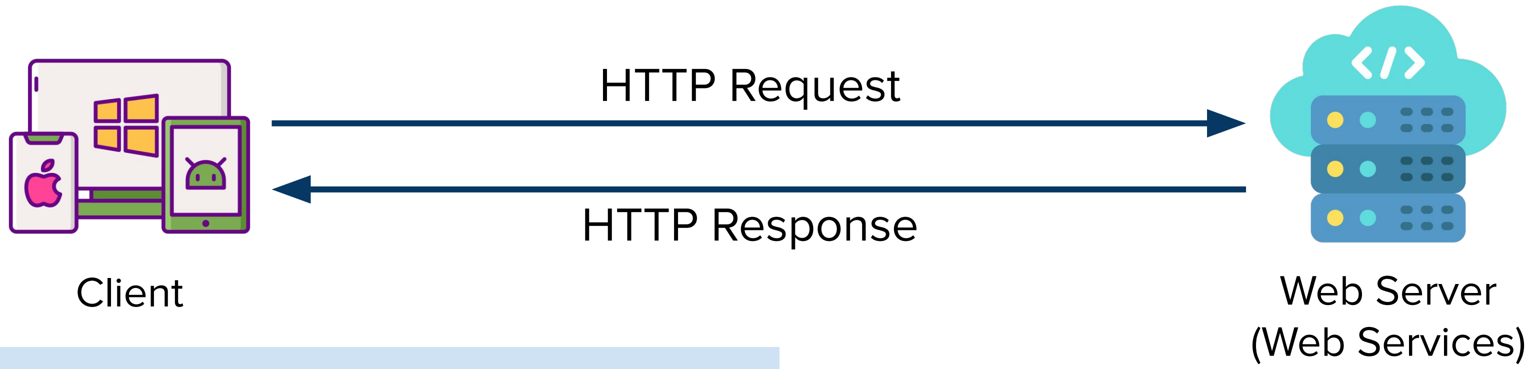
This page isn't working

localhost is currently unable to handle this request.

