

REST API in PHP 8

Sviluppo di Servizi Web Moderni con MySQL

Prof. Massimo

IIS Fermi Sacconi Ceci - Ascoli Piceno

December 30, 2025

Indice

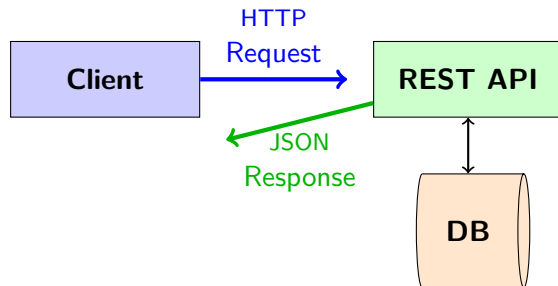
- 1 Introduzione alle REST API
- 2 Architettura del Progetto
- 3 Configurazione Database
- 4 Implementazione Models
- 5 Implementazione Controllers
- 6 Entry Point e Routing
- 7 Testing e Utilizzo
- 8 Best Practices e Sicurezza
- 9 Estensioni e Miglioramenti
- 10 Conclusioni

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



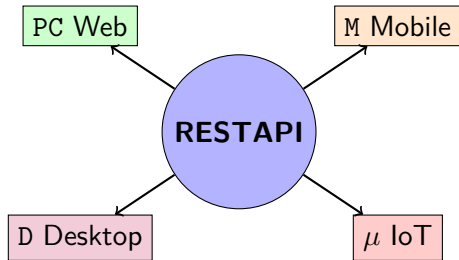
Vantaggi delle REST API

Per gli sviluppatori:

- $\langle \rangle$ Facilità di implementazione
- $[]$ Separazione front-end/back-end
- \circlearrowleft Riutilizzabilità del codice
- \star Supporto di framework moderni

Per le applicazioni:

- \leftrightarrow Scalabilità
- \triangle 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

CRUD Operations

- **Create** → POST
- **Read** → GET
- **Update** → PUT/PATCH
- **Delete** → 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

```
1 rest-api-php/  
2 |-- index.php  
3 |-- inc/  
4 |   |-- config.php  
5 |   |-- bootstrap.php  
6 |-- Model/  
7 |   |-- Database.php  
8 |   |-- UserModel.php  
9 |-- Controller/  
   |-- Api/  
       |-- BaseController.php  
       |-- UserController.php
```

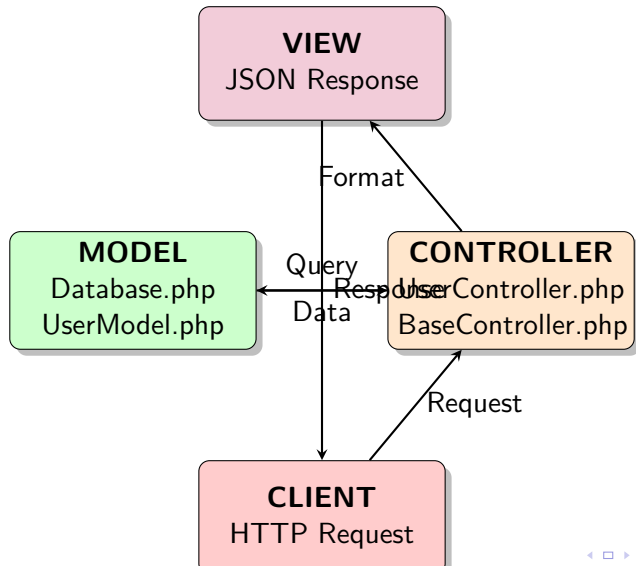
Componenti principali:

- **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

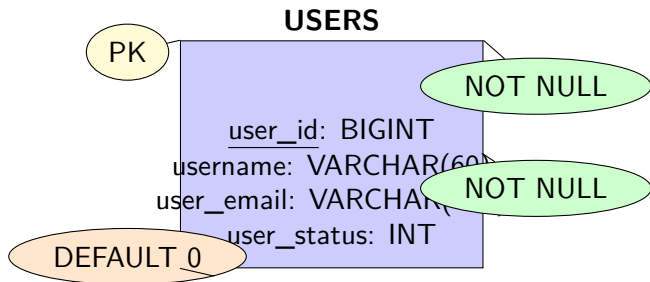
Inserimento dati di esempio:

```
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

```
1 <?php
2 // inc/config.php
3
4 define("DB_HOST", "localhost");
5 define("DB_USERNAME", "root");
6 define("DB_PASSWORD", "");
7 define("DB_DATABASE_NAME", "rest_api_php");
8 ?>
```

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

```
1 <?php
2 // inc/bootstrap.php
3
4 define("PROJECT_ROOT_PATH", __DIR__ . "../");
5
6 // Include main configuration file
7 require_once PROJECT_ROOT_PATH . "/inc/config.php";
8
9 // Include the base controller file
10 require_once PROJECT_ROOT_PATH .
11     "/Controller/Api/BaseController.php";
12
13 // Include the user model file
14 require_once PROJECT_ROOT_PATH .
15     "/Model/UserModel.php";
16 ?>
```

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

Classe Database - Parte 1

```
1 <?php
2 // Model/Database.php
3
4 class Database
5 {
6     protected $connection = null;
7
8     public function __construct()
9     {
10         try {
11             $this->connection = new mysqli(
12                 DB_HOST,
13                 DB_USERNAME,
14                 DB_PASSWORD,
15                 DB_DATABASE_NAME
16             );
17
18             if (mysqli_connect_errno()) {
19                 throw new Exception(
20                     "Could not connect to database."
21                 );
22             }
23         }
24     }
25 }
```

Classe Database - Parte 2

```
1 public function select($query = "", $params = [])
2 {
3     try {
4         $stmt = $this->executeStatement($query, $params);
5         $result = $stmt->get_result()
6                 ->fetch_all(MYSQLI_ASSOC);
7         $stmt->close();
8         return $result;
9     } catch (Exception $e) {
10        throw new Exception($e->getMessage());
11    }
12 }
13
14 private function executeStatement($query = "", $params = [])
15 {
16     try {
17         $stmt = $this->connection->prepare($query);
18         if ($stmt === false) {
19             throw new Exception(
20                 "Unable to prepare statement: " . $query
21             );
22         }
23     }
```

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

```
1 <?php
2 // Model/UserModel.php
3
4 require_once PROJECT_ROOT_PATH . "/Model/Database.php";
5
6 class UserModel extends Database
7 {
8     public function getUsers($limit)
9     {
10         return $this->select(
11             "SELECT * FROM users
12             ORDER BY user_id ASC
13             LIMIT ?",
14             ["i", $limit]
15         );
16     }
17 }
18 ?>
```

Classe BaseController - Parte 1

```
1 <?php
2 // Controller/Api/BaseController.php
3
4 class BaseController
5 {
6     /**
7      * Magic method per gestire chiamate a metodi inesistenti
8      */
9     public function __call($name, $arguments)
10     {
11         $this->sendOutput('',
12             array('HTTP/1.1 404 Not Found'));
13     }
14
15     /**
16      * Estrae i segmenti dall'URI
17      */
18     protected function getUriSegments()
19     {
20         $uri = parse_url($_SERVER['REQUEST_URI'],
21             PHP_URL_PATH);
22         $uri = explode('/', $uri);
```

Classe BaseController - Parte 2

