Smart Attendance System

Atharv Rangrao Patil[1], Prathmesh Madhukar Kanekar[2], Abhijeet Vishwas Jadhav[3],
Sharvari Bhagwan Karande[4], Nurain Aanis Sayyad[5]
[1,2,3,4,5]*Department of Computer Science, Sanjeevan Group of Institutions, Panhala, Kolhapur, India*

***Abstract*—This paper presents the design and architectural validation of a secure, multi-role student attendance management system utilizing geofencing technology. Traditional attendance methods are inefficient and susceptible to manipulation, demanding a robust digital alternative. The proposed system, built on a Flutter mobile client, a Node.js application layer, and a PostgreSQL database extended with PostGIS, addresses these limitations. A core contribution is the implementation of a highly secure server-side virtual room access control mechanism. Unlike susceptible client-side validation, the system enforces presence verification by performing Point-in-Polygon checks using PostGIS spatial functions, ensuring the integrity of the attendance record. Authorization is managed through a stringent Role-Based Access Control (RBAC) hierarchy (Student, Teacher, HOD, Principal), with fine-grained data isolation for administrative roles achieved via PostgreSQL's native Row-Level Security (RLS). This architecture ensures that attendance links are only accessible when the student's authenticated location is confirmed by the server to be within the designated 500-meter by 500-meter ArcGIS-defined virtual room boundary. Evaluation focuses on maintaining data integrity, providing accurate departmental data visibility for Heads of Department (HODs), and establishing a scalable, tamper-proof foundation for educational institutions.**

***Index Terms*—Geofencing, Role-Based Access Control, PostGIS, Row-Level Security, Flutter, Attendance Management.**

I. INTRODUCTION

A. Background and Motivation

Accurate and efficient attendance tracking is fundamental for academic assessment, resource management, and policy compliance within educational institutions. However, conventional methods including manual sign-in sheets, sign-in systems, or traditional punch-ins are universally recognized as inefficient, time-consuming, and highly vulnerable to academic misconduct, particularly proxy attendance or fraudulent check-ins. These shortcomings significantly increase administrative burden and undermine the integrity of student participation records. To mitigate these deficiencies, the adoption of ubiquitous computing solutions leveraging the Global Positioning System (GPS) capabilities inherent in modern mobile devices has become necessary for modernizing attendance tracking. Geofencing, which defines a virtual perimeter or boundary around a specific geographic area, offers a scalable and automated solution by verifying presence based on real-time location coordinates. Research indicates that such geofencing-based systems can significantly reduce administrative workload and enhance data reliability.

The proposed system mandates a highly secure, scalable, and location-aware architecture to manage student attendance across multiple organizational tiers. The chosen stack Flutter for the front-end, Node.js for the API layer, and PostgreSQL enhanced with PostGIS for the data tier perfectly aligns with a robust 3-Tier application pattern, ensuring separation of concerns and maximizing specialized performance.[1]

1.1 Technology Stack Validation

The architecture adheres to the standard model: the Presentation Tier focuses on the user experience; the Application Tier manages business logic and access control; and the Data Tier handles persistence and complex analysis.[2]

Presentation Tier (Flutter): Flutter provides a single codebase for high-performance, cross-platform mobile applications (Android and iOS). This consistency is crucial for the diverse user base (Students, Teachers, HODs, Principal). Furthermore, Flutter enables direct access to high-accuracy native location APIs, a necessary prerequisite for reliable

geofencing.[3]

Application Tier (Node.js/Express): Node.js is selected for its non-blocking I/O model, which is highly efficient for handling numerous concurrent requests, such as simultaneous attendance logging by large student cohorts at the start of a class. This tier is responsible for secure authentication, authorization (RBAC), and enforcing all business rules before communicating with the database.[2]

Data Tier (PostgreSQL/PostGIS): PostgreSQL is a reliable enterprise-grade relational database. The mandatory inclusion of the PostGIS extension transforms the Data Tier into a powerful geospatial engine.[5] PostGIS enables the native storage of virtual room geometries (polygons) and performs efficient, indexed spatial queries (ST Contains), which is foundational for the real-time location validation required by the system.