HTTP ERROR 500

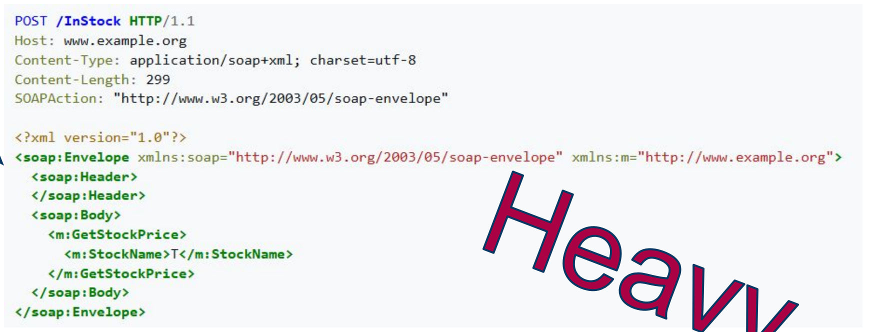
RESTful Web Service

Web Service Patterns and Protocols



SOAP and **WSDL** web services are **XML-based** protocol/language

URL Patterns
example.com/getUser
example.com/addUser
example.com/updateUser
example.com/deleteUser



```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:m="http://www.example.org">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice>
      <m:StockName>T</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Heavy

Example of SOAP message

RESTful Web Service

RESTful web services, or **REST APIs**, are a type of web service that follows the principles of **Representational State Transfer (REST)**

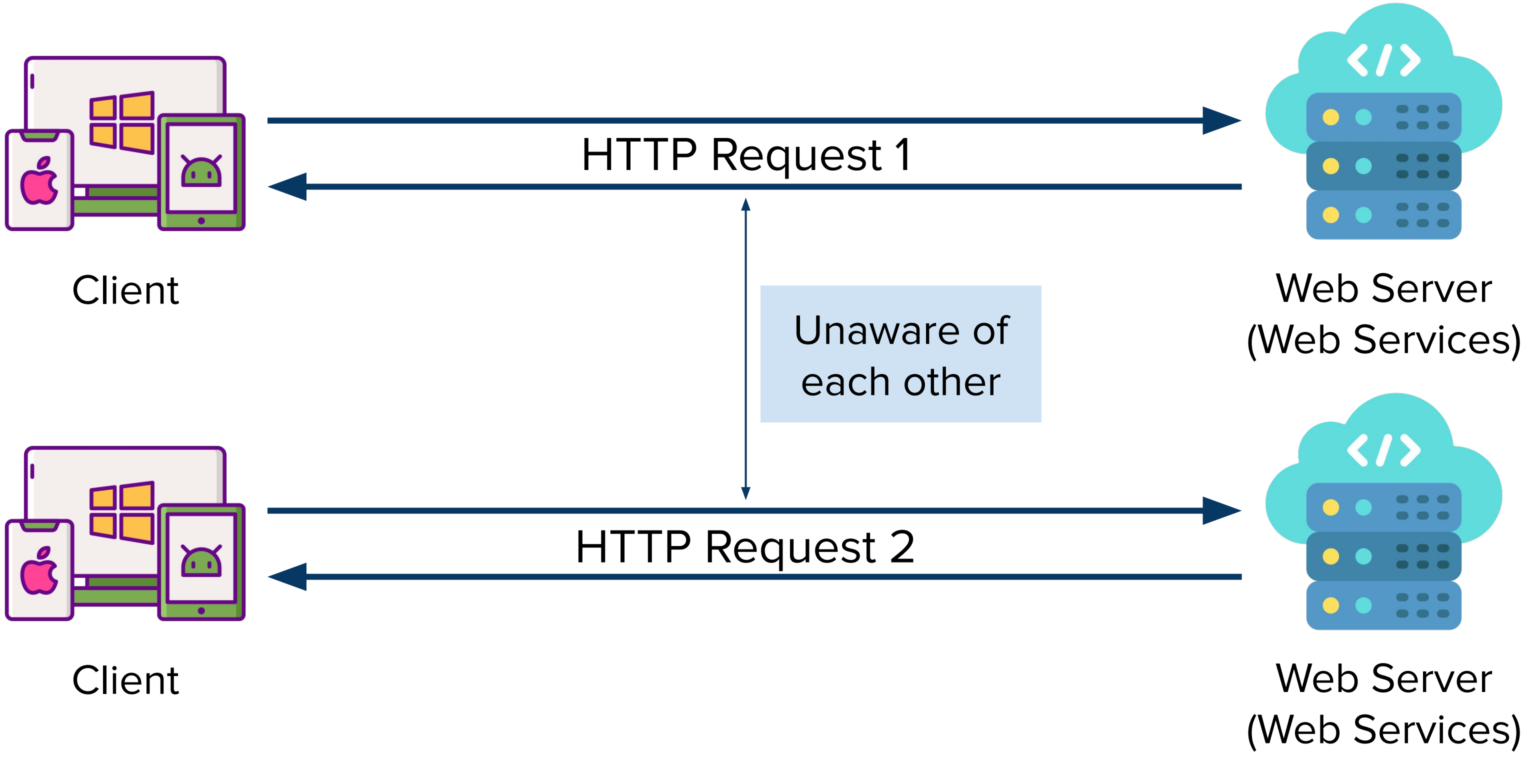
Stateless Client-Server Relationship

Utilise HTTP Methods (POST, GET, PUT, DELETE)

Structured and Consistent URLs

Consistent Data Type Transfer

Stateless Client-Server Relationship



HTTP Methods and Structured URLs

HTTP Method	Consistent URL	Web Service Operation
GET	example.com/users	Fetch User
POST	example.com/users	Add User
PUT	example.com/users	Update User
DELETE	example.com/users	Delete User

HTTP Method	Consistent URL	Web Service Operation
GET	example.com/users/123	Get user with ID of 123
DELETE	example.com/users/123/comments	Delete comments of user whose ID is 123
GET	example.com/users/123/email	Get email of the user whose ID is 123

Consistent Data Type

JSON

XML

A web service is RESTful when it provides **stateless operations** to manage data using different **HTTP methods** and **structured URLs**

JSON and JSON parsing

JSON

- JSON stands for JavaScript Object Notation
- JSON is lightweight text-data interchange format
- JSON is "self-describing" and easy to understand
- JSON supports two structures:
  - Objects: a collection of name/value pairs {"firstName": "John"}
  - Arrays: an ordered list of values

