

Objective

Introduction to the Web API
and REST API architecture

Content

- Introduction to Web API
- REST basics
- REST API user authentication
- Implementation and using REST API

Web API Introduction

API (Application Programming Interface)

- “A set of **functions and procedures** allowing the creation of applications that access the features or data of an operating system, application, or other service”

(Oxford English Dictionary)

- “A **set of programming code** that queries data, parses responses, and sends instructions between one software platform and another”

(Investopedia)

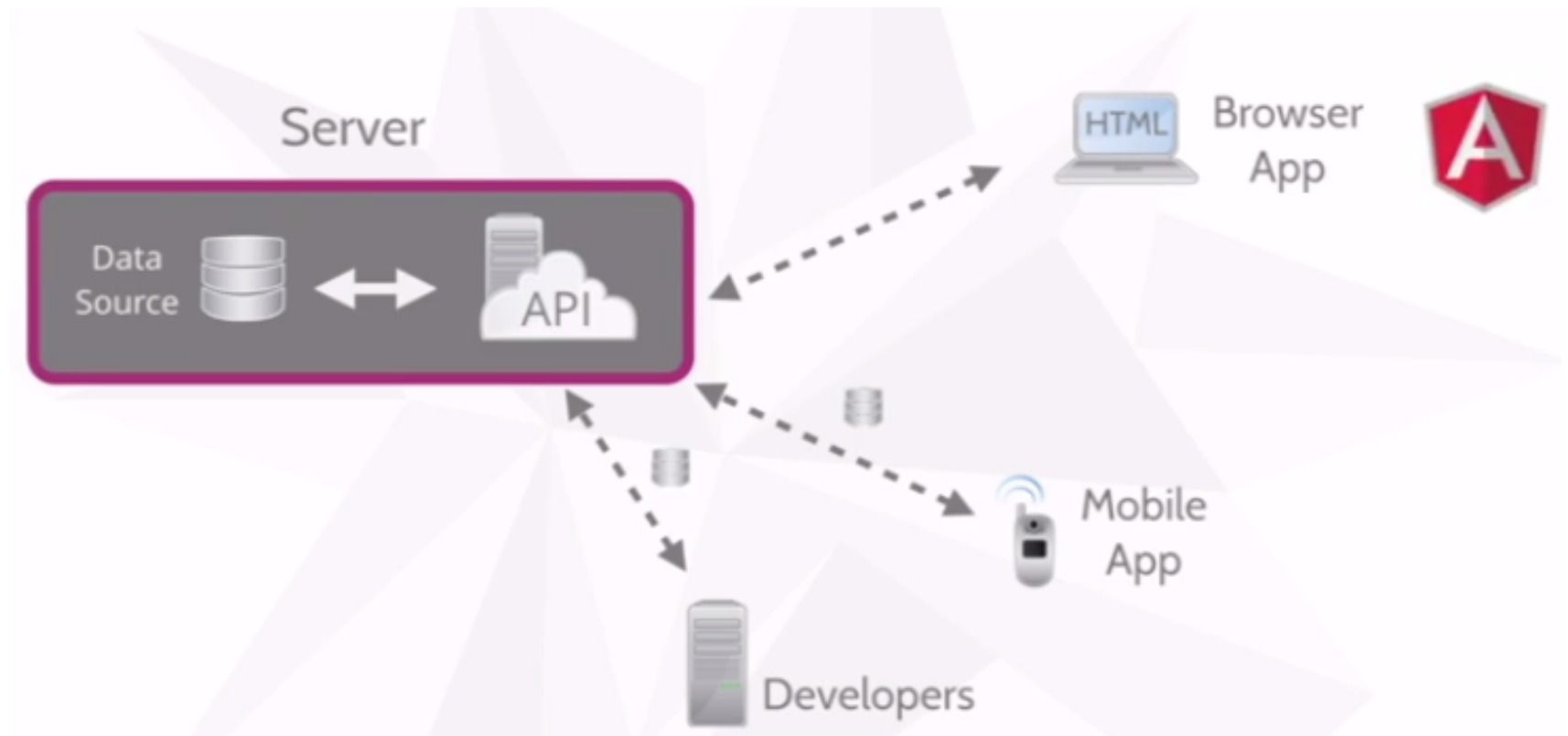
Web API

- APIs that work based on the **WWW infrastructure**:
 - Request/Response is established using HTTP protocol
 - They can be implemented by various technologies: Java, .NET, PHP,...
- Suited to the machine – machine communications
(*vs. the websites: human – machine communication*)
- Request/Response are **plain text** with JSON or XML syntax that are easy for parsing and analyzing
(*vs. HTML that is used for formatting data – presentation*)
 - **Input:** URI [+JSON/XML/...], **Output:** JSON/XML/...

