Andorid Programming

Week 7

Mina Jung

EECS, Syracuse University

Spring 2017

Part I

RESTful API

Outline I

- REST (REpresentational State Transer) architecture
  - useful to build client/server network applications
  - simple stateless architecture
  - performs requests and receive responses (interactions via HTTP protocol)
  - basically works on HTTP protocol
- 5 REST pinciples
  - uniform interface
    - Resource-Based
    - Manipulation of Resources Through Representations
    - Self-descriptive Messages
    - Hypermedia as the Engine of Application State (HATEOAS)
  - stateless communication
  - cacheable

- client-server architecture
- layered system to increase scalability

- should support most commonly used HTTP methods
  - **GET**: To fetch a resource
  - **POST**: To create a new resource
  - **PUT**: To update existing resource
  - **DELETE**: To delete a resource

- HTTP Status Code in the response

200	OK
201	Created
304	Not Modified
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
422	Unprocessable Entity
500	Internal Server Error

- URL Structure
  - Every URL for a resource should be uniquely identified
    - GET http://www.example.com/tasks/11 – will give the details of a task whose id is 11
    - POST http://www.example.com/tasks – will create a new task
  - If your API needs an API key to access, the api key should be kept in HTTP headers instead of including it in URL
- Content Type
  - Content Type in HTTP headers specifies the kind of the data should be transferred between server and client
  - Content-Type: application/json
  - Content-Type: application/xml

- PHP 5 Tutorial – https://www.w3schools.com/php/

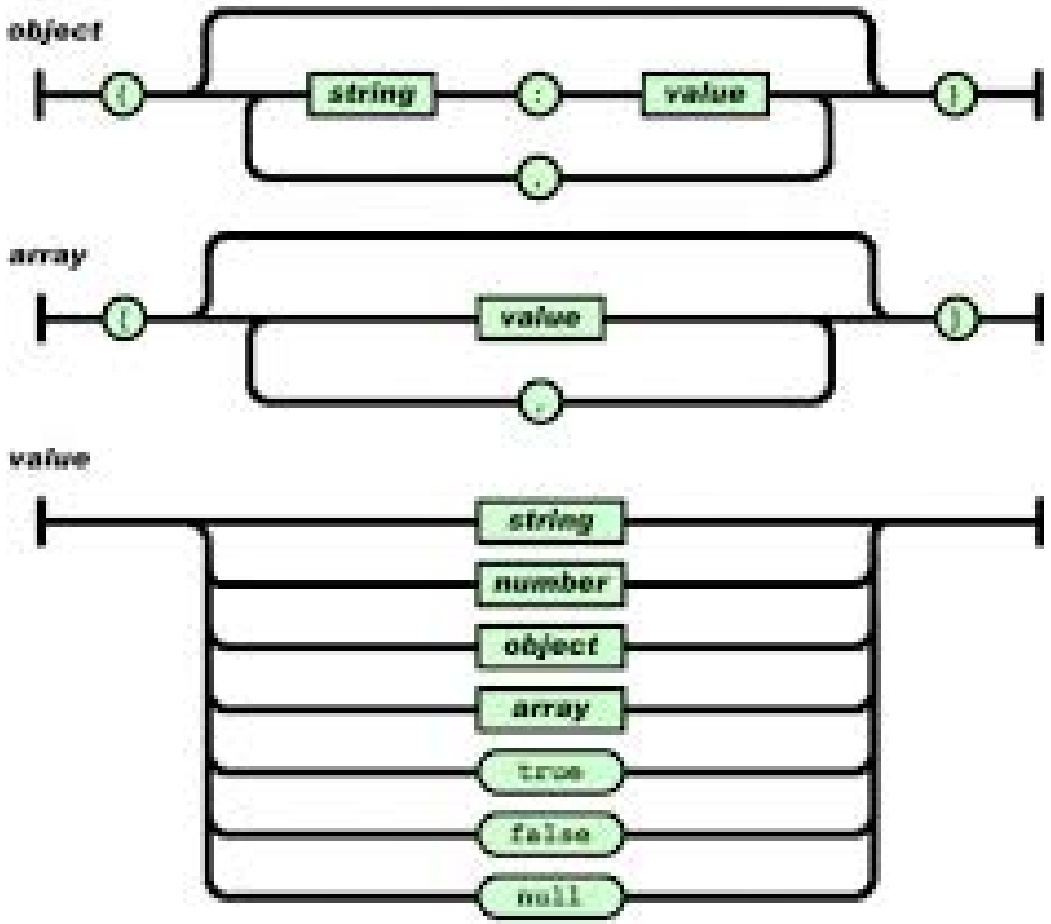
- PHP MySQL Database – https://www.w3schools.com/php/php_mysql_intro.asp

