International Journal of Information Technology Science (INTENS)
Vol. 01, No. 01, January 2026, pp. 18-30

REST API Implementation for Employee Attendance System Based on Client–Server Architecture Using Laravel Sanctum

Rizky Parlika[1], Febri Aldi[2], Dimas Galih Giovana[3]
[1]Universitas Pembangunan Nasional Veteran Jawa Timur, Surabaya, Indonesia
[2]Universitas Putra Indonesia YPTK, Padang, Indonesia
[3]Universitas Pembangunan Nasional Veteran Jawa Timur, Surabaya, Indonesia
[1]rizkyparlika.if@upnjatim.ac.id, [2]febri_aldi@upiyptk.ac.id, [3]223081010169@student.upnjatim.ac.id

Article Info

Article history:

Received January 7, 2026
Revised January 15, 2026
Accepted January 16, 2026

Keywords:

Laravel
Sanctum
REST API
Employee Attendance
HTTP

ABSTRACT

REST API has become a popular approach in modern system development due to its lightweight, flexible nature, and easy integration with various platforms. This study implements a REST API for an employee attendance system using the Laravel framework and Sanctum token authentication for access security. The system provides registration, login, clock-in and clock-out recording, attendance history, and daily and monthly reports that can only be accessed by administrators. Testing was conducted using Postman to ensure the validity of HTTP responses, JSON structures, and token-based access security. The results of the study show that all endpoints function well and are able to handle business validation appropriately. This system can be the foundation for web-based and mobile attendance applications in the future.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Febri Aldi

1. INTRODUCTION

The development of information technology has brought significant changes in the way organizations manage data and operational processes. One of the important processes in the company is the recording of employee attendance which serves as the basis for calculating productivity, performance evaluation, and personnel administration. Manual or local input-based attendance systems often cause various problems such as recording errors, time manipulation, recapitulation delays, and lack of integration with other systems. This condition requires a modern, automated, and accessible attendance system through various application platforms.

REST API (Representational State Transfer Application Programming Interface) is one of the approaches widely used in modern software development because it is lightweight, flexible, and easy to integrate with various frontend applications such as mobile, web, and desktop[7], [8]. The use of REST APIs allows the attendance process to be carried out in a structured manner with clear separation between the server and the client, making the system more modular, easy to develop, and supporting long-term scalability. [9] The Laravel framework was chosen as the API development platform because it provides a wide range of built-in features such as routing, middleware, Eloquent ORM, as well as the Laravel Sanctum authentication system that supports token-based security.

Based on these conditions, this research was conducted to design and implement the REST API employee attendance system by providing key features such as registration, user authentication, attendance time recording (clock-in and clock-out), attendance history storage, and the provision of daily and monthly reports that can only be accessed by administrators. By utilizing Laravel-based software development methods,

this research aims to produce backend services that can be used without an interface (headless) or integrated with frontend applications in the future.

2. METHOD

The methodology of this research includes a series of systematic stages carried out to design, develop, and test a REST API-based attendance system using Laravel. The research stages are organized into several subchapters which include literature study, needs analysis, system design, database design, API implementation, and system testing. This method is used to ensure that the system developed is in accordance with REST principles, safe, and can function optimally.

2.1. Studi Literature

The first stage is to conduct a literature study to understand the basic concepts underlying this research. The REST API is a style of architecture introduced by Fielding[1], which relies on the principles of resource-oriented and HTTP-based communication [2], [3]. Other literature also explains that REST-based APIs generally use JSON as the primary data exchange format due to their lightweight and easy-to-process nature [10].

In addition, a literature study on client-server architecture is also used as the basis for how APIs function as service providers (servers) and clients as parties accessing the service [11], [12]. Additional studies were conducted on the Laravel framework, which provides routing, middleware, Laravel Sanctum token authentication, and Eloquent ORM features to facilitate API implementation [13], [14], [15]. The method of token authentication is also understood from the literature related to the security of controlled access-based systems [16]. This literature study is the main foundation in designing and building a REST API-based attendance system.

2.2. Needs Analysis

Needs analysis is carried out to determine the specifications of the attendance system to be developed. Functional needs include the ability of the system to: 1) Perform user registration. 2) Log in using token authentication. 3) Clock-in and clock-out. 4) Display employee attendance history. 5) Generate daily and monthly reports for admins.

Non-functional needs include security aspects, API response consistency, HTTP code state standards [17], as well as the system's ability to integrate with other applications via JSON format [10]. Modern system development principles such as modularity and scalability are also considered [18], [19]. This specification of needs is formulated based on the results of the analysis of employee attendance business processes and refers to the practice of developing REST APIs that have been discussed in the previous literature [3], [4], [20].

2.3. System Planning

The design of the system aims to describe the architecture as well as the interaction flow of the attendance system. The REST API architecture is designed with a client–server model, where the client sends HTTP requests and the server processes the business logic and returns a JSON response [12], [21]. The endpoint structure is structured according to the naming rules of the REST standard to be easy to understand and consistent[2], [3].