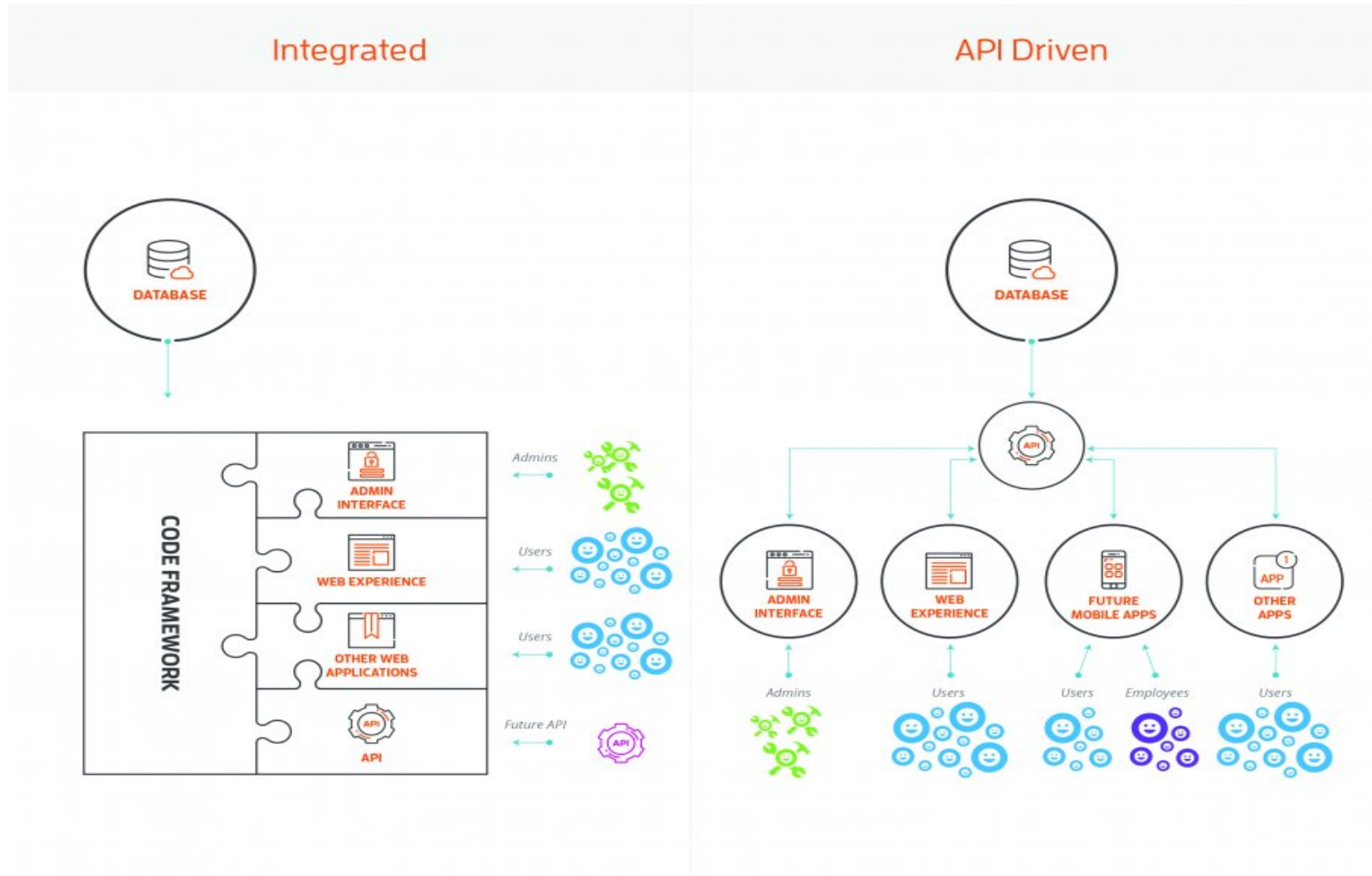
Web API

- Demo Facebook REST API

API-Driven web applications



API-Driven vs. Integrated web applications



REST Basics

What is REST?