Table 1: Proposed Technology Stack and Purpose Mapping

Feature/Tier	Technology	Primary Role
Presentation Tier (Mobile)	Flutter (Dart)	Cross-platform UI, high-accuracy location access, API consumption.
Native Location API	Fused Location Provider Client (Android) / Core Location (iOS)	Precise, battery-aware GPS coordinate acquisition.[3]
Application Tier (API)	Node.js (Express)	Business logic execution, Authentication (JWT), RBAC enforcement, API gateway.
Data Persistence & Analysis	PostgreSQL	Core data storage, security context management (RLS).
Geospatial Engine	PostGIS Extension	Geometry storage (Virtual Rooms), highly efficient server-side Point-in-Polygon validation (ST_Contains).[6]
Mapping Components	ArcGIS Maps SDK for Flutter / ArcGIS JS API	Native map rendering for students; Web-based geofence definition for teachers.[8]

1.2 Optimized 3-Tier Architecture Flow and Security Separation

The architectural design enforces strict separation of security responsibilities. The most critical element is establishing that the core geofencing validation does not occur on the mobile device, where it is easily compromised, but securely within the trusted Application and Data Tiers.

The Presentation Tier's responsibility is limited to acquiring the most accurate GPS reading possible using high-fidelity native location APIs.[3] This signed location, along with the attendance link identifier, is transmitted to the Node.js API. The server then utilizes the Data Tier to perform the required spatial validation. This approach ensures that a student cannot bypass the location requirement through simple client-side manipulations or GPS spoofing.

The implication of this decision is that the API endpoint responsible for checking the student's location (/api/v1/attendance/check-location) must be highly optimized for high throughput. This performance requirement is met by delegating the Point-in-Polygon (PIP) check directly to the PostGIS extension, leveraging its indexed spatial capabilities, which are significantly faster than application-level geometric calculations.[10]

1.3 ArcGIS Integration Strategy

ArcGIS functionality is required for two distinct purposes: defining the geofence and displaying the map interface.

For Geofence Definition (Teacher/Office role), the ArcGIS Maps SDK for JavaScript is the appropriate tool. This will likely be embedded within a Flutter WebView component or a dedicated web dashboard, allowing the teacher to easily define the 500m x 500m virtual room by selecting a center point. The JavaScript API can calculate the bounding box extent around this point.[11] This boundary data (four coordinates forming a rectangle) is then securely transmitted to the Django backend for persistence in PostGIS.

For Student Map Display, the native ArcGIS Maps SDK for Flutter is recommended.[9] Utilizing the native SDK ensures superior performance, better battery management, and a seamless user experience, which is preferred over rendering continuous map views within a WebView.[8]

II. AUTHENTICATION AND FINE-GRAINED ROLE-BASED ACCESS CONTROL (RBAC)

System security requires robust authentication using JSON Web Tokens (JWTs) and meticulous enforcement of role-based access control across all API endpoints.

2.1 Secure Authentication Flow: JWT Implementation

The system will implement a token-based authentication mechanism using JWTs to secure communications between Flutter and Node.js. Upon successful login, the Node.js API issues two tokens: a short-lived Access Token used for standard API calls, and a long-lived Refresh Token used solely to acquire a new Access Token without requiring the user to re-enter credentials.[14]

The JWTs generated by the Node.js layer must contain essential claims, including the user_id, the user's role, and critically, the organizational identifier such as the dept_id. These claims form the security context required for downstream authorization checks and database filtering.[16] The refresh token itself should be stored securely in the PostgreSQL database and invalidated upon use to mitigate token theft risks.[14]

2.2 Node.js Application Tier Security and RBAC Middleware

The Node.js API acts as the primary gatekeeper, utilizing middleware to enforce security policies.