The use case diagram is compiled to illustrate the roles of employees and admins. Employees have access to login, clock-in, clock-out, and attendance history features, while admins have additional access in the form of daily reports and monthly reports. Sequence diagrams are created to describe the flow of the clock-in process, from client requests, middleware validation, to data storage in databases. The principles of API design follow the best practice guidelines recommended by Google and AWS, namely endpoint consistency, uniform response structure, and clear error handling [6], [12].

2.3.1. Use Case Diagram

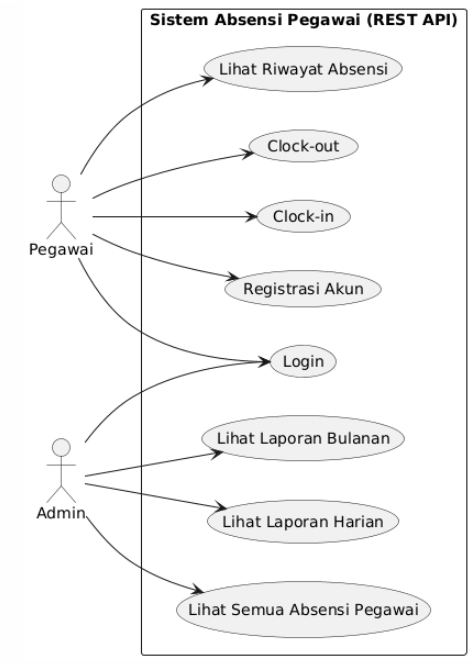


Figure 1. Use Case Diagram

The use case diagram illustrates the relationship between actors and features available in the attendance system. There are two main actors, namely Employees and Admins, each with different access rights. Employees can only carry out processes related to personal needs, such as registration, login, clock-in, clock-out, and viewing attendance history. Admins have access to all employee features plus the ability to view all employee attendance data and generate daily and monthly reports.

This distribution of access rights is an implementation of role-based access control (RBAC) which aims to maintain system security and data integrity. With RBAC, the system can ensure that only users with admin roles can access sensitive administrative functions. This use case structure is commonly used in employee management systems and supports the least privilege principle recommended in API security [16], [22].

2.3.2. System Architecture Diagram

Note in 'Figure 3.2', the system architecture diagram illustrates how the key components are interconnected in the employee attendance REST API implementation. On the client layer, users can use Postman, a web application, or a mobile application to send requests to the server. Communication is carried out via the HTTP protocol with a JSON-formatted payload, according to the REST API standard [17], [10], [23].

The server is built using the Laravel framework which serves as the central business logic. It contains several important components, such as routing to handle endpoint addresses, controllers to manage business processes, middleware for token authentication and authorization, and the Eloquent model as a link between applications and databases [13], [15]. When the server receives a request, it is processed by the controller after passing through the middleware that verifies the authentication token. If valid, the controller performs an operation against the database.

MySQL databases are used to store user and attendance data. Through Eloquent's ORM, all data storage, update, and retrieval operations are carried out in a structured and secure manner. This architecture implements the client–server pattern commonly used in modern distributed systems and ensures separation of responsibilities as recommended in the REST architecture [1], [3], [12].