- very light weight, structured, easy to parse and much human readable
- best alternative to XML when your android app needs to interchange data with your server
- Sample JSON

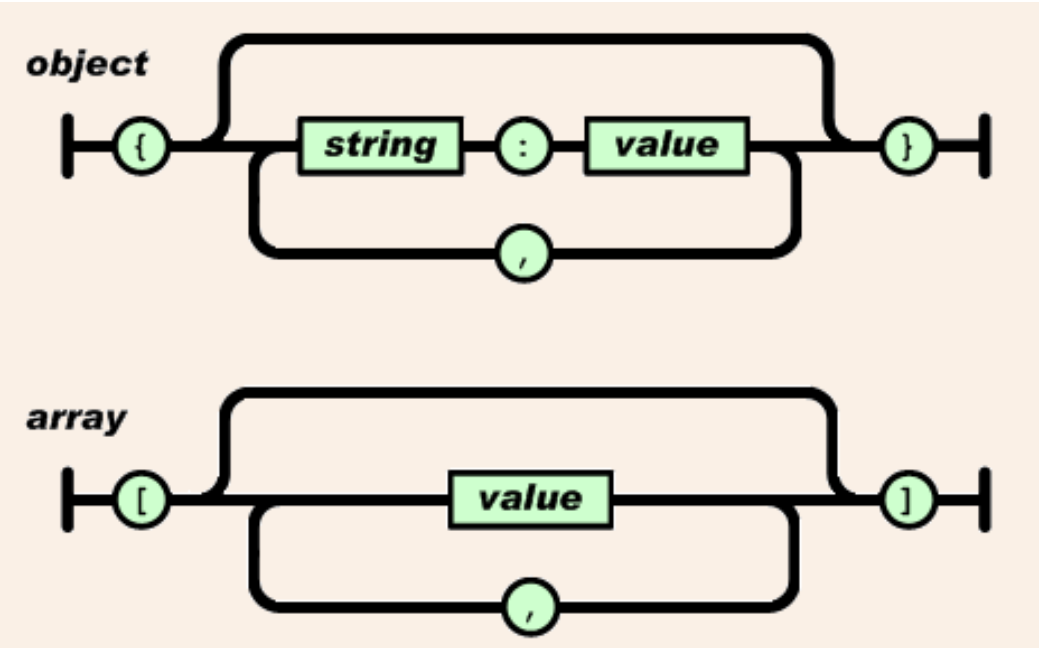
```
{
    "contacts": [
        {
                "id": "c200",
                "name": "John Doe",
                "email": "john@gmail.com",
                "address": "xx-xx-xxxx,x - street, x - country",
                "gender" : "male",
                "phone": {
                    "mobile": "+91 0000000000",
                    "home": "00 000000",
                    "office": "00 000000"
                }
```

```
        },
        {
                "id": "c201",
                "name": "Jane Doe",
                "email": "jane@gmail.com",
                "address": "xx-xx-xxxx,x - street, x - country",
                "gender" : "female",
                "phone": {
                    "mobile": "+91 0000000000",
                    "home": "00 000000",
                    "office": "00 000000"
                }
        },
        .
        .
        .
        .
  ]
}
```

- JSON nodes will start with a square bracket or with a curly bracket



- difference between [and {
  - square bracket ([) represents starting of an **JSONArray** node – getJSONArray() method
  - curly bracket ({) represents **JSONObject** – getJSONObject() method



- Instead of developing a new RESTful API from scratch
- Slim – https://www.slimframework.com/
  - very light-weight, clean and easy
  - support all HTTP methods for REST API
- Download & Install

step 1. Create a new folder (myslim) in your htdocs directory

step 2. Download Composer(https://getcomposer.org/download/) into myslim

step 3. Install Slim with Composer

```
php composer.phar require slim/slim "^3.0"
```

- Test Slim in your host (www.example.com or your own name)

step 1. Create index.php in the directory of your host (For example, ∼/htdocs/Dummy/)

```
<?php
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require '../myslim/vendor/autoload.php';

$app = new \Slim\App;
$app->get('/hello/{name}', function (Request $request,
     Response $response) {
    $name = $request->getAttribute('name');
    $response->getBody()->write("Hello, $name");

    return $response;
});
$app->run();
?>
```

step 2. Create **.htacess** file in the same directory as index.php. **.htacess** rules can get rid of index.php from the url and make some friendly urls

