Architetture per il web

Ingegneria del software

Vincenzo Bonnici
Corso di Laurea in Informatica
Dipartimento di Scienze Matematiche, Fisiche e Informatiche
Università degli Studi di Parma

2025-2026

RPC – Remote Procedure Call

- I servizi RPC chiamano letteralmente le procedure (cioè le funzioni) in remoto.
- Questo tipi di API hanno in genere un singolo endpoint, quindi tutte le richieste vengono effettuate allo stesso URL.
- Ogni richiesta includerà il nome della funzione da chiamare e può includere alcuni parametri da passare ad essa
- Esempio:
  - http://api.flickr.com/services/rest/?method=flickr.photos.search&tags=kitten&format=xmlrpc
  - All'interno dell'URL, il nome della funzione può essere visto nel parametro "method" (flickr.photos.search), i tag particolari da cercare si trovano in tags= e il parametro format richiede la risposta in formato XML-RPC
-
- Esiste una netta differenza tra l'utilizzo di un servizio di tipo RPC, con nomi di funzione e parametri inclusi nei dati forniti, e l'utilizzo di un servizio che è vero XML-RPC, che è un formato molto definito

RPC – Remote Procedure Call

- La creazione di un livello di servizio RPC per un'applicazione può essere ottenuta in modo molto semplice eseguendo il wrapping di una classe ed esponendola tramite HTTP.

```php
<?php

class Events
{
    protected $events = array(
        1 => array("name" => "Excellent PHP Event",
            "date" => 1454994000,
            "location" => "Amsterdam"
        ),
        2 => array("name" => "Marvellous PHP Conference",
            "date" => 1454112000,
            "location" => "Toronto"),
        3 => array("name" => "Fantastic Community Meetup",
            "date" => 1454894800,
            "location" => "Johannesburg"
        )
    );

    /**
     * Get all the events we know about
     *
     * @return array The collection of events
     */
    public function getEvents() {
        return $this->events;
    }

    /**
     * Fetch the detail for a single event
     *
     * @param int $event_id The identifier of the event
     *
     * @return array The event data
     */
    public function getEventById($event_id) {
        if(isset($this->events[$event_id])) {
            return $this->events[$event_id];
        } else {
            throw new Exception("Event not found");
        }
    }
}
```

RPC – Remote Procedure Call

- La creazione di un livello di servizio RPC per un'applicazione può essere ottenuta in modo molto semplice eseguendo il wrapping di una classe ed esponendola tramite HTTP.

- Per renderlo disponibile tramite un servizio di tipo RPC, è possibile scrivere un semplice wrapper che esamina i parametri in ingresso e chiama la funzione pertinente.

```php
<?php

require "Events.php";

// look for a valid action
if(isset($_GET['method'])) {
    switch($_GET['method']) {
        case "eventList":
            $events = new Events();
            $data = $events->getEvents();
            break;
        case "event":
            $event_id = (int)$_GET['event_id'];
            $events = new Events();
            $data = $events->getEventById($event_id);
            break;
        default:
            http_response_code(400);
            $data = array("error" => "bad request");
            break;
    }
} else {
    http_response_code(400);
    $data = array("error" => "bad request");
}

// let's send the data back
header("Content-Type: application/json");
echo json_encode($data);
```

SOAP - Simple Object Access Protocol

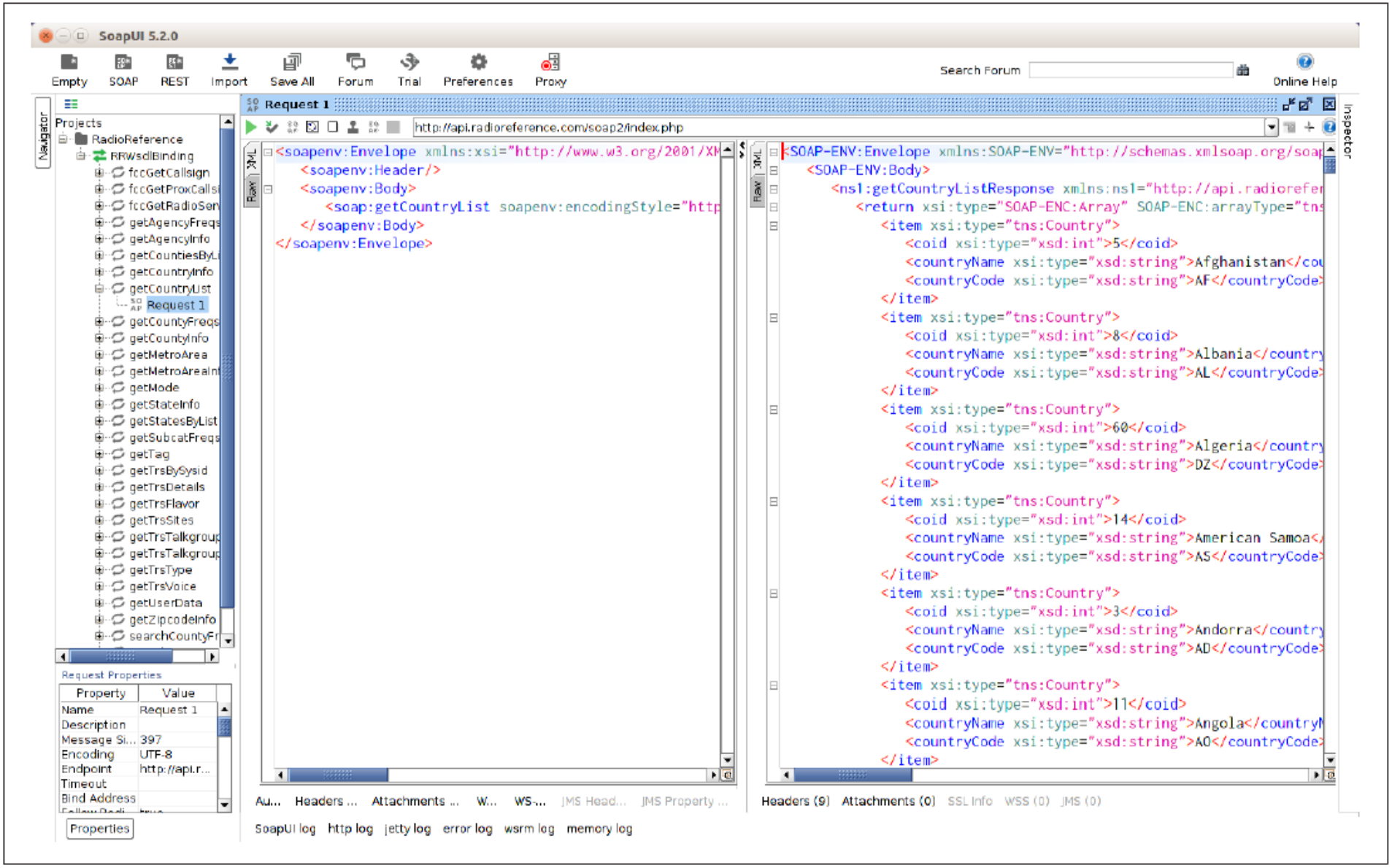
- SOAP era un tempo l'acronimo di Simple Object Access Protocol; tuttavia, questo è stato abbandonato e ora è solo "SOAP".