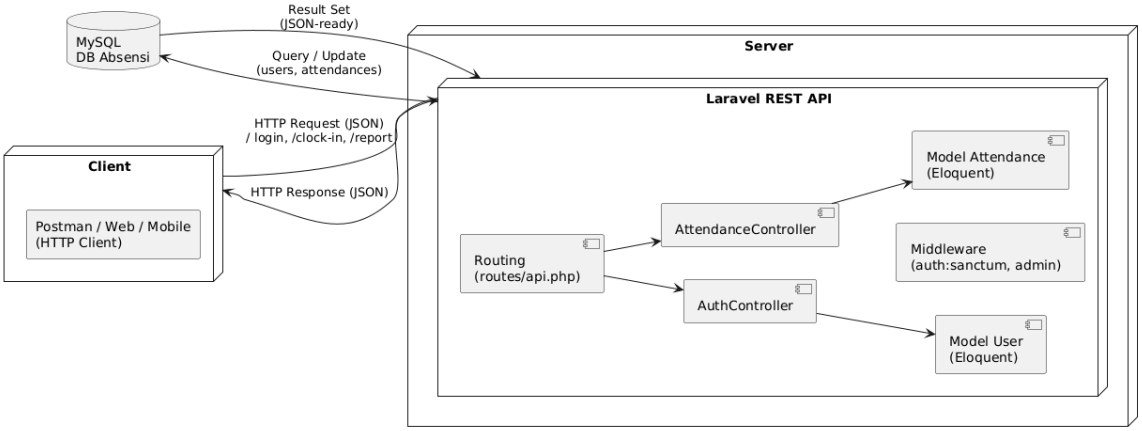


Figure 2. System Architecture Diagram

2.3.3. Sequence Diagram Clock-in & Clock-out

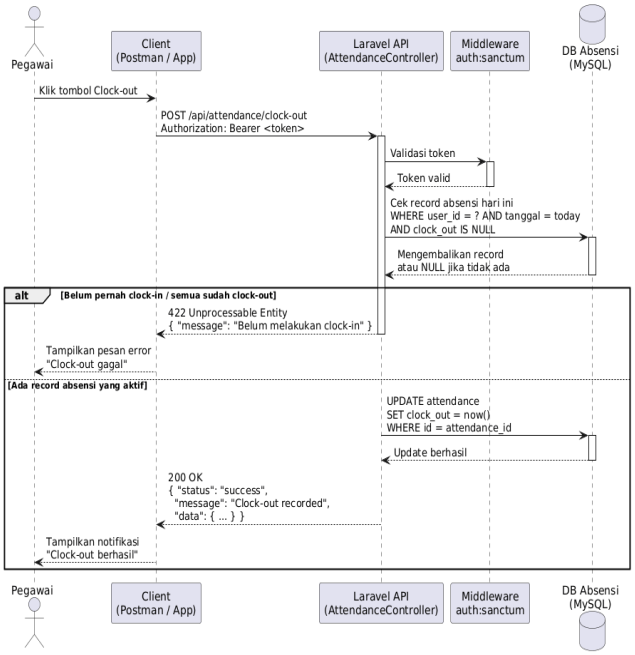


Figure 3. Sequence Diagram Clock-out

Note in 'Figure 3.3', the sequence clock-out diagram depicts the process when employees are absent. The client sends the request to the /attendance/clock-out endpoint, and the request is re-examined through authentication middleware to ensure the token is valid.

The server then checks if there are any active clock-in records for the day. If there is no clock-in record or the user has clocked out before, the system returns an error 422 response to inform that the outgoing attendance process cannot be performed. This ensures attendance flows remain logical and prevents inconsistent data.

If the clock-in record is valid and does not have a clock-out yet, the system will update the data by adding a clock-out time to the attendance entry. The server then sends a JSON response with a success status. This process reflects a strict flow control mechanism and follows REST principles in resource management[2], [3].

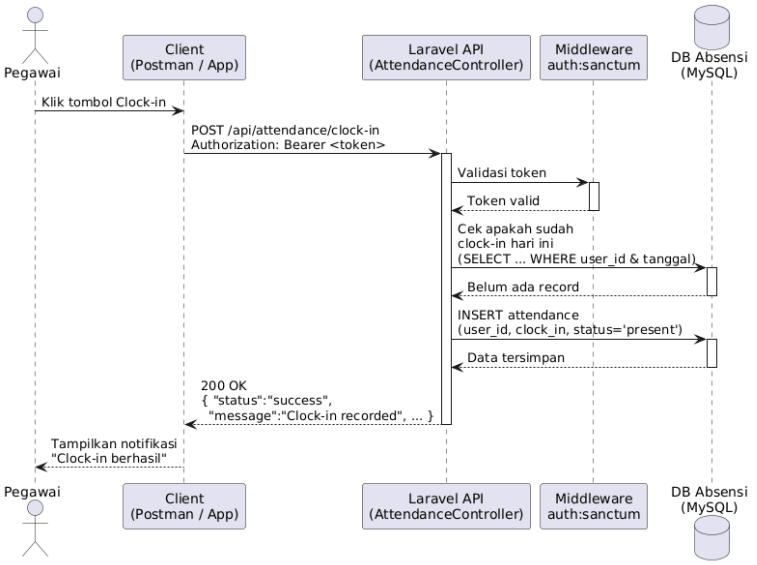


Figure 4. Sequence Diagram Clock-in

Notice in 'Figure 3.4', the clock-in sequence diagram describes the flow of stages that occur when the user is logged in. The process begins when the employee presses the clock-in button on the client. The client then sends an HTTP POST request to the /attendance/clock-in endpoint, accompanied by a Bearer authentication token issued at login.

The request first goes through the auth:sanctum middleware that verifies the token. If the token is valid, the system then checks the database to make sure that the employee has not been clocked in on the same day. This check is important as part of business logic validation to prevent duplication of attendance, as per the business rule enforcement principles in modern APIs [20].

If there is no clock-in record for the day, the server creates a new attendance entry in the database. After that, the server returns a JSON response containing the success status and the clock-in time. If the employee has clocked in before, the system returns a 422 Unprocessable Entity response as a form of rejection that complies with the HTTP standard [17].

2.4. Database Design

The database is designed using a relational model with two main entities, namely users and attendances. This design follows the principle of a relational database that emphasizes integrity, normalization, and relationships between entities through primary keys and foreign keys [24]. The user entity stores information such as name, email, encrypted password, and role (employee/admin). The attendance entity stores records the clock-in time, clock-out, attendance status, and relationship to the user. The relationship between the two entities follows a one-to-many pattern, where a single user can have multiple attendance records. ERD is used to describe relationships between entities and is the basis for creating Laravel migrations using the Eloquent ORM [13]. With the design of this database, the system can ensure the consistency of attendance data and the ease of querying reports.

