REST API in PHP 8

Sviluppo di Servizi Web Moderni con MySQL

Prof. Fedeli Massimo

IIS Fermi Sacconi Ceci - Ascoli Piceno

January 4, 2026

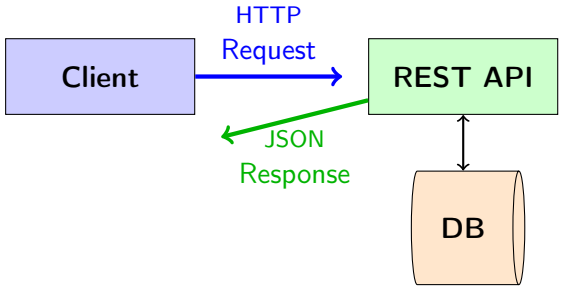
Indice

Cosa sono le REST API?

REST (Representational State Transfer) è uno stile architetturale per servizi web.

Caratteristiche principali:

- ✓ Stateless (senza stato)
- ✓ Client-Server separation
- ✓ Cacheable
- ✓ Interfaccia uniforme
- ✓ Sistema a strati



- REST è uno stile architetturale per la progettazione di sistemi distribuiti, in particolare servizi web. Non è una tecnologia in senso stretto né un protocollo, ma un insieme di principi che guidano la costruzione di API semplici, scalabili e manutenibili.
- REST non impone regole rigide, ma suggerisce buone pratiche.
- Oggi REST è lo standard dominante per le API web, soprattutto in contesti microservizi, applicazioni web e mobile. Framework come Spring Boot, Django REST Framework, FastAPI o Express lo supportano nativamente.

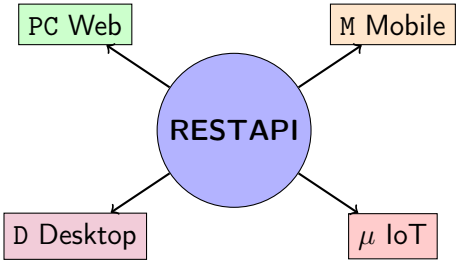
Vantaggi delle REST API

Per gli sviluppatori:

- ⟨⟩ Facilità di implementazione
- [] Separazione front-end/back-end
- ↻ Riutilizzabilità del codice
- ⋆ Supporto di framework moderni

Per le applicazioni:

- ↔ Scalabilità
- △ Sicurezza migliorata
- M Multi-piattaforma



Un unica API per tutti i client

Metodi HTTP nelle REST API

Metodo	Operazione	Esempio
GET	Recupera risorse	`GET /users` `GET /users/5`
POST	Crea nuove risorse	`POST /users`
PUT	Aggiorna risorse (completo)	`PUT /users/5`
PATCH	Aggiorna risorse (parziale)	`PATCH /users/5`
DELETE	Elimina risorse	`DELETE /users/5`

REST impone una semantica precisa ai metodi HTTP.

CRUD Operations

- **C**reate → POST
- **R**ead → GET
- **U**pdate → PUT/PATCH
- **D**elete → DELETE

Codici di Stato HTTP

2xx - Successo

- 200 OK - Richiesta riuscita
- 201 Created - Risorsa creata
- 204 No Content - Successo senza contenuto

4xx - Errori Client

- 400 Bad Request - Richiesta malformata
- 401 Unauthorized - Non autenticato
- 404 Not Found - Risorsa non trovata
- 422 Unprocessable - Dati non validi

5xx - Errori Server

- 500 Internal Error - Errore del server
- 503 Unavailable - Servizio non disponibile

Best Practice

Utilizzare sempre i codici di stato appropriati per comunicare chiaramente il risultato dell'operazione.