First, API Rate Limiting must be implemented using libraries like express-rate-limit to prevent denial-of-service attempts and brute-force attacks against login and critical attendance endpoints.[17] Second, Input Validation and sanitization are essential security best practices to protect the system from common web vulnerabilities like SQL injection, especially on fields receiving user input such as GPS coordinates and link identifiers.[19]

The RBAC Middleware executes after token validation, utilizing the claims embedded in the JWT:

1. Context Extraction: The middleware decodes the JWT to retrieve the user's role and dept_id.
2. Permission Check: It verifies if the requested operation (e.g., creating an attendance link) is authorized for the identified role (e.g., 'Teacher').[20]
3. Data Scope Check: The middleware ensures that the action requested is within the user's defined scope (e.g., a teacher can only manage attendance for subjects they are assigned to).

2.3 Core Role-Permission Matrix and Data Scoping

The system employs a strict RBAC model, supplemented by organizational unit filtering required for administrative roles.

Table 2: Role-Based Data Access Control (RBAC) Matrix

Role	Module Access	Data Scope (Visibility)	Key Responsibility/Functionality
Principal	Full College Dashboard	Full college data (All Departments, All Students)	Global oversight and aggregate counts across the institution.
HOD	Department Dashboard	Department-specific data only (Students, Teachers, Classes)	Monitoring and calculating attendance limited to their department.
Teacher	Teacher Module (CRUD)	Subjects/Classes Assigned; Student attendance records.	Create virtual rooms, generate attendance links, manage subject-specific attendance counts.
Student	Student Module (Read/Write Self)	Self-attendance records; Geofence-controlled link access (Write)	View active links and mark attendance only after location validation.
Office	Setup/Admin Module (CRUD)	College-wide administrative data (Assignments, Roles)	Assigning teachers to subjects and classes, and managing Class Coordinator designations.

III. DATA ISOLATION AND POSTGRESQL/POSTGIS SCHEMA BLUEPRINT

The data tier must be designed to support both high-performance geospatial analysis and stringent departmental data isolation requirements.

3.1 Data Model Design

The system employs a row-based tenancy model where data from all departments resides in shared tables, secured by the dept_id column which serves as the tenant identifier.[22] PostGIS is enabled on the database, allowing for the storage of location data as specialized geometry types.

Geospatial Data: The virtual_rooms table will include a column named geom_polygon storing the 500m x 500m geofence definition as GEOMETRY (POLYGON, 4326) (WGS84 spatial reference identifier). The attendance_logs table will store the student's logged check-in location in a similar format, GEOMETRY (POINT, 4326).[5]

Indexing: Crucially, the geom_polygon column must utilize a GiST (Generalized Search Tree) index. This spatial index significantly improves the efficiency of queries involving geometric relationships, such as the real-time Point-in-Polygon checks required for attendance validation.[10]

3.2 Implementing Departmental Data Isolation using RLS

To ensure the HOD role only accesses data relevant to their department, Row-Level Security (RLS) in PostgreSQL is the most robust solution.[25] RLS ensures that access policies are enforced at the database layer, mitigating the risk of data leaks caused by application-level coding errors.[22]

The implementation involves two steps:

1. Context Setting in Node.js: The Node.js RBAC middleware must execute a dedicated SQL command at the start of every database transaction session for HOD users: SET app.current_dept_id TO 'the_hod_department_uuid'; This sets a secure, session-scoped variable.[22]
2. PostgreSQL Policy Definition: An RLS policy is enabled on sensitive tables (e.g., attendance_logs, users, classes) to automatically filter rows based on the session variable: SQL CREATE POLICY dept_isolation_policy ON attendance_logs FOR SELECT USING (dept_id = current_setting('app.current_dept_id'):UUID);

This policy ensures that when an HOD runs a dashboard query (e.g., SELECT * FROM attendance_logs), PostgreSQL transparently adds the necessary department filter, restricting the result set.[26] This approach simplifies the application code and centralizes security enforcement.[23] The Principal role is handled either by bypassing RLS or by setting a policy that grants full visibility, ensuring the same SQL queries can be run by different roles while yielding correctly scoped data.