2.4.1. Entity Relationship Diagram (ERD)

Notice in 'Figure 4.1', the Entity Relationship Diagram (ERD) depicts the structure of the database relationally. In this system, there are two main entities, namely User and Attendance. The User entity stores user identity data, including name, email, encrypted password, and role. Meanwhile, the Attendance entity keeps attendance records such as clock-in time, clock-out, and attendance status. The relationship between the two is one-to-many, where a single user can have multiple attendance entries. This relationship is manifested through the foreign key 'user_id' in the 'attendances' table which refers to the primary key 'id' in the 'users' table. This structure is commonly used in attendance management systems because it allows for detailed attendance recording for each user [24].

This ERD design supports the ease of creating reports, both daily and monthly reports, because attendance data can be filtered by date and user. In addition, this database structure minimizes redundancy and ensures data consistency through the application of database normalization principles.

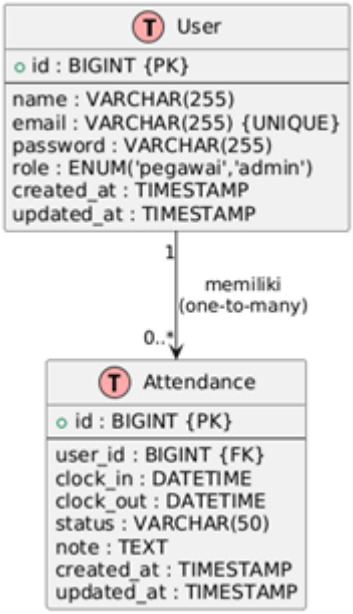


Figure 5. Entity Relationship Diagram

2.4.2. Database Structure

The database structure in this system is designed with ease of integration, data consistency, and query efficiency in mind. The users table is used to store user information while also defining access rights through role attributes. The attendances table stores all user attendance activities and has time attributes that allow for date-based analysis.

With a design like this, the system can validate attendance, generate attendance history, and create reports with specific filters. The use of data types such as DATETIME allows for high-precision time storage. Meanwhile, foreign key user_id ensure that each attendance data is always associated with the correct user, thus avoiding orphan records.

The design of this database also supports CRUD operations carried out through APIs and makes it easier for Eloquent ORM to conduct relationships between entities. As such, the structure of this database meets the needs of modern attendance applications that are modular and easy to develop. The following is the table of users and the table of attendance.

Structure Table User

The users table consists of 7 attributes, namely 'Id', 'name', 'email', 'password', 'role', created_at', and 'updated_at'. We use this table to store data from the account that will be used at the time of creation (registration) and will be taken to be used at the time of login.

Table 1. Structure Table User

Nama Kolom	Tipe Data	Keterangan
Id	BIGINT (PK, AUTO_INCREMENT)	Primary Key pengguna
Name	VARCHAR(255)	Nama lengkap pengguna
Email	VARCHAR(255), UNIQUE	Email untuk login
Password	VARCHAR(255)	Password terenkripsi
Role	ENUM(‘pegawai’, ‘admin’)	Menentukan hak akses API
Created_at	TIMESTAMP	Waktu data dibuat
Updated_at	TIMESTAMP	Waktu data diperbarui

Attendance Table Structure

The 'attendances' table consists of 8 attributes, namely 'id', 'user_id', 'clock_in', 'clock_out', 'status', 'note', 'created_at', and 'updated_at'. The 'attendance' table is connected to the 'users' table via the foreign key 'users_id' which is taken from the 'users' table. This table is used to record when the clock-in is done by the employee, and also when the clock-out is done by the employee. If it has been recorded to be clocked in that day, then if it is clocked in again, it will be rejected due to duplication of clock-in data. Here is an example of an 'attendances' table that has been filled with data.

id	user_id	clock_in	clock_out	status	note	created_at	updated_at
2	1	2025-12-11 21:57:43	2025-12-11 21:58:07	hadir	NULL	2025-12-11 21:57:43	2025-12-11 21:58:07

Figure 6. Example Data Tabel Attendances

Table 2. Attendance Table Structure

Nama Kolom	Tipe Data	Keterangan
Id	BIGINT (PK, AUTO_INCREMENT)	Primary Key absensi
User_id	BIGINT (FK -> users.id)	Identitas pengguna absensi
Clock_in	DATETIME	Waktu pengguna melakukan clock-in
Clock_out	DATETIME, NULLABLE	Waktu pengguna melakukan clock-out
Status	VARCHAR (50)	Status Kehadiran
Note	TEXT, NULLABLE	Catatan tambahan
Created_at	TIMESTAMP	Waktu data dibuat
Updated_at	TIMESTAMP	Waktu data diperbarui

2.5. REST API implementation

The implementation phase was carried out using the Laravel framework due to its support for modern API development and clear application structures [13],[25]. The system implementation includes the creation of migrations, models, controllers, API routing, middleware, and token authentication using Laravel Sanctum [15].

