# Android Dev Kotlin

CS 402: Mobile Development
Michael Ziray - michaelziray@boisestate.edu

# JSON

JavaScript Object Notation

## JSON

Stands for JavaScript Object Notation

Is essentially just a JavaScript object

Evaluates to JavaScript

## Why JSON?

Web services should be agnostic to the clients they're serving (browser, mobile, watch)

XML is clunky and hard to parse

JSON can be consumed by JavaScript web apps for free

Easily parsed by Android or iOS natively

## JSON Example

```
{
  "glossary": {
     "title": "example glossary",
     "GlossDiv": {
     "title": "S",
     "GlossList": {
          "GlossEntry": {
               "ID": "SGML",
               "SortAs": "SGML",
               "GlossTerm": "Standard Generalized Markup Language",
           }
      }
   }
}
```

# JSON Serializer Example (Java)

```
public class JSONSerializer{
  private Context mContext; private String mFilename;

  public JSONSerializer( Context c, String f) {
    mContext = c;    mFilename = f;
  }

  // Build an array in JSON public void saveCrimes( ArrayList < Crime > crimes) throws
JSONException, IOException {
    JSONArray array = new JSONArray();
    for (Crime c : crimes){
        array.put( c.toJSON()); // Write the file to disk
    }
    Writer writer = null;
    try {
        OutputStream out = mContext.openFileOutput( mFilename, Context.MODE_PRIVATE);
        writer = new OutputStreamWriter( out);
        writer.write( array.toString());
    }    finally {        if (writer != null) writer.close();    } }}
```

## toJSON Implementation

```
public JSONObject toJSON() throws
JSONException{
 JSONObject json = new JSONObject();
 json.put( JSON_ID, mId.toString());
 json.put( JSON_TITLE, mTitle);
 json.put( JSON_SOLVED, mSolved);
 json.put( JSON_DATE, mDate.getTime());
 return json;
}
```

## Reading JSON

```
public Crime( JSONObject json) throws JSONException
{
 mId = UUID.fromString( json.getString( JSON_ID));
 mTitle = json.getString( JSON_TITLE);
 mSolved = json.getBoolean( JSON_SOLVED);
 mDate = new Date( json.getLong( JSON_DATE));
}
```

## Parsing JSON

```
public ArrayList < Crime > loadCrimes() throws IOException, JSONException {
  ArrayList < Crime > crimes = new ArrayList < Crime >();
  BufferedReader reader = null;
  try {
     // Open and read the file into a StringBuilder
     InputStream in = mContext.openFileInput( mFilename);
     reader = new BufferedReader( new InputStreamReader( in));
     StringBuilder jsonString = new StringBuilder();
     String line = null;

     // Line breaks are omitted and irrelevant
     while (( line = reader.readLine()) != null) {
         jsonString.append( line);
     } // Parse the JSON using JSON
     // Build the array of crimes from JSONObjects
     Tokener JSONArray array = (JSONArray) new JSONTokener( jsonString.toString()).nextValue();
     for (int i = 0; i < array.length(); i + +) {
         crimes.add( new Crime( array.getJSONObject( i)));
     }
  }
  catch (FileNotFoundException e) { // Ignore this one; it happens when starting fresh }
  finally {
    if (reader != null) reader.close();
  }
  return crimes;
}
```

# ReST

Transferring data across a network

## Terminology

**Request** - Any time you send information to the server

**Response** - The return information from the server, in response to the original request.

## ReST

REpresentational State Transfer

Request methods are:

PUT, POST, GET

DELETE, PATCH, HEAD, OPTIONS

## GET

Gets a resource from a URI

Most webpage requests use GET

Web browsers: GET -> http://www.google.com

## PUT

Puts or replaces a new resource onto the server at a ***specified*** location

Often for updating a record, but not always.

Updating a user’s name:

i.e. /api/user/123/?name=new%20name

## POST

Posts information to a server

Sends information to the server at no particular URI

Creating a new user:

/api/user => {user: “Mike”, pass: “pass”}

## Query Params vs Body

Query Params:

/api/user/123/?name=new%20name // Params need to be encoded

Request Body

/api/user => {user: “Mike”, pass: “pass”} // JSON, XML, etc

## DELETE

Removes a resource on the server

A lot of APIs will use GET /user/123/delete or something similar

## Testing JSON

http://www.jsontest.com/

## HTTP Basic Auth

Sets the username and password in the header of each request. Must be over SSL (HTTPS).

## HTTP Basic Auth

```
@ExperimentalStdlibApi



val credentials = "username:password"

val byteData:String = Base64.encodeToString(credentials.encodeToByteArray(),
Base64.URL_SAFE or Base64.NO_WRAP)
```

## HTTP Basic Auth

Converts the string username:password into base64

Header looks like:

```
Authorization: Basic aDHR0cHdhdGNoOmY=
```

## Tokenized Authentication

Instead of sending a Username/Password for every transaction, you send an authorization token that you receive when first logging in.

Send that token with each request instead.

## Tokenized Authentication

The server can invalidate that token at any time for security reasons or session expiration.

Ex: auth=”zyxw987cba321” // expires at 5:00 AM Sept. 29, 2019

Ex: auth=”999zyxw111cba” // expires at 5:00 AM Oct 12, 2019

## A Better HTTP Library

Google Volley

Retrofit

okHTTP