- SOAP è un servizio di tipo RPC che comunica tramite un formato XML molto strettamente specificato.

- Poiché SOAP è ben specificato quando segue le convenzioni WSDL (Web Services Description Language) , è necessario poco lavoro per implementarlo in un'applicazione o per integrarlo in esso.
  - WSDL descrive la posizione di un determinato servizio, i tipi di dati utilizzati in esso e i metodi, i parametri e i valori restituiti disponibili.
  - Il formato WSDL è XML piuttosto ostile, quindi è meglio generarlo e analizzarlo da macchine piuttosto che da esseri umani

- PHP ha un set davvero eccellente di librerie SOAP sia per client che per server.

SOAP - Simple Object Access Protocol



SOAP - Simple Object Access Protocol

- Un esempio di client per ottener una lista di paesi, dopo aver creato una interfaccia SOAP a partire da un WDSL, è il seguente

```php
<?php

$client = new SoapClient('http://api.radioreference.com/soap2/?wsdl&v=latest');

$countries = $client->getCountryList();
print_r($countries);
```

- Le nostre due righe di PHP si sono collegate a un servizio remoto e ci hanno recuperato un array di oggetti contenenti le informazioni sul paese come richiesto. Questo dimostra la gioia di SOAP, ovvero che sono necessarie pochissime righe di codice per scambiare dati tra i sistemi. La classe SoapClient in PHP rende banale il consumo di dati con un file WSDL.

SOAP - Simple Object Access Protocol

- Per quanto riguarda il server, PHP mette a disposizione delle librerie molti semplici da utilizzare

- Poiché nell'esempio precedente non viene utilizzato un WSDL, è necessario fornire l'URI per il servizio. Nell'esempio viene quindi creato il SoapServer e viene indicato quale classe contiene le funzionalità che deve esporre. Quando viene aggiunta la chiamata a un dle(), tutto "funziona e basta". Il PHP per chiamare il codice assomiglia molto all'esempio precedente, ma senza un file WSDL, è necessario dire al SoapClient dove trovare il servizio impostando il parametro location e passando l'URI:

```php
<?php

require('Events.php');

$options = array("uri" => "http://localhost");

$server = new SoapServer(null, $options);

$server->setClass('Events');
$server->handle();
```

```php
<?php

$options = array("location" => "http://localhost:8080/soap-server.php",
    "uri" => "http://localhost");

try {
    $client = new SoapClient(null, $options);
    $events = $client->getEvents();
    print_r($events);
} catch (SoapFault $e) {
    var_dump($e);
}
```

REST - REpresentational State Transfer

- A differenza di protocolli come SOAP o XML-RPC, è più una filosofia o un insieme di principi che un protocollo a sé stante.

- REST è un insieme di idee su come i dati possono essere trasferiti in modo elegante e, sebbene non sia legato a HTTP, viene discusso qui nel contesto di HTTP.

- REST sfrutta al meglio le funzionalità di HTTP, quindi i capitoli precedenti che trattano questo e gli argomenti più dettagliati di intestazioni e verbi possono unirsi per supportare una buona conoscenza di REST.

- In un servizio RESTful vengono usati quattro verbi HTTP per fornire un set di base di funzionalità CRUD (Create, Read, Update, Delete): POST, GET, PUT e DELETE.

- È anche possibile visualizzare implementazioni di altri verbi nei servizi RESTful, ad esempio PATCH, per consentire l'aggiornamento parziale di un record

REST - REpresentational State Transfer

- Il nome "Representational State Transfer" è accurato; I servizi RESTful si occupano del trasferimento di rappresentazioni di risorse.
- Una r**appresentazione** potrebbe essere JSON o XML, o qualsiasi altra cosa

- Tuttavia, le operazioni vengono applicate alle **risorse** di un sistema.

- Ogni singolo record di dati in un sistema è una risorsa.

- Nella prima fase di progettazione dell'API, un punto di partenza potrebbe essere quello di considerare ogni riga del database come una singola risorsa.
  - Si pensi a un sistema di blogging immaginario come esempio: le risorse potrebbero essere post, categorie e autori.

- Ogni risorsa ha un URI, che è l'identificatore univoco per il record.

- Una **raccolta** contiene più risorse (dello stesso tipo); Di solito questo è un elenco di risorse o il risultato di un'operazione di ricerca.
  - Un esempio di blog potrebbe avere una raccolta di post e un'altra raccolta di post limitata a una particolare categoria.

REST - REpresentational State Transfer

- I servizi RESTful sono spesso considerati come servizi "graziosi URL", ma le strutture utilizzate qui non sono solo belle

- Ad esempio, le API di GitHub sono anche un bell'esempio di API RESTful appartenente a un sistema con cui gli sviluppatori potrebbero già avere familiarità

  - https://api.github.com/users/lornajane/
  - https://api.github.com/users/lornajane/repos
  - https://api.github.com/users/lornajane/gists

- Questi URL deliziosi e descrittivi consentono agli utenti di indovinare cosa verrà trovato quando li visitano e di navigare facilmente in un sistema prevedibile e chiaramente progettato.

REST - REpresentational State Transfer

- Una caratteristica chiave degli URL RESTful è che contengono solo informazioni sulla risorsa o sui dati della raccolta: non ci sono verbi in questi URL.
-
- Per modificare la modalità di visualizzazione di una raccolta (ad esempio, per aggiungere filtri o ordinamenti a esso), è comune aggiungere parametri di query all'URL, in questo modo:

    - http://api.joind.in/v2.1/events per tutti gli eventi
    - http://api.joind.in/v2.1/events?filter=past per eventi accaduti prima di oggi
    - http://api.joind.in/v2.1/events?filter=cfp per eventi con un invito a presentare documenti attualmente aperto

- Si noti che gli URL non sono sulla falsariga di /events/sortBy/Past o qualsiasi altro formato questo inserisce variabili extra nell'URL, ma usano invece le variabili di query.
- Questo dataset, in entrambi i casi, utilizza ancora la collezione /events/, ma ordinata e/o filtrata secondo i filtri richiesti

- Il modo esatto in cui la risorsa viene restituita può variare enormemente
    - REST non determina come strutturare le rappresentazioni inviate.