2.5.1. Routing API

Routing is implemented via the routes/api.php file. The main endpoints consist of:

```
    Route::post('/register', [AuthController::class,
'register']);
    Route::post('/login', [AuthController::class, 'login']);
    Route::middleware('auth:sanctum')->group(function()
{
        Route::post('/attendance/clock-in',
[AttendanceController::class, 'clockIn']);
        Route::post('/attendance/clock-out',
[AttendanceController::class, 'clockOut']);
        Route::get('/attendance/history',
[AttendanceController::class, 'history']);

        Route::middleware('admin')->group(function() {
            Route::get('/attendance/all',
[AttendanceController::class, 'all']);
            Route::get('/attendance/report/daily',
[AttendanceController::class, 'dailyReport']);
            Route::get('/attendance/report/monthly',
[AttendanceController::class, 'monthlyReport']);
        });
});
```

The program code above shows the routing implementation for the REST API defined in the routes/api.php file. Each route line connects an endpoint URL with a function on the corresponding controller. The /register and /login endpoints are defined using the POST method because they both process data submitted by the client to create new accounts and authenticate users. After the login process is successful, the system generates an authentication token that is then used to access other endpoints.

All attendance-related endpoints, such as /attendance/clock-in, /attendance/clock-out, and /attendance/history, are placed inside the auth:sanctum middleware group. This ensures that only users who are logged in and have a valid token can access these features.

In addition, there is an additional middleware group of admins that wrap the report endpoint and access all attendance data (/attendance/all, /attendance/report/daily, and /attendance/report /monthly). With this

approach, routing serves not only as a URL mapping to a function, but also as a starting layer of role-based access control at the API level. The route structure used reflects the principles of REST with the use of descriptive paths and a clear separation of responsibilities between authentication, attendance, and reporting. [26]

2.5.2. System Model

The user and attendance models shown in the program below represent the structure of the table in the database as well as the relationships between entities in the system. The User and Attendance model uses Eloquent ORM [13], the following is the program line:

```
class User extends Authenticatable {
    use HasApiTokens;
    protected $fillable = ['name','email','password','role'];
}
```

Notice the code above, the User model uses the HasApiTokens trait derived from Laravel Sanctum so that each user instance can generate and manage authentication tokens. The $fillable attributes in this model are defined as ['name', 'email', 'password', 'role'], which means that these fields can be mass-assigned during the process of creating or updating data. The existence of role attributes allows the system to distinguish between users and employee and admin roles, which are then used by middleware to set access rights.

```
class Attendance extends Model {
    protected $fillable = ['user_id','clock_in','clock_out','status'];
    public function user() { return $this->belongsTo(User::class); }
}
```

Note the code above, the Attendance model represents the attendance table and has $fillable attributes that include user_id, clock_in, clock_out, status, and notes. This model also defines the belongsTo relationship to the User model through the user() function. This relationship states that each attendance record is related to one specific user. By utilizing Eloquent ORM, these relationships allow for more intuitive data invocation, such as accessing user data from an attendance object or otherwise accessing the entire attendance list of a user object. The design of this model plays an important role in maintaining data consistency and facilitating the implementation of business logic at the controller level.

2.5.3. Middleware Role Admin

This middleware ensures that only admins can access attendance reports:

```
public function handle($request, Closure $next) {
    if ($request->user()->role !== 'admin') {
        return response()->json(['message' => 'Forbidden'], 403);
    }
    return $next($request);
}
```

The Admin Middleware in the Program Code above serves as an additional layer of security that restricts access to certain endpoints only to users with admin roles. This middleware leverages the Request object to obtain information about the user who is being authenticated via $request->user(). Next, a check is carried out on the role attributes; If the role value is not the same as 'admin', then the middleware returns a JSON response with a Forbidden HTTP code state of 403 and a message explaining that access was denied.

If the role requirement is met, then the middleware forwards the request to the next process by calling $next($request). Thus, this middleware implements a server-side role-based access control mechanism. This approach ensures that even if users have a valid token, they still can't access administrative functions such as daily reports, monthly reports, or a list of all attendances if they don't have an admin role. The use of this middleware makes authorization logic centralized, easy to manage, and consistent across protected endpoints.

2.5.4. Attendance Logic (Clock-in)

```
public function clockIn() {
        $today = now()->toDateString();
        $already = Attendance::where('user_id',
auth()->id())
                        ->whereDate('clock_in', $today)
                        ->exists();
        if ($already) {
            return response()->json(['message' => 'Already
clocked in'], 422);
        }

        return Attendance::create([
            'user_id' => auth()->id(),
            'clock_in' => now(),
            'status' => 'hadir'
        ]);
}
```

The attendance logic for the clock-in process shown in the Program Code above implements the business rule that each employee should only clock-in once a day. The clockIn() function first retrieves today's date using now()->toDateString(). The system then checks whether there are already attendance records in the attendances table for the same user_id and those dates by using the whereDate and exists() queries. If the check results show that there is already a clock-in record, the function returns a JSON response with a rejection message and a 422 Unprocessable Entity code status. This indicates that the request is considered invalid from a business rules perspective, even though the request is syntically correct.