IV. THE GEOFENCING ENGINE: LOCATION ACCURACY AND SERVER VALIDATION

The integrity of the attendance system depends entirely on the accuracy of the location data and the security of the server-side validation process.

4.1 Mobile Client Location Acquisition

The Flutter application must utilize the highest available native location accuracy settings. For Android devices, this means using the Fused Location Provider Client configured with PRIORITY_HIGH_ACCURACY.[3] This API intelligently combines GPS, Wi-Fi, and other sensors to provide the best possible geospatial positioning.

Developers must acknowledge that even high-accuracy location services may yield results with variances, potentially reporting accuracy no better than 10 meters in certain environments.[27] The system must account for this inherent inaccuracy by implementing a slight tolerance margin around the 500m x 500m geofence boundary. To combat basic location spoofing, the Flutter app must capture and send the location data along with its metadata, specifically the reported accuracy and timestamp, allowing the Django backend to reject stale or unreliable readings.

4.2 Virtual Room Creation and Geometry Persistence

When a teacher defines a virtual room using the ArcGIS interface, the Django backend manages the conversion of the defined area into a geometric polygon stored in PostGIS.

1. Center Point Submission: The teacher submits the center latitude and longitude (Lat/Lon).
2. Polygon Calculation: Since the requirement is for a 500m $\times$ 500m area, the Django server must accurately calculate the four corner points of the rectangle, accounting for the spherical nature of the Earth (geodesy) when working in WGS84 (SRID 4326). PostGIS functions like ST_Project can determine coordinates based on distance and bearing from the center point, facilitating precise corner calculation.[28]
3. Geometry Insertion: The resulting four coordinates define a polygon that is inserted into the virtual_rooms.geom_polygon column using ST_MakeEnvelope or ST_MakePolygon.[30] The room is marked as active.

4.3 Secure Server-Side Point-in-Polygon (PIP) Validation

The core attendance security measure is the validation API endpoint, which processes the student's location against the room geometry.

Upon receiving the student's location data, the Django API executes a highly optimized spatial query against PostGIS. The most efficient method for checking if a point is within a polygon uses the ST_Contains function.[7]

The structure of the query, leveraging the GiST index on the polygon column, ensures low latency:

SQL

```
$$
SELECT ST\_Contains(vr.geom\_polygon, ST\_SetSRID(ST\_MakePoint(:longitude, :latitude), 4326))
FROM virtual\_rooms vr
WHERE vr.id = :linkId
AND vr.is\_active = TRUE;
$$
```

If the database returns TRUE, the Django API grants the student access to mark themselves present and logs the attendance, including the location metadata. If FALSE, the link remains hidden or inaccessible to the student, fulfilling the location-aware access control requirement. Performance optimization is critical here, as the query speed dictates the user experience at class start times. Using ST_Contains on an indexed geometry is significantly faster than relying on bounding box checks alone, although spatial indexes are key for fast performance regardless of the function used.[10]

V. MODULE-SPECIFIC IMPLEMENTATION AND FEATURE STRATEGY

The five designated modules (Principal, HOD, Teacher, Student, Office) are enabled by the underlying RBAC and RLS framework.

5.1 Teacher Module Implementation

The Teacher module focuses on managing subjects, generating links, and viewing attendance.

- Attendance Link Generation: Creating a link involves defining the time window and persisting the geofence geometry (Section 4.2) into the virtual_rooms table. The resulting ID is the attendance link identifier.
- Standard Teacher Permissions: Standard teachers can only view and manage data related to the subjects they are explicitly assigned to, enforced by application-level checks in the Django middleware.
- Class Coordinator Permissions: The functionality for a Class Coordinator to calculate attendance for all subjects in their assigned class requires an elevated data scope. The Django application must verify the user's role and the is_coordinator flag, allowing queries that join the users and classes tables to aggregate attendance data beyond just the teacher's individual subject assignments.[20]

5.2 Student Access Control Logic

The student's interaction is simple but highly governed by security measures.