```
{"phoneNumber":
        [
            {
                "type": "home", "number": "212 555-1234"
            },
            {
                "type": "fax", "number": "646 555-4567"
            }
        ]
}
```

https://www.w3schools.com/js/js_json_intro.asp

JSON (cont’d)

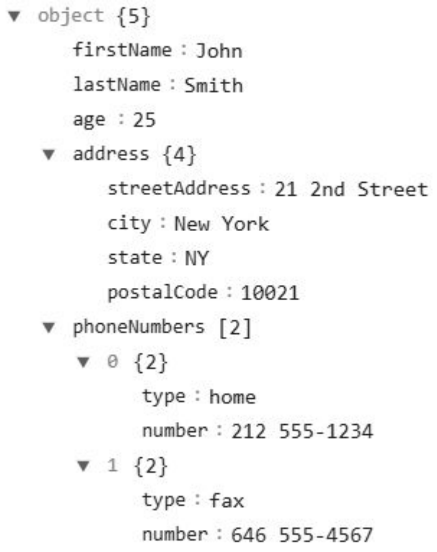
- Objects in name/value pairs (properties) separated by a colon
- A value can be a string, a number, true/false or null, an object or an array
- Data is separated by commas
- Curly braces hold objects and square brackets hold arrays

```
{ "firstName": "John", "lastName": "Smith", "age": 25, "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
   },
   "phoneNumber": [
      {
         "type": "home", "number": "212 555-1234"
      },{
         "type": "fax", "number": "646 555-4567"
      } ]
}
```

Parsing JSON

- JSON parsing online
- https://jsonformatter.org/json-parser
- https://jsoneditoronline.org/

```
{ "firstName": "John", "lastName": "Smith", "age": 25,
"address": { "streetAddress": "21 2nd Street", "city": "New
York", "state": "NY", "postalCode": 10021
},"phoneNumbers": [ {"type": "home", "number": "212
555-1234" }, {"type": "fax", "number": "646 555-4567" } ] }
```



```
▼ object {5}
    firstName : John
    lastName : Smith
    age : 25
  ▼ address {4}
      streetAddress : 21 2nd Street
      city : New York
      state : NY
      postalCode : 10021
  ▼ phoneNumbers [2]
    ▼ 0 {2}
        type : home
        number : 212 555-1234
    ▼ 1 {2}
        type : fax
        number : 646 555-4567
```

Popular HTTP Libraries
Pros & Cons

A Professional App should have ...

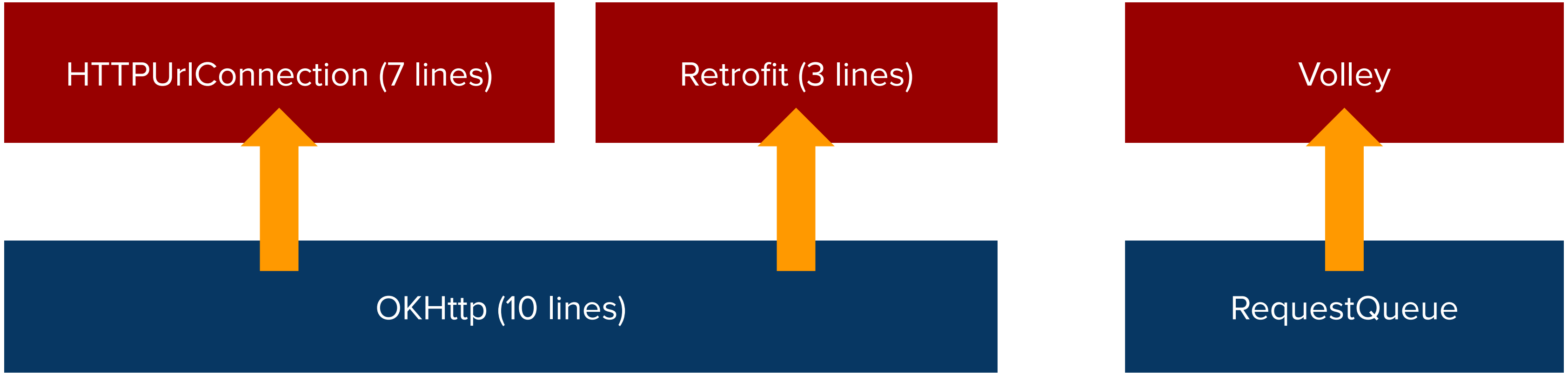
Authentication

Async Requests

Map JSON to usable objects

Load Images

Popular HTTP libraries



They all use Background Threads. Asynchronous in nature.

Disadvantages of HttpURLConnection

- Poor readability and less expressive
- Lots of boilerplate
  - Byte arrays, stream readers
- No built-in support for parsing JSON response
- Manage background threads manually
  - Poor resource management

https://developer.android.com/reference/java/net/HttpURLConnection

Disadvantages of Volley

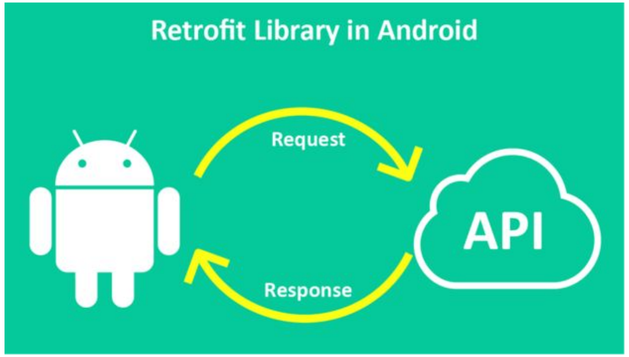
- Limited REST specific features
- Poor authentication layer
- Meagre documentation
- Smaller community

https://github.com/google/volley

Introduction to Retrofit

What is Retrofit

- Retrofit developed by Square https://square.github.io/retrofit/
- Retrofit facilitates **interactions with public APIs** in Android (http calls)
- Retrofit is built on top of OkHttp
- Retrofit supports adding and using converters such as **Gson libraries** to convert Java objects into their JSON representation or vice versa



Why use Retrofit

- Active Community
  - Easier troubleshooting
- Expressive code with more abstraction
- Manage resources efficiently
  - Background threads
  - Async calls and queries
  - Automatic JSON parsing using GSON library
  - Automatic error handling callbacks
  - Built-in user authentication support

Internet access permission & Retrofit Dependencies

- To use the internet, the manifest file must include the internet permission
  - <uses-permission android:name="android.permission.INTERNET" />

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     xmlns:tools="http://schemas.android.com/tools">

     <uses-permission android:name="android.permission.INTERNET" />

     <application

…
```