- Questo particolare esempio ci consente semplicemente di aggiungere articoli a una lista dei desideri, assegnando loro un nome e collegandosi a dove è possibile trovare quel prodotto online

```php
<?php

require("ItemStorage.php");

set_exception_handler(function ($e) {
  $code = $e->getCode() ?: 400;
  header("Content-Type: application/json", NULL, $code);
  echo json_encode(["error" => $e->getMessage()]);
  exit;
});

// assume JSON, handle requests by verb and path
$verb = $_SERVER['REQUEST_METHOD'];
$url_pieces = explode('/', $_SERVER['PATH_INFO']);
$storage = new ItemStorage();

// catch this here, we don't support many routes yet
if($url_pieces[1] != 'items') {
  throw new Exception('Unknown endpoint', 404);
}
```

```php
switch($verb) {
  case 'GET':
    ...
    break;
  // two cases so similar we'll just share code
  case 'POST':
  case 'PUT':
    ...
    break;
  case 'DELETE':
    ...
  default:
    throw new Exception('Method Not Supported', 405);
}

// this is the output handler
header("Content-Type: application/json");
echo json_encode($data);
```

- Le risorse vengono create effettuando una richiesta POST alla raccolta a cui apparterrà la nuova risorsa. Il corpo della richiesta conterrà una rappresentazione della nuova risorsa, con l'intestazione Content-Type impostata in modo appropriato in modo che il server sappia come comprenderla. Quando la risorsa è stata creata correttamente, nella risposta verrà incluso un codice di stato riuscito.

```php
case 'POST':
case 'PUT':
  // read the JSON
  $params = json_decode(file_get_contents("php://input"), true);
  if(!$params) {
    throw new Exception("Data missing or invalid");
  }
  if($verb == 'PUT') {
    $id = $url_pieces[2];
    $item = $storage->update($id, $params);
    $status = 204;
  } else {
    $item = $storage->create($params);
    $status = 201;
  }
  $storage->save();

  // send header, avoid output handler
  header("Location: " . $item['url'], null,$status);
  exit;
  break;
```

```
$ curl -v -X POST -H "Content-Type: application/json" http://localhost:8080/
rest.php/items --data '{"link": "http://www.amazon.co.uk/dp/B00FWRIAUS/","name":
"notebook"}'
* Connected to localhost (127.0.0.1) port 8080 (#0)
> POST /rest.php/items HTTP/1.1
> User-Agent: curl/7.38.0
> Host: localhost:8080
> Accept: */*
> Content-Type: application/json
> Content-Length: 69
>
* upload completely sent off: 69 out of 69 bytes
< HTTP/1.1 201 Created
< Host: localhost:8080
< Connection: close
< X-Powered-By: PHP/5.6.4-4ubuntu6
< Location: http://localhost:8080/rest.php/items/dd44d
< Content-type: text/html; charset=UTF-8
<
* Closing connection 0
```

REST - REpresentational State Transfer

- Per recuperare le rappresentazioni delle risorse, utilizzare il verbo GET applicato a una raccolta o a una singola risorsa senza inviare alcun contenuto del corpo con la richiesta GET. Le risorse di solito appaiono esattamente con la stessa struttura, indipendentemente dal fatto che siano state richieste all'interno di una raccolta o da sole.

```php
case 'GET':
  if(isset($url_pieces[2])) {
    try {
      $data = $storage->getOne($url_pieces[2]);
    } catch (UnexpectedValueException $e) {
      throw new Exception("Resource does not exist", 404);
    }
  } else {
    $data = $storage->getAll();
  }
  break;
```

- *http://localhost:8080/rest.php/items*

per ottenere tutte le risorse della collezione

- *http://localhost:8080/rest.php/items/dd44d*

per il recupero di una singola risorsa

```
$ curl -s http://localhost:8080/rest.php/items/dd44d | python -mjson.tool
{
    "link": "http://www.amazon.co.uk/dp/B00FWRIAUS/",
    "name": "notebook",
    "url": "http://localhost:8080/rest.php/items/dd44d"
}
```

```
$ curl -v http://localhost:8080/rest.php/items/nonsense
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /rest.php/items/nonsense HTTP/1.1
> User-Agent: curl/7.38.0
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 404 Not Found
< Host: localhost:8080
< Connection: close
< X-Powered-By: PHP/5.6.4-4ubuntu6
< Content-Type: application/json
<
* Closing connection 0
{"error":"Resource does not exist"}
```

REST - REpresentational State Transfer

- Per modificare i record in modo RESTfully è un processo in più fasi. Innanzitutto, la risorsa deve essere recuperata da GET. Quindi, la rappresentazione della risorsa può essere modificata in base alle esigenze e tale risorsa deve essere rimessa all'URI originale. Anche se è necessario modificare solo una piccola parte del record, REST si occupa delle rappresentazioni delle risorse, quindi l'intera risorsa verrà recuperata e inviata per l'aggiornamento. Identica a quando una risorsa è stata creata utilizzando POST, la richiesta PUT includerà la rappresentazione della risorsa nel corpo e il Content-Type appropriato nell'intestazione.

```
$ curl -v -X PUT -H "Content-Type: application/json" http://localhost:8080/
rest.php/items/dd44d --data '{"link": "http://www.amazon.co.uk/dp/
B00FWRIAUS/","name": "awesome notebook"}'
* Connected to localhost (127.0.0.1) port 8080 (#0)
> PUT /rest.php/items/dd44d HTTP/1.1
> User-Agent: curl/7.38.0
> Host: localhost:8080
> Accept: */*
> Content-Type: application/json
> Content-Length: 77
>
* upload completely sent off: 77 out of 77 bytes
< HTTP/1.1 204 No Content
< Host: localhost:8080
< Connection: close
< X-Powered-By: PHP/5.6.4-4ubuntu6
< Location: http://localhost:8080/rest.php/items/dd44d
< Content-type: text/html; charset=UTF-8
<
* Closing connection 0
```