If there is no clock-in record for the day, the system creates a new entry in the attendances table by storing user_id, clock-in time (now()), and attendance status (e.g. 'attended'). The newly created attendance object is then returned as part of the JSON response. Thus, this function not only writes data to the database but also ensures that the attendance pattern follows predefined rules. The implementation of this logic is an example of applying business logic directly to a controller that is integrated with Eloquent ORM and a token-based authentication mechanism. The implementation conforms to modern API patterns and follows the recommended RESTful design [3], [27].

2.6. System Testing

API testing is done using Postman, one of the most commonly used tools in modern API testing [5], [28]. Testing is performed on all endpoints to validate: 1) HTTP code status (200, 401, 403, 422) [17], 2) JSON structure consistency [10], 3) Validation of Laravel Sanctum token authentication, 4) Error handling such as clock-in twice, 5) Role-based access restrictions.

Test scenarios include positive tests and negative tests to ensure that all functions are running as needed and safe from unauthorized access. All test results were recorded and analyzed to validate the API's compliance with REST standards [4], [29].

3. RESULTS AND DISCUSSION

This section presents the results of the implementation of the employee attendance system REST API and an analysis of the performance and validity of API responses based on testing using Postman. All test results are presented in the form of tables, snippets of JSON responses, and analysis of their compatibility with REST API theory and HTTP standards [1], [17].

3.1. API Implementation Results

The REST API was successfully built using Laravel, with endpoints including registration, authentication, attendance logging, attendance history retrieval, and daily and monthly reports for admins. The implementation of the Sanctum token authentication feature allows each private endpoint to be accessed only with a valid token, as per modern authentication practices [15], [16].

The JSON response structure is consistently provided according to API design guidelines [6], [19]. Each endpoint returns the corresponding HTTP code state, such as 200 OK, 401 Unauthorized, 403 Forbidden, and 422 Unprocessable Entity, following the RFC 7231 standard [17].

3.2. Proof of API Success in the Form of JSON Snippets

The following snippet is a snapshot of Postman's response from the API that has been implemented.

3.2.1. Registration Successful



```
{
  "status": "success",
  "message": "User registered successfully",
  "user": {
      "id": 1,
      "name": "Dimas",
      "email": "dimas@example.com"
  },
  "token": "3|q2jf...xyz"
}
```


Figure 7. Registration Successful

The response snippet shows the results of the POST /api/register endpoint test when the user registers with valid data. The status value is "success" and is followed by a message "User registered successfully" which indicates that the registration process has been processed without errors by the server. User objects contain newly created user information, including ids, names, and emails stored in the database. In addition, the system also generates a token that will be used as authentication credentials on subsequent requests. This is consistent with the concept of token-based authentication in the REST API, where after the registration or login process, the client acquires a token to access the protected endpoint.

3.2.2. Login Successfully



```
{
    "status": "success",
    "message": "Login berhasil",
    "user": {
        "id": 1,
        "name": "Dimas",
        "email": "dimas@example.com",
        "email_verified_at": null,
        "created_at": "2025-12-11T10:02:05.000000Z",
        "updated_at": "2025-12-11T10:02:05.000000Z",
        "role": "pegawai"
    },
    "token": "4|cSLJlai86xsRLdwE4aQzVEcENvqk2oZYGToIIpp
}
```


Figure 8. Login Successfully

The above response is the result of testing the POST /api/login endpoint with the correct email credentials and password. The system returns a "success" status and a "Login successful" message indicating that authentication was successful. No sensitive information such as passwords is returned, but only authentication tokens generated by Laravel Sanctum. This token is used as a token bearer on the Authorization header for each request that requires authentication. This behavior is in line with the security practices of the REST API, where the user's identity is represented by a token that is temporary and can be revoked at any time.

3.2.3. Clock-in Successful



```
{
    "status": "success",
    "message": "Clock-in berhasil",
    "data": {
        "user_id": 1,
        "clock_in": "2025-12-11T21:57:43.045429Z",
        "status": "hadir",
        "updated_at": "2025-12-11T21:57:43.000000Z",
        "created_at": "2025-12-11T21:57:43.000000Z",
        "id": 2
    }
}
```


Figure 9. Clock-in Successful

This response snippet is generated from the POST /api/attendance/clock-in endpoint test when the user clocks in for the first time of the day with a valid token. The status value of "success" and the message "Clock-in recorded" indicate that the server successfully saved the attendance data into the attendances table. Inside the data object it appears that user_id refers to the user who is logged in, clock_in contains a timestamp when the request is processed, and the status states the condition of attendance, e.g. "present". This response is also proof that the relationship between the user and attendance is working well and that the authentication mechanism successfully connects the request with the correct user identity.