- REST = Representational State Transfer
 - A software architecture style that defines a set of constraints for creating web services
- REST constraints:
 - Client/Server, stateless, cache, uniform interface (between components), layered system, code-on-demand (optional)
- Basic concepts: resource, representation state and manipulation of resources

Resource

- An object with a type, associated data, relationships to other resources and a set of methods that operate on it (any information: document, image,...)
- Identified by an **URI** (Uniform Resource Identifier)
 - Several URIs may refer to the same resource
- Type of resources:
 - collection resource:
 - Example: /lectures
 - instance resource:
 - Example: /lectures/001533

Representation

- The state of a resource at any particular timestamp
 - consists of data, metadata describing the data and hypermedia links which can help the clients in transition to next desired state
 - is the information exchanged between client and server
- Typically in JSON or XML syntax
- Example:
 - Resource: person
 - Representation: {
"name": "Perter Jackson",
"address": "123 Ada Street, PN",
"phone": "(+64) 123456" }

Resource methods

- Used to perform the desired transitions
 - **GET** (idempotent): read
 - **DELETE** (idempotent): delete
 - **POST**: create (or update)
 - **PUT** (idempotent): update (or create)
- All above methods should be implemented for every resource to reduce the error (use response state as an indication if the method is not supported)

Create a new resource with PUT

- Provide ID and information of new resource
 - PUT <new resource URI>
{
 . . . *//properties of the new resource*
}
- Example, to create a new lecturer:
 - PUT /lectures/clientSpecifiedID
{
 //properties of the new lecturer
}

Update a resource with PUT

- Updating data and status of a resource:
 - `PUT <existing resource URI>`
`{`
`. . . //new properties of the resource`
`}`
- Example, update lecturer's information
 - `PUT /lecturers/existingID`
`{`
`"name": "Bùi Võ Quốc Bảo",`
`"dept": "IT"`
`}`

Create a new resource with POST

- Create a new resource:
 - `POST <new resource ID>`
`{`
`. . . //properties of the new resource`
`}`
- Example, create a new lecturer (auto ID):
 - `POST /lecturers`
`{`
`"name": "Bùi Võ Quốc Bảo"`
`}`

Update a resource with POST

- Update an existing resource:
 - `POST <existing resource ID>`
`{`
`. . . //properties of the existing resource`
`}`
- Example, update information of a lecturer:
 - `POST /lecturers/003025`
`{`
`"name": "Nguyen Van A"`
`}`

REST API – Best practices

- Use **nouns** to represent **resources** (may be singular or plural)
- A resource should **provide all methods** GET, PUT, POST, DELETE (unsupported methods are indicated by resp. status)
- Status information must be hold at the **client side** (e.g. if the client need to perform several steps, it must hold the current step)
- Clients are not required to know all URI of all resources but the API will provide links to other related resouces instead (**HATEOAS**: Hypermedia As The Engine Of Application State)

HATEOAS

- HATEOAS: Hypermedia as the Engine of Application State

Request: GET /api/v1/cars/711

Response:

```
{  
  "id": 711,  
  "manufacturer": "bmw",  
  "model": "X5",  
  "seats": 5,  
  "drivers": [{  
    "id": "23",  
    "name": "Stefan Jauker",  
    "links": [{"rel": "self",  
               "href": "/api/v1/drivers/23"}] } ]  
}
```