Struttura del Progetto

```
rest-api-php/
|-- index.php
|-- inc/
|   |-- config.php
|   |-- bootstrap.php
|-- Model/
|   |-- Database.php
|   |-- UserModel.php
|-- Controller/
    |-- Api/
        |-- BaseController.php
        |-- UserController.php
```

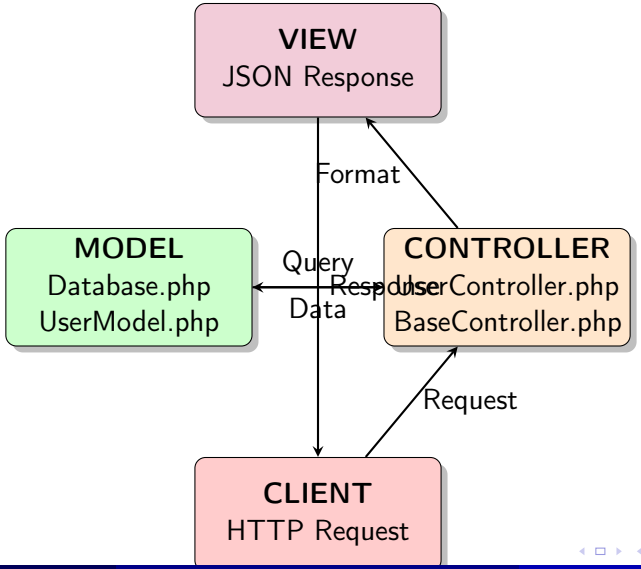
Componenti principali:

- **index.php**: Entry point dell'applicazione
- **inc/**: File di configurazione e bootstrap
- **Model/**: Data Access Layer
- **Controller/**: Logica di business

Pattern MVC

Adottiamo il pattern Model-View-Controller per separare le responsabilità e migliorare la manutenibilità del codice.

Pattern MVC nell'API REST



Creazione del Database MySQL

Step 1: Creare il database

```
CREATE DATABASE rest_api_php;
```

Step 2: Creare la tabella users

```
USE rest_api_php;

CREATE TABLE `users` (
  `user_id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(60) COLLATE utf8mb4_unicode_ci
             NOT NULL DEFAULT '',
  `user_email` varchar(100) COLLATE utf8mb4_unicode_ci
               NOT NULL DEFAULT '',
  `user_status` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
  COLLATE=utf8mb4_unicode_ci;
```

Popolamento del Database

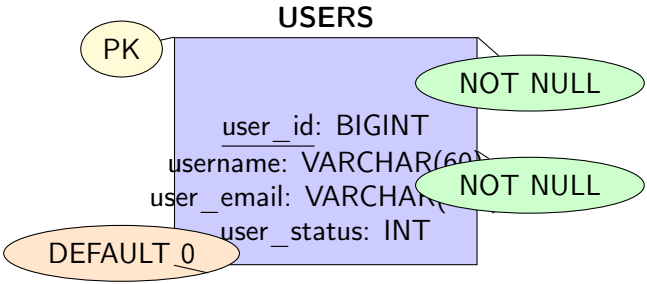
Inserimento dati di esempio:

```
INSERT INTO `users`
  (`user_id`, `username`, `user_email`, `user_status`)
VALUES
  (1, 'Gina', 'ginaverdi@gmail.com', 0),
  (2, 'Mario', 'mario.rossi@email.com', 1),
  (3, 'Laura', 'laura.bianchi@email.com', 1),
  (4, 'Paolo', 'paolo.verdi@email.com', 0),
  (5, 'Sara', 'sara.neri@email.com', 1);
```

Nota

Il campo `user_status` può indicare se l'utente è attivo (1) o inattivo (0).

Schema Entity-Relationship



Vincoli:

- user_id è la chiave primaria (AUTO_INCREMENT)
- username e user_email sono obbligatori
- user_status ha valore predefinito 0

File di Configurazione - config.php

```php
<?php
// inc/config.php

define("DB_HOST", "localhost");
define("DB_USERNAME", "root");
define("DB_PASSWORD", "");
define("DB_DATABASE_NAME", "rest_api_php");
?>
```

Best Practice

- Separare sempre la configurazione dal codice
- Utilizzare variabili d'ambiente in produzione
- Non versionare mai le credenziali su Git
- Considerare l'uso di file .env

File Bootstrap

```php
<?php
// inc/bootstrap.php

define("PROJECT_ROOT_PATH", __DIR__ . "/../");

// Include main configuration file
require_once PROJECT_ROOT_PATH . "/inc/config.php";

// Include the base controller file
require_once PROJECT_ROOT_PATH .
    "/Controller/Api/BaseController.php";

// Include the user model file
require_once PROJECT_ROOT_PATH .
    "/Model/UserModel.php";
?>
```