```
1  /**
2   * Recupera parametri della query string
3   */
4  protected function getQueryStringParams()
5  {
6      parse_str($_SERVER['QUERY_STRING'], $query);
7      return $query;
8  }
9
10 /**
11  * Invia l'output JSON al client
12  */
13 protected function sendOutput($data,
14                               $httpHeaders = array())
15 {
16     header_remove('Set-Cookie');
17
18     if (is_array($httpHeaders) && count($httpHeaders)) {
19         foreach ($httpHeaders as $httpHeader) {
20             header($httpHeader);
21         }
22     }
```

Metodi della Classe BaseController

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

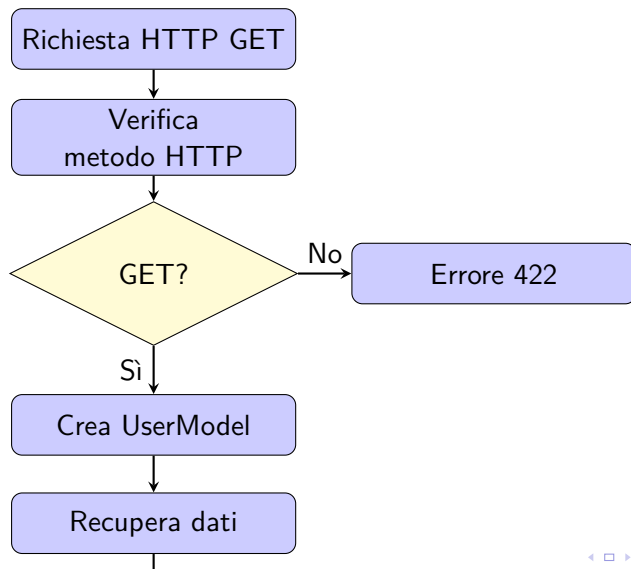
Classe UserController - Parte 1

```
1 <?php
2 // Controller/Api/UserController.php
3
4 class UserController extends BaseController
5 {
6     /**
7      * Endpoint: /user/list
8      * Restituisce l'elenco degli utenti
9      */
10    public function listAction()
11    {
12        $strErrorDesc = '';
13        $requestMethod = $_SERVER["REQUEST_METHOD"];
14        $arrQueryStringParams = $this->getQueryStringParams();
15
16        if (strtoupper($requestMethod) == 'GET') {
17            try {
18                $userModel = new UserModel();
19                $intLimit = 10;
20
21                if (isset($arrQueryStringParams['limit']) &&
22                    $arrQueryStringParams['limit']) {
23                    $intLimit = $arrQueryStringParams['limit'];
24                }
25
26                $arrUsers = $userModel->getUsers($intLimit);
27                $responseData = json_encode($arrUsers);
```

Classe UserController - Parte 2

```
1      } catch (Error $e) {
2          $strErrorDesc = $e->getMessage() .
3              'Something went wrong! Please contact support.';
4          $strErrorHeader = 'HTTP/1.1 500 Internal Server Error';
5      }
6  } else {
7      $strErrorDesc = 'Method not supported';
8      $strErrorHeader = 'HTTP/1.1 422 Unprocessable Entity';
9  }
10
11  // Invia output
12  if (!$strErrorDesc) {
13      $this->sendOutput(
14          $responseData,
15          array('Content-Type: application/json',
16              'HTTP/1.1 200 OK')
17      );
18  } else {
19      $this->sendOutput(
20          json_encode(array('error' => $strErrorDesc)),
21          array('Content-Type: application/json',
22              $strErrorHeader)
23      );
24  }
25 }
26 }
```

Flusso di Esecuzione in UserController



Entry Point - index.php

```
1 <?php
2 // index.php
3
4 require __DIR__ . "/inc/bootstrap.php";
5
6 $uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
7 $uri = explode('/', $uri);
8
9 // Validazione URI
10 if ((isset($uri[2]) && $uri[2] != 'user') ||
11     !isset($uri[3])) {
12     header("HTTP/1.1 404 Not Found");
13     exit();
14 }
15
16 // Carica il controller appropriato
17 require PROJECT_ROOT_PATH .
18     "/Controller/Api/UserController.php";
19
20 $objFeedController = new UserController();
21 $strMethodName = $uri[3] . 'Action';
22 $objFeedController->{$strMethodName}();
```


Analisi del Routing URI

Struttura URI:

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

Componenti URI:

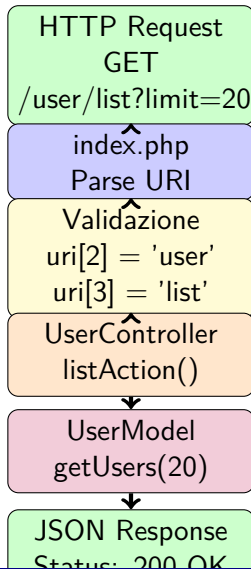
- `$uri[0]` → `"` (vuoto)
- `$uri[1]` → `'index.php'`
- `$uri[2]` → `'user'` (modulo)
- `$uri[3]` → `'list'` (metodo)

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:

```
1 [
2   {
3     "user_id": "1",
4     "username": "Gina",
5     "user_email": "ginaverdi@gmail.com",
6     "user_status": "0"
7   },
8   {
9     "user_id": "2",
10    "username": "Mario",
11    "user_email": "mario.rossi@email.com",
12    "user_status": "1"
13  },
14  {
15    "user_id": "3",
16    "username": "Laura",
17    "user_email": "laura.bianchi@email.com",
18    "user_status": "1"
19  }
20 ]
```

Headers della risposta:

- Content-Type: application/json

- HTTP/1.1 200 OK

Gestione degli Errori

Errore metodo non supportato (422):

```
1 {  
2     "error": "Method not supported"  
3 }
```

Errore interno del server (500):

```
1 {  
2     "error": "Something went wrong!  
3             Please contact support."  
4 }
```

Risorsa non trovata (404):

```
1 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:

- $\langle \rangle$ Separazione delle responsabilità
- $\langle \rangle$ DRY (Don't Repeat Yourself)
- $\langle \rangle$ Dependency Injection
- $\langle \rangle$ Logging e monitoring

Sicurezza:

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

Prestazioni:

- \gg Caching
- \gg Compressione GZIP
- \gg Query ottimizzate
- \gg CDN per asset statici

Protezione da SQL Injection

Codice VULNERABILE:

```
1 // NON FARE MAI COSÌ!  
2 $query = "SELECT * FROM users  
3         WHERE user_id = "  
4         . $_GET['id'];  
5 $result = mysqli_query(  
6     $conn, $query  
7 );
```

! Attacco possibile:

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

Codice SICURO:

```
1 // Prepared Statement  
2 $stmt = $conn->prepare(  
3     "SELECT * FROM users  
4     WHERE user_id = ?"  
5 );  
6 $stmt->bind_param("i", $id);  
7 $stmt->execute();
```

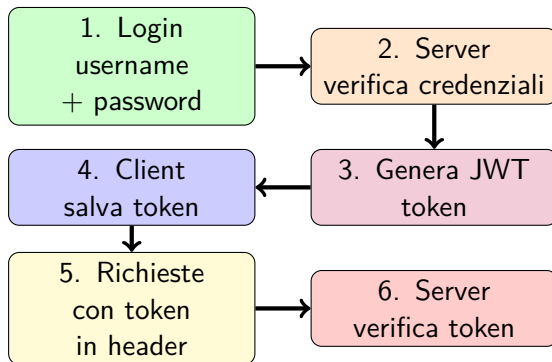
✓ Protetto!

- 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

```
1 // Validazione del parametro limit
2 public function listAction()
3 {
4     $arrQueryStringParams = $this->getQueryStringParams();
5
6     // Valore di default
7     $intLimit = 10;
8
9     // Validazione
10    if (isset($arrQueryStringParams['limit'])) {
11        $limit = $arrQueryStringParams['limit'];
12
13        // Verifica che sia un numero
14        if (is_numeric($limit) && $limit > 0 && $limit <= 100) {
15            $intLimit = (int)$limit;
16        } else {
17            // Gestione errore
18            $this->sendOutput(
19                json_encode(['error' => 'Invalid limit parameter']),
20                ['Content-Type: application/json',
21                 'HTTP/1.1 400 Bad Request']
22            );
23        }
24    }
25 }
```