- Using Retrofit, we need to add the required dependencies to the module level gradle file

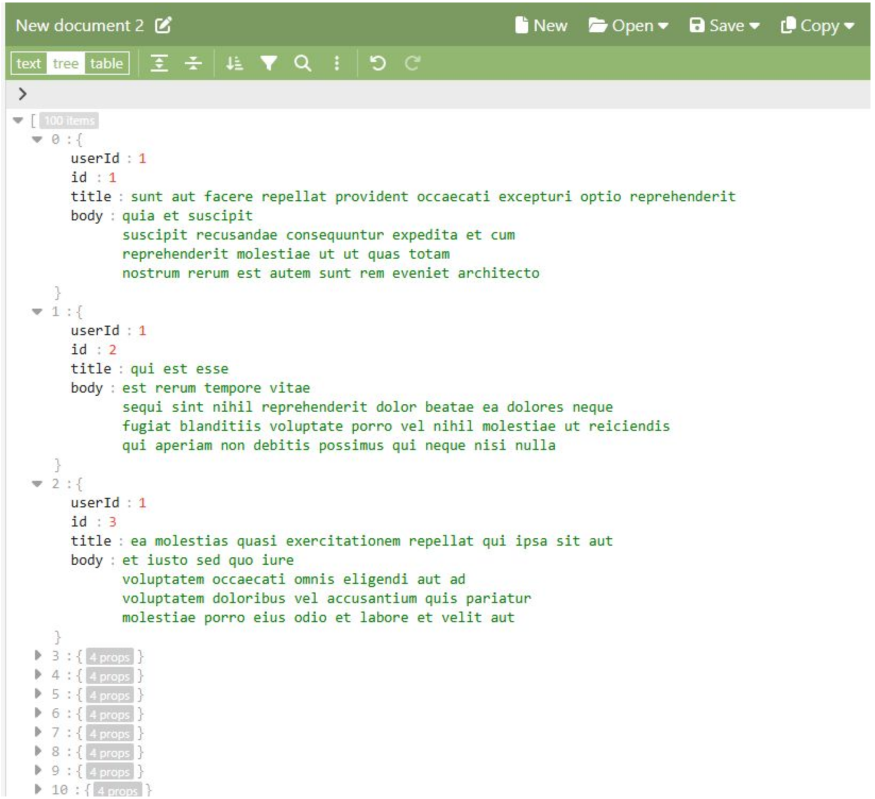
```
implementation("com.squareup.retrofit2:retrofit:2.11.0")
implementation("com.squareup.retrofit2:converter-gson:2.11.0") // Gson Converter
```