Il file bootstrap carica tutti i componenti necessari all'avvio dell'applicazione, definisce il path root e

Classe Database - Parte 1

```php
<?php
// Model/Database.php

class Database
{
    protected $connection = null;

    public function __construct()
    {
        try {
            $this->connection = new mysqli(
                DB_HOST,
                DB_USERNAME,
                DB_PASSWORD,
                DB_DATABASE_NAME
            );

            if (mysqli_connect_errno()) {
                throw new Exception(
                    "Could not connect to database."
                );
            }
```

Classe Database - Parte 2

```php
    public function select($query = "", $params = [])
    {
      try {
        $stmt = $this->executeStatement($query, $params);
        $result = $stmt->get_result()
        ->fetch_all(MYSQLI_ASSOC);
        $stmt->close();
        return $result;
      } catch (Exception $e) {
        throw new Exception($e->getMessage());
      }
    }
```

Classe Database - Parte 2

```
1
2     private function executeStatement($query = "", $params = [])
3     {
4       try {
5         $stmt = $this->connection->prepare($query);
6         if ($stmt === false) {
7           throw new Exception(
8           "Unable to prepare statement: " . $query
9           );
10        }
11        if ($params) {
12          $stmt->bind_param($params[0], $params[1]);
13        }
14        $stmt->execute();
15        return $stmt;
16      } catch (Exception $e) {
17        throw new Exception($e->getMessage());
18      }
19    }
20  }
```

Caratteristiche della Classe Database

Funzionalità:

- ○ Gestione connessione MySQL
- △ Prepared Statements (SQL Injection)
- ! Gestione eccezioni
- [] Pattern Template per estensione

Metodi principali:

- `__construct()`: Stabilisce connessione
- `select()`: Esegue query SELECT
- `executeStatement()`: Prepara ed esegue statement

Sicurezza

L'uso di **Prepared Statements** con `bind_param()` protegge da attacchi SQL Injection parametrizzando le query.

Estensibilità

La classe è progettata per essere estesa da Model specifici (UserModel, ProductModel, ecc.) che implementano la logica di business.

Classe UserModel

```php
<?php
// Model/UserModel.php

require_once PROJECT_ROOT_PATH . "/Model/Database.php";

class UserModel extends Database
{
    public function getUsers($limit)
    {
        return $this->select(
            "SELECT * FROM users
             ORDER BY user_id ASC
             LIMIT ?",
            ["i", $limit]
        );
    }
}
?>
```

Classe BaseController - Parte 1

```php
<?php
// Controller/Api/BaseController.php

class BaseController
{
    /**
     * Magic method per gestire chiamate a metodi inesistenti
     */
    public function __call($name, $arguments)
    {
        $this->sendOutput('',
            array('HTTP/1.1 404 Not Found'));
    }
```

Classe BaseController - Parte 1

```
1
2       /**
3       * Estrae i segmenti dall'URI
4       */
5       protected function getUriSegments()
6       {
7         $uri = parse_url($_SERVER['REQUEST_URI'],
8         PHP_URL_PATH);
9         $uri = explode('/', $uri);
10        return $uri;
11      }
```

Classe BaseController - Parte 2

```php
    /**
     * Recupera parametri della query string
     */
    protected function getQueryStringParams()
    {
        parse_str($_SERVER['QUERY_STRING'], $query);
        return $query;
    }

    /**
     * Invia l'output JSON al client
     */
    protected function sendOutput($data,
                                  $httpHeaders = array())
    {
        header_remove('Set-Cookie');

        if (is_array($httpHeaders) && count($httpHeaders)) {
            foreach ($httpHeaders as $httpHeader) {
                header($httpHeader);
            }
        }
```

Metodi della Classe BaseController

Questi metodi forniscono funzionalità comuni a tutti i controller dell API.