3.2.4. Clock-in Rejected (Duplication)

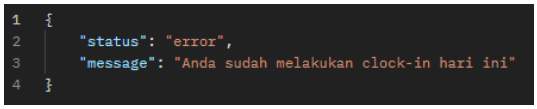


Figure 10. Clock-in Rejected

The above response shows the test results when the user tries to clock-in for the second time on the same day. Although the response format is simpler, the "You've clocked in today" message explains that the request was rejected for violating established business rules, which is that one employee can only clock-in per day. On the HTTP side, the server returns a 422 Unprocessable Entity code state, which indicates that the request is syntactically correct but cannot be processed because it does not meet the rules of application logic. This proves that the business validation of the clockIn() function is working correctly and helps maintain consistency of attendance data in the database.

3.2.5. Clock-out Successful



Figure 11. Clock-out Successful

This response results from the POST /api/attendance/clock-out endpoint test when the user who has previously clocked in clocked out correctly. The "success" status and the "Clock-out successful" message indicate that the system successfully updated the attendance entry in question by adding a clock_out value. The data object displays the timestamp of the exit time stored in the database. This behavior indicates that the system is able to correlate between clock-in and clock-out records for the same day, and that an update query against the attendances table has been executed appropriately. Thus, the recorded attendance data can be used for the purpose of reporting daily and monthly working hours.

3.2.6. 403 Responses for Employees Accessing Admin reports



Figure 12. Response Error/Rejected

This snippet is the result of a test when a user with an employee role tries to access an admin-only endpoint, such as /api/attendance/report/daily or /api/attendance/report/monthly. The system returns a "Forbidden" message with a Forbidden code status of 403, which means the user has been authenticated but does not have the right to access the resource. This response is generated by the admin middleware that checks

the role attributes on the user object before forwarding the request to the controller. This behavior proves that the system's role-based access control mechanism works well, thus ensuring that sensitive data can only be accessed by the authorities.

3.3. API Testing Scenario Table

The test scenario table (Table 3.1) is compiled to ensure that each function in the REST API runs according to the functional requirements that have been designed. Each scenario includes specific input conditions, expected outputs, and HTTP states that the system should return. Through this table, all endpoints are tested using the positive test (valid case) and negative test (error case) approach. This test allows verification that the API not only provides correct results under normal conditions, but is also capable of handling invalid conditions such as incorrect credentials, clock-in duplication, or unauthorized access. As such, this scenario table serves as a basis for assessing the completeness and accuracy of API behavior against HTTP business rules and standards. Here's a table of the scenarios we used:

Table 3. API Functional Testing Scenarios

Fitur	Input	Expected Output	Status HTTP
Registrasi	Data valid	Akun dibuat, token keluar	200
Login	Kredensial valid	Token keluar	200
Login	Password salah	Unauthorized	401
Clock-in	Pertama hari itu	Clock-in tercatat	200
Clock-in	Kedua kali	Ditolak	422
Clock-out	Tanpa Clock-in	Ditolak	422
Riwayat	Token valid	Data riwayat tampil	200
Laporan admin	Role admin	Data laporan tampil	200
Laporan Admin	Role pegawai	Forbidden	403

3.4. Test Results Table

The test results table (table 3.2) provides a summary of the execution of all test scenarios that have been performed. This table shows whether each endpoint is responding in accordance with the expected output. The test results showed that all key functions—from registration, login, clock-in, clock-out, to admin report access—were running as designed. Endpoints that require authentication successfully restrict access using tokens, while admin-specific endpoints are successfully filtered by role-based access control middleware. This table proves that the API implementation meets the standards of functionality, security, and consistency of JSON response required for integration with frontend or mobile applications.

Table 4. API Endpoint Test Results

Endpoint	Deskripsi	Hasil Uji	Kesimpulan
POST /register	Pendaftaran pengguna	Token & data user muncul	Berhasil
POST /login	Autentikasi	Token Muncul	Berhasil
POST /attendance/clock-in	Absensi Masuk	Data disimpan ke DB	Berhasil
POST /attendance/clock-in	Duplikasi Clock-in	Error 422	Valid
POST /attendance/clock-out	Absensi Keluar	Data diperbarui	Berhasil
Get /attendance/ history	Riwayat absensi	Data tampil	Berhasil
Get /attendance/report/history	Akses admin	Laporan muncul	Berhasil
Get /attendance/report/daily	Akses pegawai	403 Forbidden	Berhasil

3.5. API Performance Analysis

Testing shows that the entire endpoint provides a fast and consistent response. The JSON response structure follows the common pattern of modern REST APIs[2], [3]. From the test results, the API is proven to meet: 1) The HTTP response standard is RFC 7231 [17]. 2) REST principles: stateless, uniform interface, resource-based[1],[2]. 3) Token-based security, as recommended for distributed applications [15], [16]. 4)Consistency of data structure facilitates integration with mobile and web applications [6], [21], [30].

