Universidad Nacional de Quilmes

API REST

Construcción de Interfaces de Usuario

1er Cuatrimestre 2019

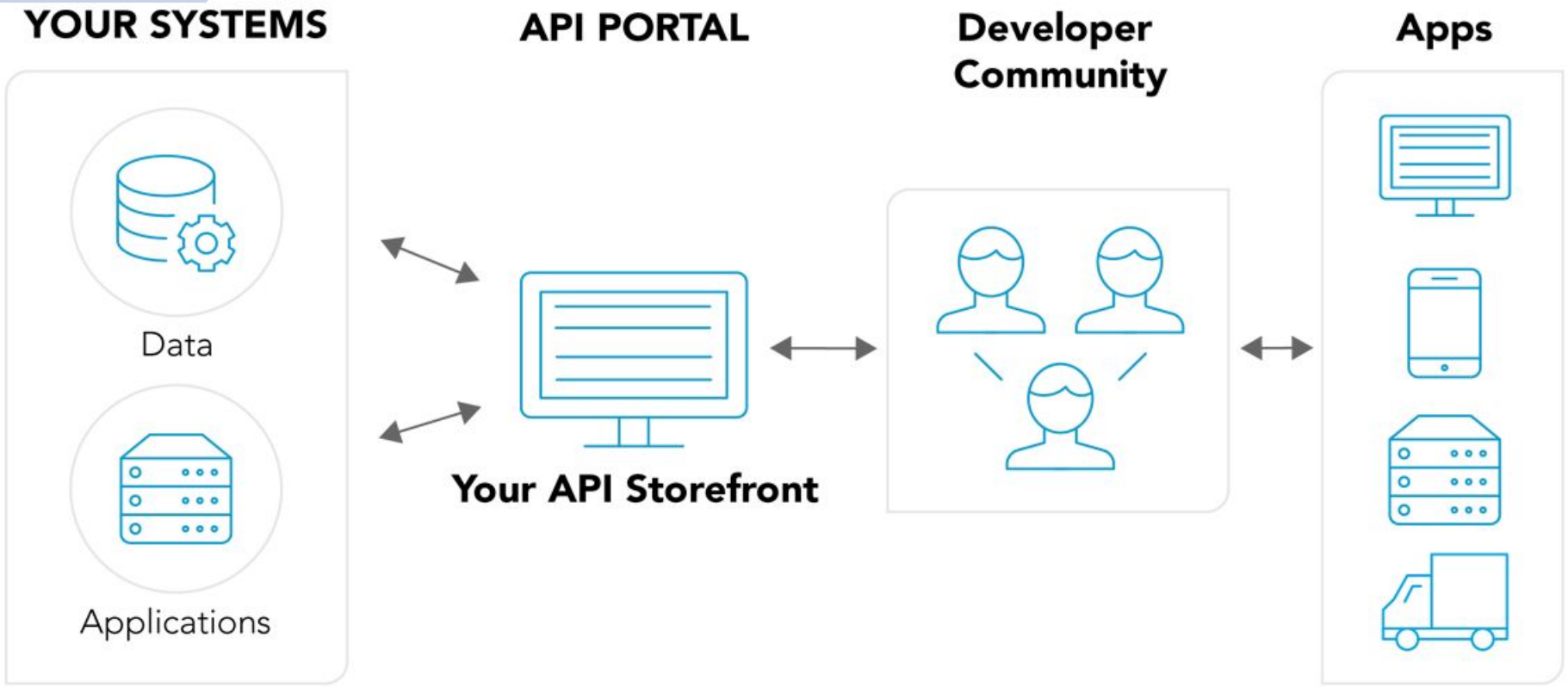
API

- Interfaz de Programación de Aplicaciones
  - *Application Programming Interface*
- Conjunto de comandos, protocolos, funciones, objetos, etc...
- Provee estándares para facilitar la interacción con componentes
- Encapsula tareas complejas en otras más simples

API

- ¿Es una Interfaz de Usuario?
- ¿Quién es el Usuario?
- ¿Conocen ejemplos de APIs?

Esquema



Get your APIs to market on a portal

APIs Web

- Implementan servicios
- Exponen recursos
- Permite interactuar con multiplicidad de tecnologías
- Establecen un protocolo de comunicación
- Agregan una capa de seguridad

REST

- Transferencia de Estado Representacional
  - *REpresentational State Transfer*
- *Estilo* de arquitectura de software que provee estándares para la interacción entre sistemas web
- Hace que la comunicación entre sistemas sea más simple

APIs RESTful

- Aquellas aplicaciones que son compatibles con los principios REST:
  - Stateless
  - Arquitectura Cliente-Servidor
  - Uso de Caché
  - Interface Uniforme

API REST

Interface Uniforme

- Identificación del Recurso
  - `usuarios/`, `restaurants/`, `pedidos/`, etc...
- Operaciones bien definidas
  - `GET`, `POST`, `PUT`, `DELETE`, etc...
- Sintaxis Universal
  - `GET usuarios/`, `DELETE usuarios/`, etc...
- Hypermedia
  - `application/json`, `text/html`, etc...

API REST

Formato de Intercambio

Se necesita un formato definido para intercambiar información. Los dos formatos más extendidos son:

JSON

JavaScript Object Notation

```
{
  "credentials": {
    "username": "hodor",
    "password": "hodor"
  }
}
```

XML

Extensible Markup Language

```
<credentials>
    <username>hodor</username>
    <password>hodor</password>
</credentials>
```

JSON

Reglas de Sintaxis

La sintaxis de JSON deriva de la sintaxis de notación de objetos de JavaScript:

- Información como par **“key”:“value”**
- Datos separados por coma (**,**)
- Las llaves (**{}**) contienen objetos
- Los corchetes (**[]**) contienen listas

JSON

Tipos de datos

Las **keys** son *“strings”*.