The student accesses the attendance link through the Flutter app. Before displaying the "Mark Present" button, the app securely captures the student's current high-accuracy location.[3] This location is sent to the Django validation API (Section 4.3). The Flutter app's presentation logic is entirely conditional: the attendance link remains hidden or inaccessible until the server-side PostGIS check returns a positive validation. This crucial mechanism ensures that the student is physically present within the designated virtual room boundary before they can mark their attendance.

5.3 Principal/HOD Dashboard Strategy: Leveraging RLS for Aggregation

The difference in data visibility between the principal (college-wide) and the HOD (department-only) is handled by the RLS policies established in the Data Tier.

When an HOD accesses their dashboard, the Django session sets the app.current_dept_id.[22] Aggregate queries executed by the HOD, such as counting total present students, are automatically filtered by PostgreSQL's RLS policies.[26] This allows the HOD to see statistics restricted only to their department's student and attendance data.

The Principal's dashboard, requiring a view and count of all students across the entire college, executes the same aggregate queries, but their database session is

either configured to bypass the RLS policies or the RLS policy explicitly grants full visibility based on their 'Principal' role.

For large institutions, running real-time COUNT(*) queries on potentially millions of attendance_logs records can introduce latency.[32] To ensure dashboard responsiveness, a scheduled background task (run by Django) should calculate and populate department-level daily attendance summaries into a separate, indexed table (a Materialized View). The HOD and Principal dashboards would then query this optimized summary table, with RLS still enforced via the dept_id column on the summary data, guaranteeing fast loading times for institutional reporting.

5.4 Office Module Design

The Office module is the administrative hub responsible for defining the organizational structure and managing relationships. This requires the highest level of trust and stringent RBAC enforcement.

The module handles critical CRUD operations, including:

- Assigning teachers to subjects.
- Mapping subjects to specific classes.
- Designating specific teachers as Class Coordinators (is_coordinator flag).
- Managing user roles and department affiliations.

All data input in this module must undergo rigorous validation and sanitization in the Django backend to prevent administrative data corruption.[19]

VI. CONCLUSION AND RECOMMENDATIONS

The comprehensive analysis confirms that the Flutter, Django, and PostgreSQL stack, augmented by PostGIS, is optimal for building this role-based geospatial attendance system. The core success and security of the platform hinge on three non-negotiable architectural principles.

1. Server-Side Geospatial Validation: The geofence check must be performed exclusively in the Django Application Tier, utilizing the spatial efficiency of the PostGIS Data Tier (ST_Contains). This prevents GPS location spoofing by students, ensuring the integrity of the attendance log.
2. Row-Level Security (RLS) for Data Isolation: The requirement for HODs to see only departmental data must be enforced using PostgreSQL RLS policies tied to session variables (app.current_dept_id). This approach provides robust data isolation that is less prone to error than application-level filtering, simplifying the Django codebase.
3. Performance Optimization: Given the high concurrent load during class attendance, the system must employ GiST indexing on all PostGIS geometry columns and leverage Materialized Views for high-volume aggregate reporting required by the Principal and HOD dashboards.

Final Recommendations

- Geospatial Stack: Maintain the dual ArcGIS strategy, using the ArcGIS Maps SDK for JavaScript for the web-based geofence creation interface and the native ArcGIS Maps SDK for Flutter for the student's mobile map display, balancing feature requirements with performance gains.
- Security Implementation: Strictly enforce a JWT flow with refresh tokens and implement granular RBAC middleware in Django, complemented by aggressive API rate limiting and input validation on all endpoints, particularly those receiving location data.
- Data Accuracy Handling: Implement logic in the Django backend to evaluate the accuracy and timestamp metadata submitted by the Fused Location Provider Client, automatically rejecting location attempts that fall outside acceptable quality thresholds to further mitigate fraudulent check-ins.