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ %{ENV:BASE}index.php [QSA,L] // or
     RewriteRule ^ index.php [QSA,L]
```

step 3. If your .htaccess file doesn't work, please check your Apache configuration (httpd-vhosts.conf)

change `AllowOverride None` to **AllowOverride All**

Part II

Connecting to MySQL Database using PHP – Actual Server-Side Programming

Outline I

Outline II

Handling API calls in index.php using Slim and PHP

- GET : Get All Movies
- GET : Get Single Movie
- Example GET & POST routes

- add xamppfiles/mysql/bin
- add xamppfiles/php

Step 1. Open phpmyadmin by opening the address http://localhost/phpmyadmin/ in your browser

- either phpMyAdmin tool or MySQL command line tool to create a database and a table
- We will use phpMyAdmin

Step 2. Download androidMovieDB.sql from the blackboard

Step 3. Create a new database (androidMovieDB)

Step 4. Load data into "androidMovieDB" database

- import androidMovieDB.sql into the database

Step 5. Run SQL query on database "androidMovieDB"

- PHP 5 and later can work with a MySQL database using:
  - MySQLi extension (the "i" stands for improved)
    - support object-oriented and procedural
    - only work with MySQL databases
  - PDO (PHP Data Objects)
    - object-oriented
    - work on 12 different database systems

- Connect to MySQL database using **MySQLi**

```
function getDB() {
$dbhost = "localhost";
$dbuser = "root";
$dbpass = "";
$dbname = "androidMovieDB";

// Create a DB connection
$conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
if ($conn->connect_error) {
die("Connection failed: " . $conn->connect_error . "\n");
}

return $conn;
}
```

mysqli_connect()	Opens a new connection to the MySQL server
mysqli_close()	Closes a previously opened database connection
mysqli_query()	Performs a query against the database
mysqli_num_rows()	Returns the number of rows in a result set
mysqli_fetch_all()	Fetches all result rows as an associative array, a numeric array, or both
mysqli_fetch_array()	Fetches a result row as an associative, a numeric array, or both
mysqli_fetch_assoc()	Fetches a result row as an associative array
mysqli_prepare()	Prepares an SQL statement for execution

- db_config.php: define database connection variables

```
<?php
/**
 * Database configuration
 */
define('DB_USERNAME', 'root');
define('DB_PASSWORD', '');
define('DB_HOST', 'localhost');
define('DB_NAME', 'androidMovieDB');
?>
```

- db_connect.php: a class file to connect to database

```php
<?php

/**
 * Handling database connection
 */
class DbConnect {

    private $conn;

    function __construct() {
    }

    /**
     * Establishing database connection
     * @return database connection handler
     */
    function connect() {
        include_once dirname(__FILE__) . '/db_config.php';

        // Connecting to mysql database
        $this->conn = new mysqli(DB_HOST, DB_USERNAME, DB_PASSWORD,
            DB_NAME);

        // Check for database connection error
        if (mysqli_connect_errno()) {
```

```
            echo "Failed to connect to MySQL: " .
                mysqli_connect_error();
        }

        // returing connection resource
        return $this->conn;
    }

}

?>
```

- db_config.php: define database connection variables

```
<?php

/*
 * All database connection variables
 */

define('DB_USER', "root"); // db user
define('DB_PASSWORD', ""); // db password  if you have
define('DB_DATABASE', "androidMovieDB"); // database name
define('DB_SERVER', "localhost"); // db server
?>
```

- db_connect.php: a class file to connect/disconnect to database

```php
<?php

/**
 * A class file to connect to database
 */
class DB_CONNECT {

    // constructor
    function __construct() {
        // connecting to database
        $this->connect();
    }

    // destructor
    function __destruct() {
        // closing db connection
        $this->close();
    }

    /**
     * Function to connect with database
     */
    function connect() {
        // import database connection variables
        require_once __DIR__ . '/db_config.php';
```

```
        // Connecting to mysql database
        $con = mysql_connect(DB_SERVER, DB_USER, DB_PASSWORD) or
            die(mysql_error());

        // Selecing database
        $db = mysql_select_db(DB_DATABASE) or die(mysql_error()) or
            die(mysql_error());

        // returing connection cursor
        return $con;
    }

    /**
     * Function to close db connection
     */
    function close() {
        // closing db connection
        mysql_close();
    }

}

?>
```

- very useful against SQL injections

- Prepared Statements and Bound Parameters
  1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?").
  2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
  3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

```
$conn = getDB();

// prepare and bind
$stmt =
      $conn->$prepare("INSERT INTO Members (first, last, email) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $first, $last, $email);

// set parameters and execute
$first = "John";
$last = "Doe";
$email = "john@example.com";
$stmt->execute();

$first = "Jane";
$last = "Doe";
$email = "jane@example.com";
$stmt->execute();