REST API Authentication

Token-based authentication

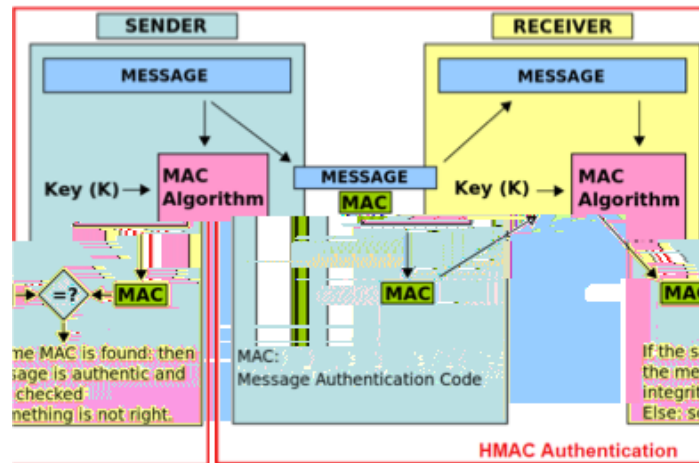
- Server generates a token for the user (based on credentials)
 - Token: một chuỗi được sinh ra ngẫu nhiên
- Every request sent to server will include the token in the HTTP header, e.g.:
 - `curl -X GET https://127.0.0.1/api/example`
`-H 'X-API-Key: 9944b09199c62bcf9418ad846dd0e4bety89eff'`
- Server will validate the token to check whether the token is valid or not

HMAC-based authentication

- HMAC: Hash-based message authentication code
- Use a pair of hashes: private hash and public hash
 - Public hash: used to identify the user, can be public
 - Private hash: kept secret (only the user and server know)
- Client:
 - Private hash is combined with the request content to create content hash
 - Content hash + public hash + request are sent to the server

HMAC-based authentication

- Server:
 - uses public hash to identify the user and the corresponding private hash
 - hashes the request content + private hash
 - compare the computed hash with the hash value in the request to perform the authentication



HMAC ba ed a e ca

- P a e a a u r
 - S d c r a g e e a g e
 - a c ue

- E a e
 - a a a c a d e d b e

a a a a u

HMAC-based authentication – Client

array

array

true

echo

\n

\n

true

\n\n

HMAC-based authentication – Server

```
<?php
require_once 'vendor/autoload.php';
$app = new \Slim\App();
$app->post('/', function($request, $response) {
    $publicHash = $request->getHeaderLine('X-Public');
    $contentHash = $request->getHeaderLine('X-Hash');
    $privateHash = 'e249c439ed7697df2a4b045d97d4b9b7e1854c3ff8dd668c779...';

    $content = $request->getBody();
    $hash = hash_hmac('sha256', $content, $privateHash);

    if ($hash == $contentHash){
        echo "match!\n";
    }
});
?>
```

Creating and Consuming REST API

C a REST API

- S a a a C T
 - a C T a a
- PHP a a → J a
 - a a
 - a *// {"a":1,"b":2,"c":3,"d":4,"e":5}*
- J → PHP a a a
 - a
 - *// \$obj is a stdClass object*
 - a *// \$arr is an associative array*

Jsend-based response format

- Success (2xx): yêu cầu được xử lý thành công

```
{
  "status" : "success",
  "data" : {
    "post" : { "id" : 1,
               "title" : "A blog post",
               "body" : "Some useful content"
            }
  }
}

{
  "status" : "success",
  "data" : null
}
```

Json-based response format

- Thất bại (4xx): yêu cầu không hợp lệ (lỗi client)

```
{  
  "status" : "failed",  
  "data" : {"title" : "A title is required"}  
}
```

- Thất bại (5xx): lỗi trong quá trình xử lý y/c (lỗi server)

```
{  
  "status" : "error",  
  "message" : "Unable to communicate with database"  
}
```

JSON in Javascript

- JS object → Json string: **JSON.stringify(obj)**
 - `var j = {"name" : "binchen"};`
 - `JSON.stringify(j); // '{"name":"binchen"}'`
- Json string → JS object: **JSON.parse(json)**
 - `var json = '{"result":true,"count":1}';`
 - `var obj = JSON.parse(json);`

Consume JSON data with AJAX (jQuery)

- Receive JSON data from server:

```
$.ajax({  
  type: 'GET',  
  url: '/api/contacts',  
  data: {name: 'Bao'},  
  dataType: 'json',  
  success: function (res) {  
    // data is converted from JSON to JS object  
    // automatically by jQuery  
    // res: JS object  
  }  
});
```

Consume JSON data with AJAX (jQuery)

- Send JSON data to server:

```
var contact = {name: 'Bao', phone: '1234567890'};
$.ajax({
  type: 'POST',
  url: '/api/contacts',
  data: JSON.stringify(contact),
  contentType: 'application/json',
  success: function (res) {
    alert(res)
  }
});
```



Question?

CT313H – WEB TECHNOLOGIES