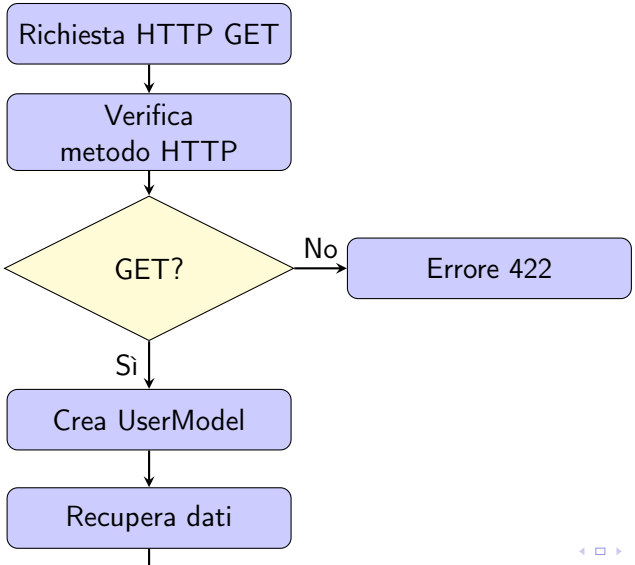
Classe UserController - Parte 1

```php
<?php
// Controller/Api/UserController.php

class UserController extends BaseController
{
    /**
     * Endpoint: /user/list
     * Restituisce l'elenco degli utenti
     */
    public function listAction()
    {
        $strErrorDesc = '';
        $requestMethod = $_SERVER["REQUEST_METHOD"];
        $arrQueryStringParams = $this->getQueryStringParams();

        if (strtoupper($requestMethod) == 'GET') {
            try {
                $userModel = new UserModel();
                $intLimit = 10;

                if (isset($arrQueryStringParams['limit']) &&
                    $arrQueryStringParams['limit']) {
                    $intLimit = $arrQueryStringParams['limit'];
                }

                $arrUsers = $userModel->getUsers($intLimit);
                $responseData = json_encode($arrUsers);
```

Classe UserController - Parte 2

```
            } catch (Error $e) {
                $strErrorDesc = $e->getMessage() .
                    'Something went wrong! Please contact support.';
                $strErrorHeader = 'HTTP/1.1 500 Internal Server Error';
            }
        } else {
            $strErrorDesc = 'Method not supported';
            $strErrorHeader = 'HTTP/1.1 422 Unprocessable Entity';
        }

        // Invia output
        if (!$strErrorDesc) {
            $this->sendOutput(
                $responseData,
                array('Content-Type: application/json',
                      'HTTP/1.1 200 OK')
            );
        } else {
            $this->sendOutput(
                json_encode(array('error' => $strErrorDesc)),
                array('Content-Type: application/json',
                      $strErrorHeader)
            );
        }
    }
}
```

Flusso di Esecuzione in UserController



Entry Point - index.php

```php
<?php
// index.php

require __DIR__ . "/inc/bootstrap.php";

$uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
$uri = explode('/', $uri);

// Validazione URI
if ((isset($uri[2]) && $uri[2] != 'user') ||
    !isset($uri[3])) {
    header("HTTP/1.1 404 Not Found");
    exit();
}

// Carica il controller appropriato
require PROJECT_ROOT_PATH .
    "/Controller/Api/UserController.php";

$objFeedController = new UserController();
$strMethodName = $uri[3] . 'Action';
$objFeedController->{$strMethodName}();
```

Analisi del Routing URI

Struttura URI:

`http://localhost/index.php/user/list?limit=20`

Componenti URI:

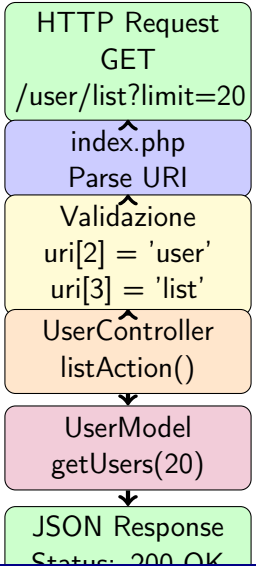
- `$uri[0]` → '' (vuoto)
- `$uri[1]` → 'index.php'
- `$uri[2]` → 'user' (modulo)
- `$uri[3]` → 'list' (metodo)

Processo:

1. Parse dell'URI
2. Validazione modulo
3. Caricamento controller
4. Invocazione metodo dinamico

`$uri[3] . 'Action'` → `listAction()`

Diagramma del Sistema di Routing



Chiamare l'API REST

Sintassi generale:

`http://localhost/index.php/{MODULE}/{METHOD}?{PARAMS}`

Esempio pratico:

```
1 http://localhost/index.php/user/list?limit=20
```

Parametri disponibili:

- `limit`: Numero massimo di utenti da recuperare
- Default: 10 utenti se non specificato

Testing Tools

Utilizzare strumenti come:

- Browser web (per GET)

Formato della Risposta JSON

Risposta di successo:

```
[
    {
        "user_id": "1",
        "username": "Gina",
        "user_email": "ginaverdi@gmail.com",
        "user_status": "0"
    },
    {
        "user_id": "2",
        "username": "Mario",
        "user_email": "mario.rossi@email.com",
        "user_status": "1"
    },
    {
        "user_id": "3",
        "username": "Laura",
        "user_email": "laura.bianchi@email.com",
        "user_status": "1"
    }
]
```

Headers della risposta:

- `Content-Type: application/json`
- `HTTP/1.1 200 OK`

Gestione degli Errori

Errore metodo non supportato (422):

```
{
    "error": "Method not supported"
}
```

Errore interno del server (500):

```
{
    "error": "Something went wrong!
              Please contact support."
}
```

Risorsa non trovata (404):

```
HTTP/1.1 404 Not Found
```

Ogni tipo di errore restituisce il codice HTTP appropriato e un messaggio descrittivo in formato JSON.

Testing con cURL

GET Request - Recupera 5 utenti:

```
curl -X GET "http://localhost/index.php/user/list?limit=5"
```

GET Request con headers visibili:

```
curl -i -X GET "http://localhost/index.php/user/list?limit=10"
```

Test metodo non supportato:

```
curl -X POST "http://localhost/index.php/user/list"
```

Opzioni cURL utili

- `-X`: Specifica il metodo HTTP
- `-i`: Mostra gli headers della risposta
- `-v`: Modalità verbose per debugging

Best Practices per REST API

Design dell'API:

- ✓ Utilizzare nomi plurali per le risorse
- ✓ Versioning dell'API (v1, v2)
- ✓ Paginazione per grandi dataset
- ✓ Filtri e ordinamento
- ✓ HATEOAS (Hypermedia)

Codice:

- ⟨⟩ Separazione delle responsabilità
- ⟨⟩ DRY (Don't Repeat Yourself)
- ⟨⟩ Dependency Injection
- ⟨⟩ Logging e monitoring

Sicurezza:

- △ Autenticazione (JWT, OAuth)
- △ Validazione input
- △ HTTPS obbligatorio
- △ Rate limiting
- △ CORS policy

Prestazioni:

- ≫ Caching
- ≫ Compressione GZIP
- ≫ Query ottimizzate
- ≫ CDN per asset statici

Protezione da SQL Injection

Codice VULNERABILE:

```php
// NON FARE MAI COSÌ!
$query = "SELECT * FROM users
          WHERE user_id = "
          . $_GET['id'];
$result = mysqli_query(
    $conn, $query
);
```

Codice SICURO:

```php
// Prepared Statement
$stmt = $conn->prepare(
    "SELECT * FROM users
     WHERE user_id = ?"
);
$stmt->bind_param("i", $id);
$stmt->execute();
```

! Attacco possibile:

```
?id=1 OR 1=1
?id=1; DROP TABLE users;
```

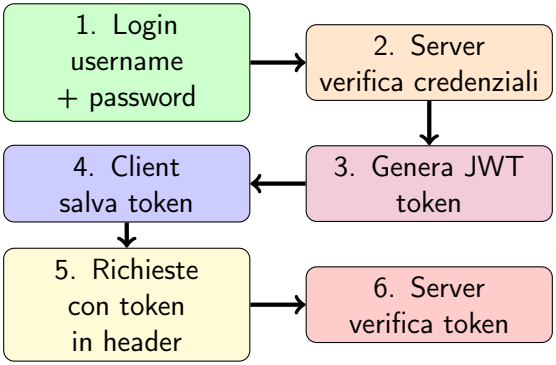
✓ Protetto!

- Parametri separati dalla query
- Type checking automatico
- Escape automatico

Regola d'Oro

Mai concatenare direttamente input utente nelle query SQL!