- Il verbo DELETE viene inviato con una richiesta all'URI dell'elemento da eliminare, senza che sia necessario alcun contenuto del corpo. Molti servizi restituiscono 200 per "OK", o semplicemente 204 per "Nessun contenuto", quando un elemento è stato eliminato correttamente e un 404 "Non trovato" se l'elemento non esiste.

```php
case 'DELETE':
    $id = $url_pieces[2];
    $storage->remove($id);
    $storage->save();
    header("Location: http://localhost:8080/items", null, 204);
    exit;
    break;
```

```
$ curl -X DELETE -v http://localhost:8080/rest.php/items/958a9
* Connected to localhost (127.0.0.1) port 8080 (#0)
> DELETE /rest.php/items/958a9 HTTP/1.1
> User-Agent: curl/7.38.0
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 204 No Content
< Host: localhost:8080
< Connection: close
< X-Powered-By: PHP/5.6.4-4ubuntu6
< Location: http://localhost:8080/items
< Content-type: text/html; charset=UTF-8
<
* Closing connection 0
```

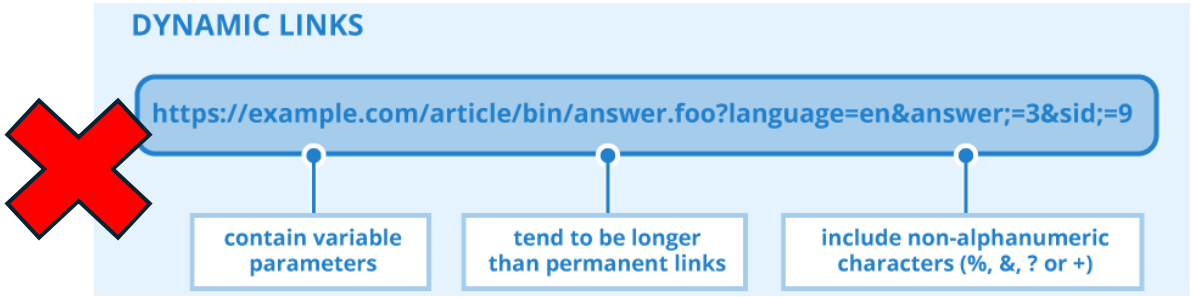
Permalink

- Un permalink o collegamento permanente è un tipo di URL che si riferisce a una specifica informazione, implementato in modo da non cambiare o almeno da rimanere lo stesso per lunghi periodi di tempo.

`https://www.kasadellacomunicazione.it/archives/123`

`https://www.kasadellacomunicazione.it/2021/04/07/articolo-di-esempio/`

`https://www.kasadellacomunicazione.it/?p=123`



MVC – Model-View-Controller

- MVC è il tipo più comune di modello architettonico che gli sviluppatori PHP incontrano.

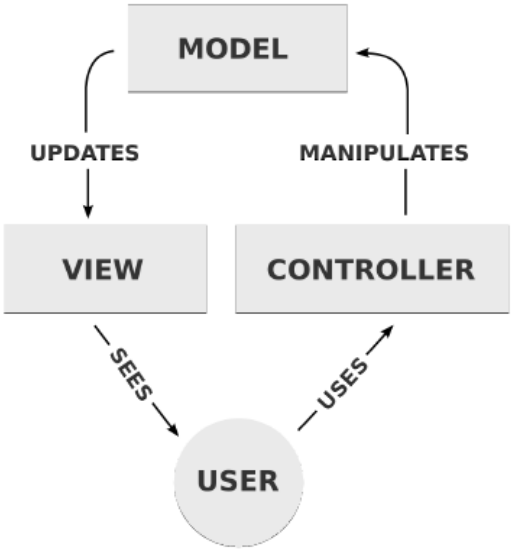
- Fondamentalmente, MVC è un modello architettonico per l'implementazione delle interfacce utente

- Funziona in gran parte intorno alla seguente metodologia:

  - **Model**: fornisce i dati all'applicazione, indipendentemente dal fatto che provengano da un database MySQL o da qualsiasi altro archivio dati.

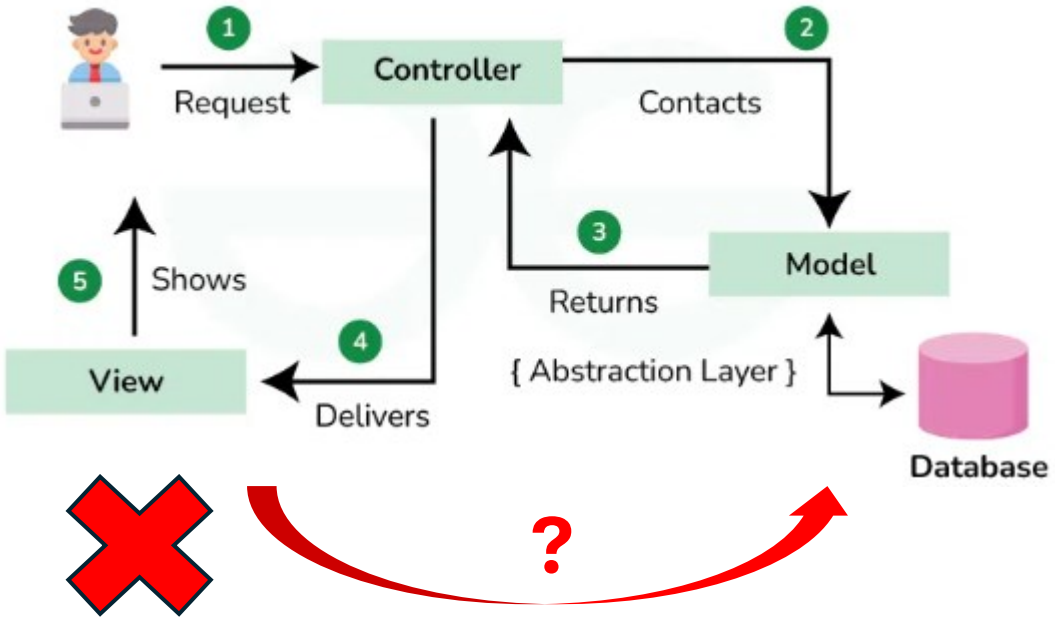
  - **Controller**: un controller è essenzialmente dove si trova la logica di business. Il controller gestisce tutte le query fornite dalla visualizzazione, usando il modello per assisterlo in questo comportamento.

  - **View**: il contenuto effettivo fornito all'utente finale. Questo è comunemente un modello HTML.



Il controller funge da intermediario tra il modello e la vista. Gestisce l'input dell'utente e aggiorna il modello di conseguenza, e aggiorna la vista per riflettere le modifiche nel modello. Contiene la logica dell'applicazione, come la convalida dell'input e la trasformazione dei dati.