Los **value** pueden ser:

- String `"algún texto"`
- Number `1 | -1 | 1.2 | -1.0`
- Object `{"key": "value"}`
- List `[1, "dos", {"val": 3.0}]`
- Boolean `true | false`
- Null `null`

JSON

```
{
  "lugar":    "Universidad Nacional de Quilmes",
  "coordenadas": {
    "latitud": -34.706294,
    "longitud": -58.278522
  },
  "distancias": [
    {
      "lugar": "Obelisco",
      "kms": 14.81,
    }, {
      "lugar": "Mendoza",
      "kms": 996.52
    },
  ]
}
```

API REST

REQUESTs

Para cada **REQUEST**, en una API REST se define la estructura a la cual el cliente se debe ajustar para recuperar o modificar un recurso. En general consiste de:

- **Verbo HTTP:** define qué tipo de operación realizar
- **Protocolo aceptado:** HTTP 1.1, HTTP 1.0
- **Media Data** aceptada: html, json, xml
- **Encabezado:** (opcional) permite pasar información extra
- **Ruta** al recurso
- **Cuerpo de mensaje** (opcional) que contiene datos

API REST

REQUESTs » Ejemplos

```
GET /users/23 HTTP/1.1
Accept: text/html, application/json
```

```
POST /users HTTP/1.1
Accept: application/json
Body: {“user”: {
    “name”: “Arya Stark”
    “email”: “nobody@braavos.org”
}}
```

API REST

RESPONSEs

Por cada **REQUEST** que se recibe se debe retornar un **RESPONSE** con la información necesaria para describir lo que ocurrió:

- **HTTP Code** acorde a lo sucedido con la ejecución
- **Protocolo** de respuesta
- **Media-data** de la respuesta
- **Cuerpo de mensaje** (opcional) con la información requerida

API REST

RESPONSEs » Ejemplo (I)

```
GET /users/42 HTTP/1.1
Accept: text/html, application/json
```

```
HTTP/1.1 200 (OK)
Content-Type: application/json
Body: {“user”: {
    “name”:“Hodor”
    “email”: “hodor@winterfell.com”
}}
```

API REST

RESPONSEs » Ejemplo (II)

```
POST /users HTTP/1.1
Body: {“user”: {
    “name”: “Arya Stark”
    “email”: “nobody@braavos.org”
}}
```

```
201 (CREATED)
Content-type: application/json
```

API REST

CRUD

El modelo debe poder **crear**, **leer**, **actualizar** y **eliminar** recursos (**C**reate, **R**ead, **U**pdate, **D**elete). A esto se le llama CRUD. Es la funcionalidad mínima que se espera de un modelo.

El paradigma CRUD es muy común en la construcción de aplicaciones web porque proporciona un modelo mental sobre los recursos, de manera que sean completos y utilizables.

API REST

CRUD ≫ Estándares ≫ Definición

Los CRUD se suelen arman respetando un estándar de URIs y métodos:

- `Crear` `POST` `/users`
- `Leer (todos)` `GET` `/users`
- `Leer (uno)` `GET` `/users/:id`
- `Actualizar` `PUT` `/users/:id`
- `Eliminar` `DELETE` `/users/:id`

API REST

CRUD ≫ Estándares ≫ Respuesta

- **POST /users**
  - 201 (Created)
  - {“user”: Nuevo Usuario}
- **GET /users**
  - 200 (OK)
  - {“users”:[Listado]}
- **GET /users/:id**
  - 200 (OK)
  - {“user”: Usuario Pedido}
- **PUT /users/:id**
  - 200 (OK)
  - {“user”: Usuario Actualiz.}
- **DELETE /users/:id**
  - 204 (No Content)
  - Body: Vacío

API REST

CRUD ≫ Estándares ≫ Errores

- **POST /users**
  - 404 (Not Found)
  - 409 (Conflict)
- **GET /users/:id**
  - 404 (Not Found)
- **PUT /users/:id**
  - 404 (Not Found)
  - 409 (Conflict)
- **DELETE /users/:id**
  - 404 (Not Found)
  - 405 (Method Not Allowed)

API REST

Errores Genéricos

- 401 (Unauthorized)
- 403 (Forbidden)
- 405 (Method Not Allowed)
- 500 (Internal Server Error)

API REST

Parámetros de consulta

Muchas veces es necesario agregar información a la solicitud. Puede ser para filtrar una búsqueda o bien para que la respuesta incluya más o menos información.

Para estos casos se suelen utilizar parámetros de consulta (*query parameters*). Se escriben como un par **`clave=valor`** separados por **`&`**.

API REST

Parámetros de consulta Ejemplos

- `GET /users?mail=gmail&born_in=1990`
- `GET /users/123?include=orders`
- `GET /users?page=3&per_page=25`

No es buena práctica incluir parámetros en otros métodos que no sean de consulta (`GET`). Para enviar información (`POST`, `PUT`) se debe usar el *body*.


javalin
A simple web framework
for Java and Kotlin
Java
Kotlin
Copy
import io.javalin.Javalin;
public class HelloWorld {
public static void main(String[] args) {
Javalin app = Javalin.create().start(7000);
app.get("/", ctx -> ctx.result("Hello World"));
}
}

LINKS ÚTILES

- https://json.org/json-es.html
- https://www.restapitutorial.com/
- https://jsonapi.org/
- https://www.codecademy.com/articles/what-is-rest
- https://www.codecademy.com/articles/what-is-crud
- https://javalin.io/
- https://github.com/toddmotto/public-apis

¿Preguntas?