Gestione CORS (Cross-Origin Resource Sharing)

```
1 // Aggiungere in BaseController::sendOutput()
2
3 protected function sendOutput($data, $httpHeaders = array())
4 {
5     // Headers CORS
6     header('Access-Control-Allow-Origin: *');
7     header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE');
8     header('Access-Control-Allow-Headers: Content-Type, Authorization');
9     header('Access-Control-Max-Age: 3600');
10
11     // Gestione preflight request
12     if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
13         http_response_code(200);
14         exit();
15     }
16
17     header_remove('Set-Cookie');
18
19     if (is_array($httpHeaders) && count($httpHeaders)) {
20         foreach ($httpHeaders as $httpHeader) {
21             header($httpHeader);
22         }
23     }
```

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

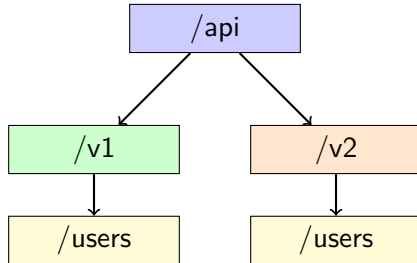
```
1 // In UserModel.php
2 public function getUsers($limit, $offset = 0)
3 {
4     return $this->select(
5         "SELECT * FROM users
6         ORDER BY user_id ASC
7         LIMIT ? OFFSET ?",
8         ["ii", $limit, $offset]
9     );
10 }
11
12 public function getTotalUsers()
13 {
14     $result = $this->select("SELECT COUNT(*) as total FROM users");
15     return $result[0]['total'];
16 }
17
18 // In UserController.php
19 public function listAction()
20 {
21     $page = isset($_GET['page']) ? (int)$_GET['page'] : 1;
22     $limit = isset($_GET['limit']) ? (int)$_GET['limit'] : 10;
23     $offset = ($page - 1) * $limit;
24
25     $userModel = new UserModel();
26     $users = $userModel->getUsers($limit, $offset);
27     $total = $userModel->getTotalUsers();
28
29     $response = [
30         '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:

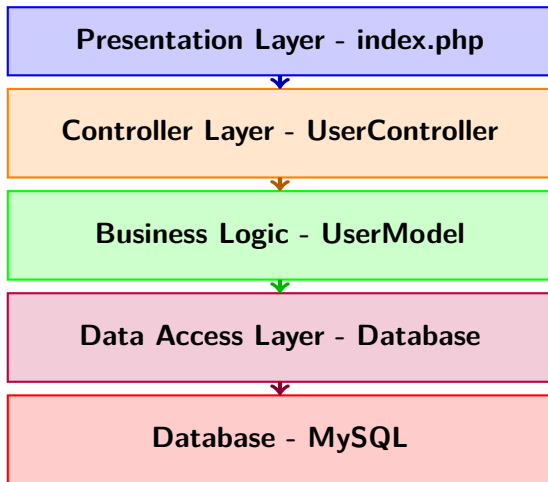
- Endpoint disponibili
- Metodi HTTP supportati
- Parametri richiesti/opzionali
- Formato richiesta/risposta
- Codici di stato
- Esempi di utilizzo
- Autenticazione
- Rate limits

Testing Automatizzato

PHPUnit - Test per UserModel:

```
1 <?php
2 use PHPUnit\Framework\TestCase;
3
4 class UserModelTest extends TestCase
5 {
6     private $userModel;
7
8     protected function setUp(): void
9     {
10         $this->userModel = new UserModel();
11     }
12
13     public function testGetUsersReturnsArray()
14     {
15         $users = $this->userModel->getUsers(10);
16         $this->assertIsArray($users);
17     }
18
19     public function testGetUsersRespectsLimit()
20     {
21         $limit = 5;
22         $users = $this->userModel->getUsers($limit);
23         $this->assertLessThanOrEqual($limit, count($users));
24     }
25
26     public function testUserStructure()
27     {
28         $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.

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!