Comunicazione tra i componenti
Il flusso di comunicazione sottostante garantisce che ogni componente sia responsabile di un aspetto specifico della funzionalità dell'applicazione, portando a un'architettura più manutenibile e scalabile



MVC – Model-View-Controller

- **Interazione dell'utente con la vista**: l'utente interagisce con la vista, ad esempio cliccando su un pulsante o inserendo del testo in un modulo.
- **La vista riceve l'input dell'utente**: la vista riceve l'input dell'utente e lo inoltra al controller.
- **Il controller elabora l'input dell'utente**: il controller riceve l'input dell'utente dalla View. Interpreta l'input, esegue tutte le operazioni necessarie (come l'aggiornamento del modello) e decide come rispondere.
- **Il controller aggiorna il modello**: il controller aggiorna il modello in base all'input dell'utente o alla logica dell'applicazione.
- **Il modello notifica alla vista le modifiche**: se il modello cambia, lo notifica alla vista.
- **La vista richiede dati dal modello**: la vista richiede dati dal modello per aggiornare la sua visualizzazione.
- **Il controller aggiorna la vista**: il controller aggiorna la vista in base alle modifiche apportate al modello o in risposta all'input dell'utente.
- **La vista esegue il rendering dell'interfaccia utente aggiornata**: la vista esegue il rendering dell'interfaccia utente aggiornata in base alle modifiche apportate dal controller.

MVC – Un esempio in Java

```java
class Student {
    private String rollNo;
    private String name;

    public String getRollNo() {
        return rollNo;
    }

    public void setRollNo(String rollNo) {
        this.rollNo = rollNo;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

class StudentView {
    public void printStudentDetails(String studentName, String
studentRollNo) {
        System.out.println("Student:");
        System.out.println("Name: " + studentName);
        System.out.println("Roll No: " + studentRollNo);
    }
}
```

```java
class StudentController {
    private Student model;
    private StudentView view;

    public StudentController(Student model, StudentView view) {
        this.model = model;
        this.view = view;
    }

    public void setStudentName(String name) {
        model.setName(name);
    }

    public String getStudentName() {
        return model.getName();
    }

    public void setStudentRollNo(String rollNo) {
        model.setRollNo(rollNo);
    }

    public String getStudentRollNo() {
        return model.getRollNo();
    }

    public void updateView() {
        view.printStudentDetails(model.getName(), model.getRollNo());
    }
}
```

```java
public class MVCPattern {
    public static void main(String[] args) {
        Student model = retriveStudentFromDatabase();

        StudentView view = new StudentView();

        StudentController controller = new StudentController(model,
view);

        controller.updateView();

        controller.setStudentName("Vikram Sharma");

        controller.updateView();
    }

    private static Student retriveStudentFromDatabase() {
        Student student = new Student();
        student.setName("Lokesh Sharma");
        student.setRollNo("15UCS157");
        return student;
    }
}
```

MVC – Un esempio in PHP

index.php

PHP Code:

```php
<?php

include 'Router.php';
include 'UserController.php';

$router = new Router(require 'routes.php');
$router->route($_SERVER['REQUEST_URI']);

?>
```

UserController.php

PHP Code:

```php
<?php

class UserController {
    public function show($id) {
        // Logic to handle user with the specified ID
        echo "Showing user with ID: $id";
    }
    public function showPost($uid, $pid) {
        // Logic to handle user with the specified ID and post with the specified ID
        echo "Showing post with ID $pid for user with ID $uid";
    }
}

?>
```

MVC – Un esempio in PHP

router.php

PHP Code:

```php
<?php

class Router {

    private $routes;

    public function __construct(array $routes) {
        $this->routes = $routes;
    }

    public function route($url) {
        foreach ($this->routes as $route => $handler) {
            $pattern = $this->buildPattern($route);
            if (preg_match($pattern, $url, $matches)) {
                array_shift($matches); // Remove the full match
                $this->invokeControllerAction($handler, $matches);
                return;
            }
        }
        // Handle 404 - Route not found
        $this->handle404();
    }

    private function buildPattern($route) {
        return '#^' . preg_replace('/\{(\w+)\}/', '(?P<$1>[^/]+)', $route) . '$#';
    }

    private function invokeControllerAction($handler, $params) {
        list($controllerName, $action) = explode('@', $handler);
        $controller = new $controllerName();
        $this->callControllerAction($controller, $action, $params);
    }

    private function callControllerAction($controller, $action, $params) {
        $reflectionMethod = new ReflectionMethod($controller, $action);
        $methodParameters = $reflectionMethod->getParameters();

        $resolvedParams = [];
        foreach ($methodParameters as $param) {
            $paramName = $param->getName();
            $resolvedParams[] = $params[$paramName] ?? null;
        }

        $reflectionMethod->invokeArgs($controller, $resolvedParams);
    }

    private function handle404() {
        // Handle the 404 error
        echo "404 Not Found";
    }
}

?>
```

MVC – Un esempio in PHP

routes.php

PHP Code:

```php
<?php

return [
    '/testRouter/users/{id}'                => 'UserController@show',
    '/testRouter/users/{uid}/posts/{pid}'   => 'UserController@showPost',
    // Other routes...
];

?>
```

MVC – Esempio 2

PHP Code:

```php
class UserModel {
    private $db;

    public function __construct($database) {
        $this->db = $database;
    }

    public function getUser($id) {
        $stmt = $this->db->prepare("SELECT * FROM users WHERE id = ?");
        $stmt->execute([$id]);
        return $stmt->fetch();
    }

    public function createUser($data) {
        $stmt = $this->db->prepare("INSERT INTO users (name, email) VALUES (?, ?)");
        return $stmt->execute([$data['name'], $data['email']]);
    }

    // More CRUD operations...
}
```

MVC – Esempio 2

PHP Code:

```php
class UserController {
    private $model;

    public function __construct($model) {
        $this->model = $model;
    }

    public function showUser($id) {
        $user = $this->model->getUser($id);
        include 'user_view.php';
    }

    public function createUser($data) {
        $this->model->createUser($data);
        header('Location: /users');
    }

    // More actions...
}
```

PHP Code:

```php
// user_view.php
<!DOCTYPE html>
<html>
<head>
    <title>User Details</title>
</head>
<body>
    <h1>User Details</h1>
    <p>Name: <?= htmlspecialchars($user['name']) ?></p>
    <p>Email: <?= htmlspecialchars($user['email']) ?></p>
</body>
</html>
```