Autenticazione con JWT



JWT (JSON Web Token): Standard aperto (RFC 7519) per trasmettere informazioni in modo sicuro tra parti come oggetto JSON.

Validazione e Sanitizzazione Input

```php
// Validazione del parametro limit
public function listAction()
{
    $arrQueryStringParams = $this->getQueryStringParams();

    // Valore di default
    $intLimit = 10;

    // Validazione
    if (isset($arrQueryStringParams['limit'])) {
        $limit = $arrQueryStringParams['limit'];

        // Verifica che sia un numero
        if (is_numeric($limit) && $limit > 0 && $limit <= 100) {
            $intLimit = (int)$limit;
        } else {
            // Gestione errore
            $this->sendOutput(
                json_encode(['error' => 'Invalid limit parameter']),
                ['Content-Type: application/json',
                 'HTTP/1.1 400 Bad Request']
            );
```

Gestione CORS (Cross-Origin Resource Sharing)

```php
// Aggiungere in BaseController::sendOutput()

protected function sendOutput($data, $httpHeaders = array())
{
    // Headers CORS
    header('Access-Control-Allow-Origin: *');
    header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE');
    header('Access-Control-Allow-Headers: Content-Type, Authorization');
    header('Access-Control-Max-Age: 3600');

    // Gestione preflight request
    if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
        http_response_code(200);
        exit();
    }

    header_remove('Set-Cookie');

    if (is_array($httpHeaders) && count($httpHeaders)) {
        foreach ($httpHeaders as $httpHeader) {
            header($httpHeader);
        }
```

Possibili Estensioni dell'API

Nuovi Endpoint:

- `POST /user` - Crea utente
- `GET /user/{id}` - Dettagli utente
- `PUT /user/{id}` - Aggiorna utente
- `DELETE /user/{id}` - Elimina utente
- `GET /user/search` - Ricerca utenti

Funzionalità:

- Paginazione avanzata
- Filtri multipli
- Ordinamento personalizzato
- Export in CSV/PDF

Sicurezza:

- Sistema di autenticazione
- Gestione ruoli e permessi
- Rate limiting
- API Key management
- Audit logging

Performance:

- Redis caching
- Query optimization
- Connection pooling
- Async processing

Implementazione Paginazione

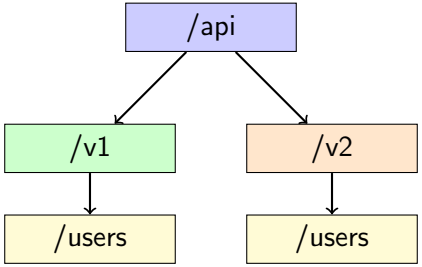
```php
// In UserModel.php
public function getUsers($limit, $offset = 0)
{
    return $this->select(
        "SELECT * FROM users
         ORDER BY user_id ASC
         LIMIT ? OFFSET ?",
        ["ii", $limit, $offset]
    );
}

public function getTotalUsers()
{
    $result = $this->select("SELECT COUNT(*) as total FROM users");
    return $result[0]['total'];
}

// In UserController.php
public function listAction()
{
    $page = isset($_GET['page']) ? (int)$_GET['page'] : 1;
    $limit = isset($_GET['limit']) ? (int)$_GET['limit'] : 10;
    $offset = ($page - 1) * $limit;

    $userModel = new UserModel();
    $users = $userModel->getUsers($limit, $offset);
    $total = $userModel->getTotalUsers();

    $response = [
        'data' => $users,
```

Versioning dell'API

Strategie di versioning:

1. **URI Path:** `/api/v1/users`, `/api/v2/users`
2. **Query String:** `/api/users?version=1`
3. **Header:** `Accept: application/vnd.api.v1+json`

Esempio struttura con versioning:



Documentazione dell'API

Strumenti per documentare REST API:

OpenAPI/Swagger:

- Specifica standard
- UI interattiva
- Generazione client SDK
- Testing integrato

Postman:

- Collezioni condivisibili
- Test automatizzati
- Mock server
- Documentazione generata