The admin report endpoint successfully differentiates access based on user roles, according to the theory of Role-Based Access Control (RBAC) [16], [22]. This proves that middleware and authorization are effective. Business validation such as double clock-in prevention and pre-clock-in has also worked according to needs analysis. This is in line with the business rule enforcement theory in modern APIs [20], [29]. Overall, the implementation results show that the API runs stable, secure, and consistently and is ready to be integrated with any frontend system that uses the HTTP protocol as a communication mediator.

4. CONCLUSION

This research successfully implemented an employee attendance system based on REST API using the Laravel framework with token authentication through Laravel Sanctum. All core features, including registration, login, clock-in and clock-out logging, attendance history retrieval, and daily and monthly reports for admins, have been working well and according to the design requirements. The results of testing using Postman show that each endpoint produces a consistent, accurate, and HTTP-compliant response. In addition, the authentication and authorization mechanisms are effective, as demonstrated by access restrictions based on user roles, token validation, as well as handling business logic errors such as the prevention of double clock-in and pre-clock-in without clock-out. This implementation proves that the REST API built meets REST principles, has a standardized data structure, and is ready to integrate with various frontend platforms.

However, this study still has some limitations, such as the unavailability of a location verification feature (GPS) or proof of presence in the form of photos, and a stress test has not been carried out to measure the API's ability to handle large-scale requests. Further research suggests adding additional security features, improving monitoring and logging capabilities, and integrating APIs with mobile or web applications for use on a broader operational scale. Thus, this research provides a strong foundation for the development of a secure, flexible, and distributed digital attendance system.

REFERENCES

[1] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," *Ph.D. Diss. Univ. California, Irvine*, no. 1995, p. 662, 2000.
[2] L. R. and S. Ruby, "RESTful Web Services," *O'Reilly Media*, 2007.
[3] M. Masse, *REST API Design*. 2012.
[4] A. Pautasso, "REST APIs: A Survey," *Futur. Internet J.*, vol. 6, no. 4, 2014.
[5] Postman Inc., "API Testing Framework," [Online]. [Online]. Available: https://www.postman.com/api-platform/api-testing/
[6] Google Developers, "API Design Guide," [Online].
[7] R. Buyya, *Cloud Computing: Principles and Paradigms*. 2011. doi: 10.1002/9780470940105.ch15.
[8] D. Barry, *Web Services and Service-Oriented Architectures*, MA: Morgan. Burlington: MA: Morgan, 2003. [Online]. Available: https://archive.org/details/webservicesservi0000barr_p0b8/page/n9/mode/2up
[9] J. Webber, "REST in Practice," *IEEE Internet Comput.*, vol. 6, no. 2, pp. 92–95, 2010.
[10] D. Crockford, "The application/json Media Type for JavaScript Object Notation (JSON)," *RFC 4627*, vol. 122, no. 1995, pp. 25–27, 2006.
[11] M. Fowler, "Microservices," martinfowler.com.
[12] Amazon Web Services, "API Gateway Developer Guide."
[13] Laravel, "Laravel Documentation," [Online]. [Online]. Available: https://laravel.com/docs/
[14] M. Amundsen, "Building Hypermedia APIs with HTML5 and Node," *Sebastopol, CA O'Reilly Media*, 2013.
[15] Laravel Sanctum Documentation, "Token Authentication," [Online]. [Online]. Available: https://laravel.com/docs/sanctum
[16] M. Benantar, *Access Control Systems: Security, Identity Management and Trust Models*. New York, 2006.
[17] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," *RFC 7231*, 2014.
[18] M. Fowler and J. Lewis, "Microservices: a Definition," 2014.
[19] NGINX, "API Gateway Design Patterns," 2022.
[20] Microsoft Azure, "REST API Best Practices."
[21] P. Bourgon, "API Design for Scale," *ACM Queue*, vol. 17, no. 17, pp. 40–66, 2019.
[22] C. Pahl, "Service-Oriented Architecture: A Systematic Review," *ACM Comput. Surv.*, vol. 40, no. 2, pp. 1–39, 2007.
[23] K. Sim, "Web Service Performance Metrics," *IEEE Softw.*, vol. 22, no. 3, pp. 70–77, 2005.
[24] and J. W. H. Garcia-Molina, J. D. Ullman, *DATABASE SYSTEMS The Complete Book*, 3rd ed. Boston: MA: Pearson, 2021.
[25] A. Sehgal, *Designing APIs with Swagger and OpenAPI*. New York: NY: Apress, 2021.
[26] IBM Developer, "REST API Handbook," 2018.
[27] J. Bloch, *Effective Java*, 3rd ed. Boston, 2017.
[28] Sun Microsystems, "Representational State Transfer (REST) Architectural Style."
[29] T. Banks, *Microservice API Patterns*. Boston, 2020.
[30] M. Daconta, *XML Web Services Development*. Indianapolis, 2003.