Retrofit Model Class

- To provide a mapping from the structure of the JSON’s response body to Kotlin objects, we use **data classes** (other options are also possible)
- Using data classes, we can obtain the objects we want from the long body of the JSON response
- *E.g. in the Posts response (JSON) we want to access key-value pairs under the ‘Posts’ so we create a Retrofit Model class to map them*

```kotlin
data class Post(
   val userId: Int,
   val id: Int,
   val title: String,
   val body: String
)
```

Retrofit Model Class (cont'd)



```
[ 100 items
  0 : {
    userId : 1
    id : 1
    title : sunt aut facere repellat provident occaecati excepturi optio reprehenderit
    body : quia et suscipit
           suscipit recusandae consequuntur expedita et cum
           reprehenderit molestiae ut ut quas totam
           nostrum rerum est autem sunt rem eveniet architecto
  }
  1 : {
    userId : 1
    id : 2
    title : qui est esse
    body : est rerum tempore vitae
           sequi sint nihil reprehenderit dolor beatae ea dolores neque
           fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis
           qui aperiam non debitis possimus qui neque nisi nulla
  }
  2 : {
    userId : 1
    id : 3
    title : ea molestias quasi exercitationem repellat qui ipsa sit aut
    body : et iusto sed quo iure
           voluptatem occaecati omnis eligendi aut ad
           voluptatem doloribus vel accusantium quis pariatur
           molestiae porro eius odio et labore et velit aut
  }
  3 : { 4 props }
  4 : { 4 props }
  5 : { 4 props }
  6 : { 4 props }
  7 : { 4 props }
  8 : { 4 props }
  9 : { 4 props }
  10 : { 4 props }
```

```kotlin
data class Post(
    val userId: Int,
    val id: Int,
    val title: String,
    val body: String
)
```

Retrofit Interface

- Retrofit interface handles the HTTP API
- An interface defines http methods (**GET**) and the HTTP API (@Path or @Query)

  [suspend fun will be covered next week]
  - @Query is used to define query parameters for HTTP requests
  - @Path is used to define path parameters that are included in a URL path
- The Model class should **match the returned type** in the Retrofit Interface

```
interface MyAPI {
    @GET("posts")
    fun getPosts(): Call<List<Post>>
}
```

https://square.github.io/retrofit/

```
interface MyAPI {
    @GET("Posts/{id}")
    suspend fun getPostsByID(
        @Path("id") id: Int
    ): Call<List<Post>>
}
```

Understand Query and Path in HTTP Requests

Retrofit Builder

- **Retrofit.Builder** is used to create **an instance of Retrofit** by calling the **build()**
- The build() uses the **baseUrl** and the **Gson converter factory** provided to
- create the Retrofit instance
- We then call the **create()** on the Retrofit instance and pass the Retrofit interface

```
val BASE_URL = "https://jsonplaceholder.typicode.com/"

val api = Retrofit.Builder()
     .baseUrl(BASE_URL)
     .addConverterFactory(GsonConverterFactory.create())
     .build()
     .create(MyAPI::class.java)
```

Retrofit Response

- The **enqueue()** function asynchronously send the **“GET”** request defined in our API via **getPosts()** and notify Callback of its responses. We need to override the default behaviours for functions **onResponse()** and **onFailure().**

```
val TAG: String = "CHECK_RESPONSE"

api.getPosts().enqueue(object : Callback<List<Post>> {
   override fun onResponse(p0: Call<List<Post>>, p1: Response<List<Post>>) {
       if(p1.isSuccessful){
           p1.body()?.let {
               for (post in it){
                   Log.i(TAG,"onResponse: ${post.body}")
               }
           }
       }
   }

   override fun onFailure(p0: Call<List<Post>>, p1: Throwable) {
       Log.i(TAG, "onFailure: ${p1.message}")
   }
})
```

Reminders and Announcements

Assignments:

- **App Critiques (10%) - Deadline (Thursday 11:55 PM)**
- **Peer Engagement Weekly Task (2%)**

Lab activities this week

- Develop three apps that connect to WebAPIs
  - Use Coil3 Library and Retrofit Library
  - Generate and use Google Gemini API Key

Week 8

- *Introduction to coroutines and async tasks in Kotlin*
- *Structured concurrency basics*

Reference

- Pari Delir Haghighi (S1 2024) Network Connection and Retrofit [PowerPoint slides], FIT5046: Mobile and Distributed Computing Systems, Monash University.
- Flaticon: https://www.flaticon.com/
- https://medium.com/ibtech/activity-vs-fragment-703c749c1bbd
- ChatGPT Image Generation - one week off