$stmt->close();
$conn->close();
```

- Question mark (?) where we want to substitute in an integer, string, double or blob value
- bind_param()'s first argument (list of parameter data type, what type of data to expect) may be one of four types:
  - i - integer
  - d - double
  - s - string
  - b - BLOB
- Repeated execution
  - A prepared statement can be executed repeatedly
  - Upon every execution the current value of the bound variable is evaluated and sent to the server
  - The statement is not parsed again
  - The statement template is not transferred to the server again

- Advantages compared to executing SQL statements directly
  - reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
  - minimize bandwidth to the server as need to send only the parameters each time, and not the whole query
  - very useful against SQL injections. If the original statement template is not derived from external input, SQL injection cannot occur.

```php
<?php

/**
 * Class to handle all db operations
 */
class DbHandler {

    private $conn;

    function __construct() {
        require_once dirname(__FILE__) . '/db_connect.php';
        // opening db connection
        $db = new DbConnect();
        $this->conn = $db->connect();
    }

    public function Close() {
        $this->conn->close();
    }

    public function getAllMovies() {
     ...
    }

    public function getMovieById($mid) {
     ...
```

```
    }

    public function getMoviesByRating($rating) {
      ...
    }

    public function addNewMovie(...) {
      ...
    }

    public function deleteMovieById($mid) {
      ...
    }

    ...
}
?>
```

```
/**
 * Fetching all movie
 */
public function getAllMovies() {
    $stmt = $this->conn->prepare("SELECT * FROM Movies WHERE
        1");
    $stmt->execute();
    $movies = $stmt->get_result();
    $stmt->close();
    return $movies;
}
```

```
/**
 * Fetching movie by id
 * @param String $mid -- movie id
 */
public function getMovieById($mid) {
    $stmt = $this->conn->prepare("SELECT * FROM Movies WHERE id
        = ?");
    $stmt->bind_param("s", $mid);
    if ($stmt->execute()) {
        $movie = $stmt->get_result()->fetch_assoc();
        $stmt->close();
        return $movie;
    } else {
        return NULL;
    }
}
```

```
// Write Your Own Code
```

```
// Write Your Own Code
```

```
// Write Your Own Code
```

```
<?php

require_once './db_handler.php';
require_once '../myslim/vendor/autoload.php';


$app = new \Slim\App;



// Run the Slim Application
$app->run();
?>
```

```
/**
 * Get all Movies
 */
$app->get(
  '/movies/',
  function() {
    $response = array();
    $db = new DbHandler();

    $result = $db->getAllMovies();

    if($result != NULL) {
      $response['error'] = false;
      $response['movies'] = array();

      while( $movie = $result->fetch_assoc() ) {
        $tmp = array();
        $tmp['id'] = $movie['id'];
        $tmp['name'] = $movie['name'];
        $tmp['description'] = $movie['description'];
        $tmp['rating'] = "".$movie['rating'];
        $tmp['url'] = $movie['url'];

        array_push($response['movies'], $tmp);
      }
```

```
        echo json_encode($response);
    }
    else{
        $response['error'] = true;
        $response['message'] = "There is no movie in database";

        echo json_encode($response);
    }

    $db->Close();
  }
);
```

```
/**
 * Get Single Movie
 */
$app->get(
  '/movies/id/{mid}',
  function($request, $response, $args) {
    $response = array();
    $db = new DbHandler();

    $result = $db->getMovieById($args['mid']);

    if( $result != NULL ) {
      $response['error'] = false;

      $response['id'] = $result['id'];
      $response['name'] = $result['name'];
      $response['description'] = $result['description'];
      $response['rating'] = "".$result['rating'];
      $response['url'] = $result['url'];

      echo json_encode($response);
    }
    else {

      $response['error'] = true;
```

```
        $response['message'] = "There is no requested movie in
            database";

        echo json_encode($response);
    }

    $db->Close();
  }
);
```

```
/**
 * Get (no arguments/patterns/segments)
 */
$app->get(
  '/',
  function() {
    echo '<h1>Android Programming</h1><br><br>';
    echo '<form action="/post" method="post">';
    echo '<label>First Name: </label>';
    echo '<input name="first" type="text"/> <br>';
    echo '<label>Last Name: </label>';
    echo '<input name="last" type="text"/> <br>';
    echo '<input type="submit" value="Submit"/>';
    echo '</form>';
  }
);

/**
 * POST
 */
$app->post(
  '/post',
  function($request, $response, $args) {
    echo "This is a POST route. <br>";
    $data = $request->getParsedBody();
```

```
    echo $data['first']."<br>";
    echo $data['last'];
  }
);
```