PHP Code:

```php
// index.php
require 'models/UserModel.php';
require 'controllers/UserController.php';

$database = new PDO('mysql:host=localhost;dbname=testdb', 'root', '');
$model = new UserModel($database);
$controller = new UserController($model);

if ($_SERVER['REQUEST_METHOD'] === 'GET' && isset($_GET['id'])) {
    $controller->showUser($_GET['id']);
} elseif ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $controller->createUser($_POST);
} else {
    // Default action or error handling
}
```

MVC – Esempio 3 : un Dispatcher più elegante

```php
class Dispatcher {
    protected $params;
    protected $controller;
    public function __construct($controller) {
        $this->controller = $controller;
        $this->params = $this->parse($_SERVER['REQUEST_URI']);
    }
    public function handle() {
        $action = $this->params['method'];
        $args = $this->params['args'];
        if(method_exists($this->controller, $action)) {
            if(!is_null($args)) {
                $this->controller->$action($args);
            } else {
                $this->controller->$action();
            }
        } else {
            Site::error();
        }
    }
```

MVC – Esempio 3 : un Dispatcher più elegante

```
    protected function parse($path) {
        if($path == '/') {
            return [
                'method' => 'index',
                'args' => null
            ];
        } else {
            $parts = array_values(array_filter(explode('/', $this->removeQueryStringVariables($path))));
            $method = $this->convertToCamelCase($parts[0]);
            $args = array_slice($parts, 1);
            return [
                'method' => $method,
                'args' => (count($args) > 0 ) ? $args : null
            ];
        }
    }
    protected function convertToStudlyCaps($string) {
        return str_replace(' ', '', ucwords(str_replace('-', ' ', $string)));
    }
    protected function convertToCamelCase($string) {
        return lcfirst($this->convertToStudlyCaps($string));
    }
    protected function removeQueryStringVariables($url) {
        $parts = explode('?', $url);
        return $parts[0];
    }
}
```

MVC – Un esempio in PHP

Parte della magia è perché .htaccess invia tutte le richieste a index.php e index PHP avvia il router e gli passa l'URI.

Quindi, la classe router suddivide l'URI in parti e le confronta con le rotte registrate, quindi richiama le classi, chiama le azioni e passa i parametri nell'azione/metodo.

.htaccess

Code:

```
<IfModule mod_rewrite.c>
    # Send Requests To Front Controller...
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>
```

Prima però bisogna abilitare il modulo mod_rewrite di Apache
https://httpd.apache.org/docs/2.4/rewrite/intro.html

```
sudo apt install apache2
sudo a2enmod rewrite
sudo nano /etc/apache2/sites-available/000-default.conf
        <Directory /var/www/html>
           AllowOverride All
        </Directory>
sudo systemctl restart apache2
```

Quando utilizzare il modello di progettazione MVC:

- **Applicazioni complesse :** usa MVC per app con molte funzionalità e interazioni utente, come i siti di e-commerce. Aiuta a organizzare il codice e a gestire la complessità.

- **Modifiche frequenti all'interfaccia utente :** se l'interfaccia utente necessita di aggiornamenti regolari, MVC consente modifiche alla vista senza influire sulla logica sottostante.

- **Riutilizzabilità dei componenti :** se vuoi riutilizzare parti della tua app in altri progetti, la struttura modulare di MVC semplifica questa operazione.

- **Requisiti di test :** MVC supporta test approfonditi, consentendo di testare ogni componente separatamente per un migliore controllo di qualità.

Quando non utilizzare il modello di progettazione MVC:

- **Applicazioni semplici :** per piccole app con funzionalità limitate, MVC può aggiungere complessità non necessaria. Un approccio più semplice potrebbe essere migliore.

- **Applicazioni in tempo reale :** MVC potrebbe non funzionare bene per le app che richiedono aggiornamenti immediati, come i giochi online o le app di chat.

- **Interfaccia utente e logica strettamente collegate** : se l'interfaccia utente e la logica aziendale sono strettamente collegate, MVC potrebbe complicare ulteriormente le cose.

- **Risorse limitate :** per i team di piccole dimensioni o per coloro che non hanno familiarità con MVC, progetti più semplici possono portare a uno sviluppo più rapido e a meno problemi.

MVP – Model-View-Presenter

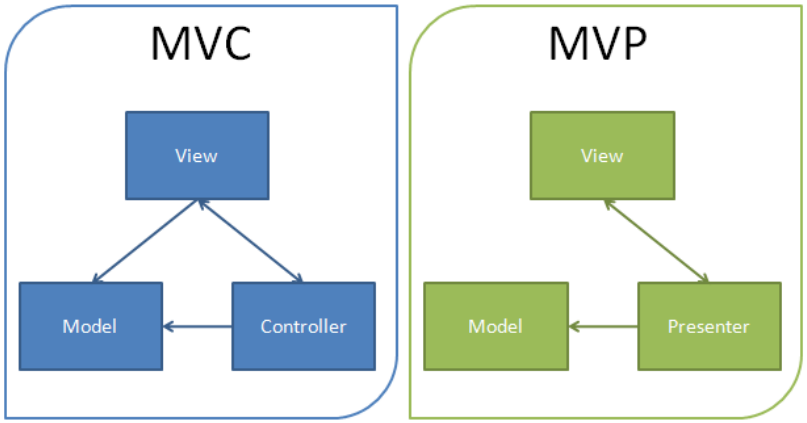
- La logica di business per un'interazione non è strettamente separata da un'altra interazione. Non esiste una separazione formale tra le diverse classi di un'applicazione.