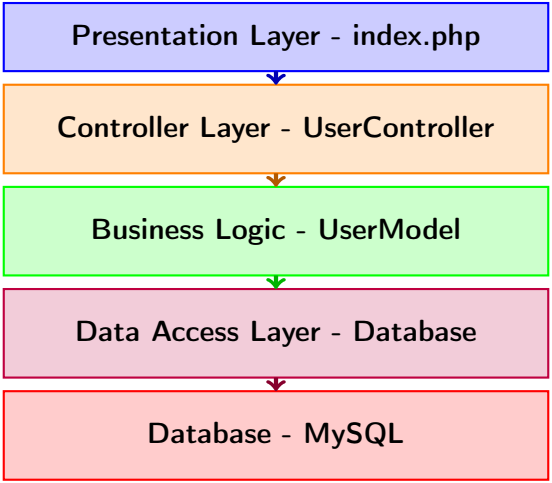
Elementi essenziali:

- Endpoint disponibili
- Metodi HTTP supportati
- Parametri richiesti/opzionali
- Formato richiesta/risposta
- Codici di stato
- Esempi di utilizzo
- Autenticazione
- Rate limits

PHPUnit - Test per UserModel:

```php
<?php
use PHPUnit\Framework\TestCase;

class UserModelTest extends TestCase
{
    private $userModel;

    protected function setUp(): void
    {
        $this->userModel = new UserModel();
    }

    public function testGetUsersReturnsArray()
    {
        $users = $this->userModel->getUsers(10);
        $this->assertIsArray($users);
    }

    public function testGetUsersRespectsLimit()
    {
        $limit = 5;
        $users = $this->userModel->getUsers($limit);
        $this->assertLessThanOrEqual($limit, count($users));
    }

    public function testUserStructure()
    {
        $users = $this->userModel->getUsers(1);
```

Riepilogo dell'Architettura



Separazione dei livelli garantisce:

Vantaggi dell'Approccio REST

Tecnici:

- ✓ Architettura scalabile
- ✓ Stateless (facilita load balancing)
- ✓ Caching ottimizzato
- ✓ Separazione client/server
- ✓ Standard HTTP universale

Di Business:

- $ Riduzione costi di sviluppo
- $ Time-to-market più rapido
- $ Manutenzione semplificata

Per gli Sviluppatori:

- U Codice più leggibile
- U Testing facilitato
- U Collaborazione migliorata
- U Documentazione standardizzata

Per gli Utenti:

- ♡ Performance migliori
- ♡ Esperienza multi-dispositivo
- ♡ Affidabilità superiore

Punti Chiave del Tutorial

1. **REST API** sono lo standard de facto per servizi web moderni
2. **PHP 8** offre strumenti potenti per implementare API robuste
3. **Pattern MVC** separa responsabilità e migliora manutenibilità
4. **Prepared Statements** proteggono da SQL Injection
5. **Routing URI** permette endpoint puliti e RESTful
6. **Gestione errori** fornisce feedback chiari ai client
7. **JSON** è il formato standard per lo scambio dati
8. **Estensibilità** è garantita dalla struttura modulare

Ricorda

Questo è solo l'inizio! Le REST API possono essere molto più complesse e potenti.

Risorse per Approfondire

Documentazione Ufficiale:

- PHP: `https://www.php.net/manual/`
- MySQL: `https://dev.mysql.com/doc/`
- REST API: `https://restfulapi.net/`

Framework PHP per API:

- Laravel (con Passport/Sanctum)
- Symfony (con API Platform)
- Slim Framework
- Lumen

Strumenti Utili:

- Postman - Testing API
- Swagger/OpenAPI - Documentazione
- Docker - Containerizzazione
- Git - Version Control

Esercizi Proposti

Livello Base:

1. Aggiungere un endpoint per recuperare un singolo utente per ID
2. Implementare filtri per user_status nella lista utenti
3. Aggiungere ordinamento personalizzato (ASC/DESC)

Livello Intermedio:

1. Implementare endpoint POST per creare nuovi utenti
2. Aggiungere validazione email con regex
3. Implementare sistema di paginazione completo

Livello Avanzato:

1. Implementare autenticazione JWT
2. Creare sistema di rate limiting
3. Aggiungere logging delle richieste
4. Implementare cache con Redis

Domande?

@ prof@example.com

Git github.com/username

Grazie per l'attenzione!