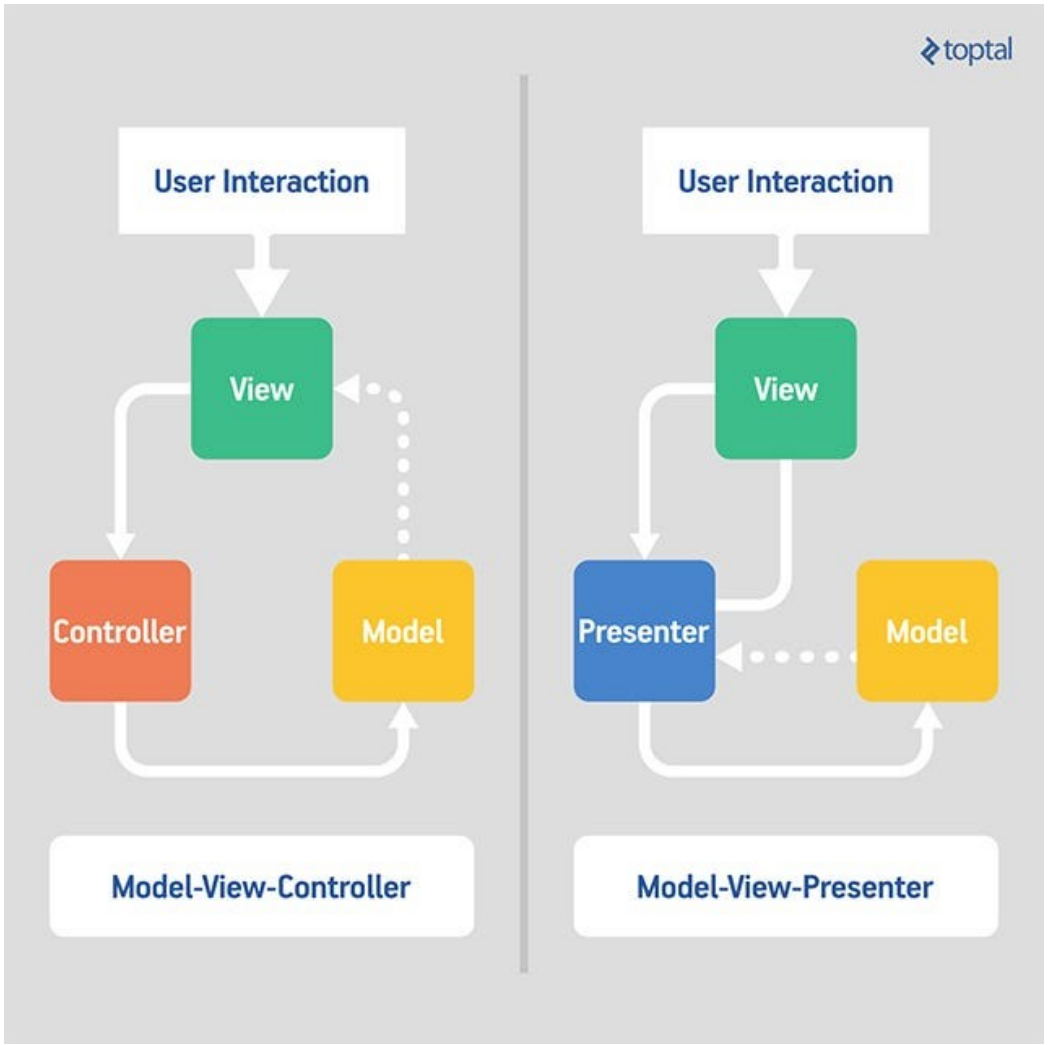
- È fondamentale considerare che il modello MVC è principalmente un modello dell'interfaccia utente, quindi non viene ridimensionato correttamente in un'applicazione. Detto questo, il rendering delle interfacce utente viene sempre più eseguito tramite applicazioni JavaScript, un'app HTML JavaScript a pagina singola che utilizza semplicemente un'API RESTfull
  - Se stai utilizzando JavaScript, puoi utilizzare un framework come Backbone.js (Model-View-Presenter), React.js o Angular.

- Nel caso in cui ci si trovi in un ambiente in cui non è possibile usare un'app JavaScript e si deve invece gestire il codice HTML di cui è stato eseguito il rendering, spesso è consigliabile che l'app MVC utilizzi semplicemente un'API REST. L'API REST esegue tutta la logica di business, ma il rendering del markup viene eseguito nell'app MVC. Sebbene ciò aumenti la complessità, offre una maggiore separazione delle responsabilità e, di conseguenza, non è necessario che l'HTML venga unito alla logica di business principale. Detto questo, anche all'interno di questa API REST è necessaria una qualche forma di separazione delle preoccupazioni, è necessario essere in grado di separare il rendering del markup dalla logica di business effettiva.

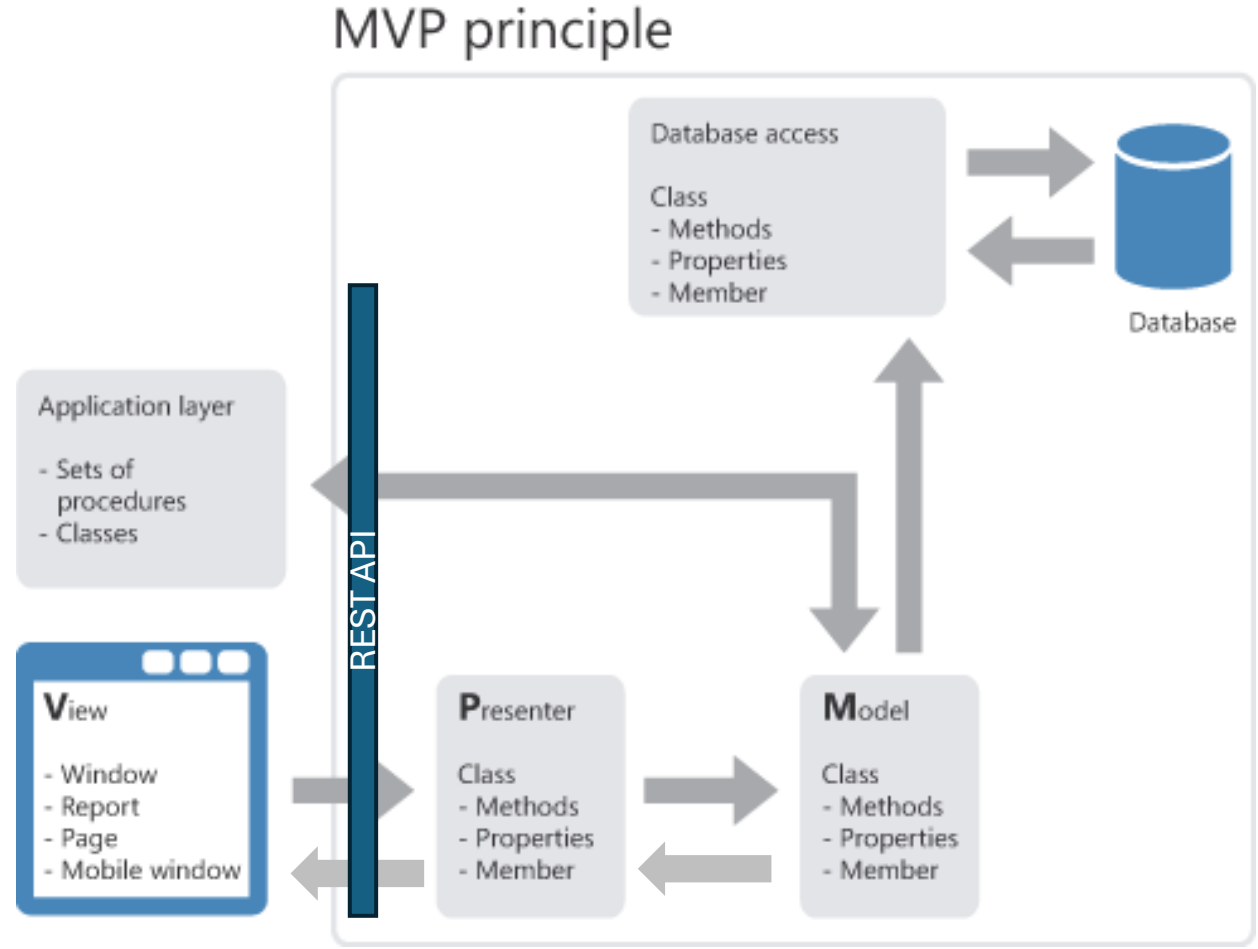
MVP – Model-View-Presenter

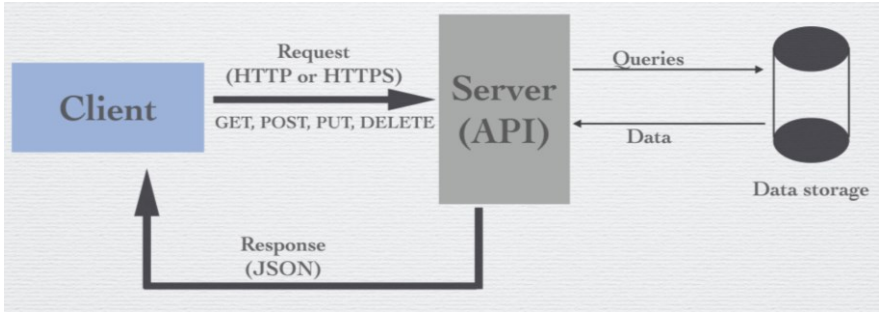
- In MVP, tutta la logica di presentazione viene inviata al presentatore.





MVP – Model-View-Presenter





REST API – Un esempio (pasticciato)

.htaccess

```
RewriteEngine on
RewriteRule . index.php
```

index.php

```php
<?php

spl_autoload_register(function ($class) {
    require __DIR__ . DIRECTORY_SEPARATOR . "$class.php";
});

$api_prefix = str_replace("index.php","",$_SERVER["SCRIPT_NAME"]);

$controller = new RESTAPIController;
$controller->set_api_prefix($api_prefix);
$controller->handle_request();

?>
```

Many developers writing object-oriented applications create one PHP source file per class definition. One of the biggest annoyances is having to write a long list of needed includes at the beginning of each script (one for each class).
The spl_autoload_register() function registers any number of autoloaders, enabling for classes and interfaces to be automatically loaded if they are currently not defined. By registering autoloaders, PHP is given a last chance to load the class or interface before it fails with an error.

REST API – Un esempio (pasticciato)

RESTAPIController.php

```php
<?php
class RESTAPIController{
    private $api_prefix = "";

    public function set_api_prefix($prefix){
        $this->api_prefix = $prefix;
    }

    public function handle_request(){
        $uri = preg_replace('/^'. preg_quote($this->api_prefix, '/') .'/', "", $_SERVER["REQUEST_URI"]);
        $uri = preg_replace('/\\/$/', "", $uri);

        $parts = explode("/", $uri);



        switch($parts[0]){
            case 'strains':
                $gateway = new StrainsGateway;
                break;
            case 'gm':
                $gateway = new GrowthMediaGateway;
                break;
            case 'localities':
                $gateway = new LocalitiesGateway;
                break;
            default:
                ErrorHandler::bad_request();
                return;
        }

            try{
                $gateway->handle_request($parts);
            }catch(Exception $e){
                var_dump($e);
                ErrorHandler::bad_request();
                return;
            }

        }
    };

    ?>
```

REST API – Un esempio (pasticciato)

GrowthMediaGateway.php

```php
<?php
class GrowthMediaGateway extends Gateway{
    public function handle_request($parts){
        if ($_SERVER["REQUEST_METHOD"] == "GET"){
            if( (count($parts) > 1) and ( $parts[1] == 'list' ) ){

                $db = DBConnectionFactory::getFactory();
                $sql = "SELECT * FROM upcc_growth_media";

                $rows = $db->fetch_all($sql);


                header("Access-Control-Allow-Origin: *");//this allows
                header('Content-Type: application/json');

                print('[');

                $ni = 0;
                $nofprinted = 0;
                foreach($rows as $row){

                    $orow['acronym'] = $row['acronym'];
                    $orow['description'] = trim($row['description']);
                    $orow['fullDescription'] = trim($row['full_description']);

                    if(strlen($orow['description'])==0){
                        $orow['description'] = "no description";
                    }
                    if(strlen($orow['fullDescription'])==0){
                        $orow['fullDescription'] = "no description";
                    }

                    if($nofprinted>0) print(',');
                    print(json_encode($orow));
                    $nofprinted++;

                    $ni++;
                }

                print(']');


            }
            else{
                //echo "devo fare qualcosa altro";
                ErrorHandler::bad_request();
            }
        }
        else{
            ErrorHandler::bad_request();
        }
    }
}
?>
```

Gateway.php

```php
<?php
abstract class Gateway{
    abstract public function handle_request($parts);
}
?>
```

REST API – Un esempio (pasticciato)

DBConnectionFactory.php

```php
<?php

class DBConnectionFactory{

private static $factory;
public static function getFactory(){
    if (!self::$factory){
        self::$factory = new DBConnectionFactory();
        self::$factory->db = null;
    }
    return self::$factory;
}

private $db;

public function getConnection(){
    if (is_null($this->db))
        $this->db = new mysqli(
        if ($this->db->connect_error){
            throw new Exception("Connect Error ("
                . $this->db->connect_errno
                . ") "
                . $this->db->connect_error
            );
        }
    return $this->db;
}

public function closeConnection(){
    if (! is_null($this->db)){
        $this->db::close();
        $this->db = null;
    }
}


public function fetch_all($sql){
    $result = $this->getConnection()->query($sql);
    return $result->fetch_all(MYSQLI_ASSOC);
}

}


?>
```

REST API – Un esempio (pasticciato)

ErrorHandler.php

```php
<?php
class ErrorHandler{
    public static function bad_request(){
        http_response_code(400);
        echo json_encode(array(
            "success" => false,
            "error" => array(
            "code" => 400,
            "message" => "bad request"
            )));
    }
}
?>
```