REFERENCES

[1] “3 Tier Architecture,” Scribd. [Online]. Available: https://www.scribd.com/document/851150949/3-Tier-Architecture

[2] “Understanding 3-Tier Architecture: A Comprehensive Guide,” Medium. [Online]. Available: https://medium.com/@ujjawalshriwastav9771/understanding-3-tier-architecture-a-comprehensive-guide-dcdc651ef9dc

[3] “Fused Location Provider API,” Google for

Developers. [Online]. Available: https://developers.google.com/location-context/fused-location-provider

[4] samuelkubinsky, "fused_location: A Flutter plugin for high-accuracy location and orientation data, leveraging advanced native APIs," GitHub. [Online]. Available: https://github.com/samuelkubinsky/fused_location

[5] "PostGIS: A Powerful Geospatial Extension for PostgreSQL," Red Hat Developer. [Online]. Available: https://developers.redhat.com/articles/2025/10/02/postgis-powerful-geospatial-extension-postgresql

[6] PostGIS. [Online]. Available: https://postgis.net/

[7] Oleg, "Geospatial Polygons: A Full-Stack Guide with PostGIS, Node.js," Medium. [Online]. Available: https://medium.com/@KilgortTrout/geospatial-polygons-a-full-stack-guide-with-postgis-node-js-and-react-da55ab0e7fdd

[8] "Mobile & Desktop Development | ArcGIS Maps SDKs for Native Apps," Esri. [Online]. Available: https://www.esri.com/en-us/arcgis/products/develop-with-arcgis/features/arcgis-maps-sdks-for-native-apps

[9] "ArcGIS Maps SDK for Flutter," Esri Developer. [Online]. Available: https://developers.arcgis.com/flutter/

[10] "Find Points Within Polygon with PostGIS Really Slow," GIS Stack Exchange. [Online]. Available: https://gis.stackexchange.com/questions/338395/find-points-within-polygon-with-postgis-really-slow

[11] "Extent | API Reference | ArcGIS Maps SDK for JavaScript 4.33," Esri Developer. [Online]. Available: https://developers.arcgis.com/javascript/latest/api-reference/esri-geometry-Extent.html

[12] "Calculating Center Point Inside Polygon Using ArcGIS JavaScript API," GIS Stack Exchange. [Online]. Available: https://gis.stackexchange.com/questions/7998/calculating-center-point-inside-polygon-using-arcgis-javascript-api

[13] "Android Java Native (and Older SDK) vs Latest SDK and Flutter," Reddit. [Online]. Available: https://www.reddit.com/r/ArcGIS/comments/1n1nz87/android_java_native_and_older_sdk_vs_latest_sdk/

[14] "JWT Access Token and Refresh Token Flow," Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/67856006/jwt-access-token-and-refresh-token-flow

[15] "JWT Refresh Token Flow," Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/27726066/jwt-refresh-token-flow

[16] "JWT Token Authentication," FlutterFlow Docs. [Online]. Available: https://docs.flutterflow.io/integrations/authentication/firebase/jwt-auth/

[17] "API Rate Limiting in Node.js: Strategies and Best Practices," DEV Community. [Online]. Available: https://dev.to/hamzakhan/api-rate-limiting-in-nodejs-strategies-and-best-practices-3gef

[18] "Best Practices for Securing Node.js APIs," Medium. [Online]. Available: https://medium.com/deno-the-complete-reference/best-practices-for-securing-node-js-apis-0f3d2cd21e4d

[19] "11 Best Practices to Secure NodeJS API," Indusface Blog. [Online]. Available: https://www.indusface.com/blog/how-to-secure-nodejs-api/

[20] "How to Implement RBAC and ABAC Authorization in Node.js APIs with Frontegg." [Online]. Available: https://frontegg.com/how-to-implement-rbac-and-abac-authorization-in-node-js-apis-with-frontegg

[21] "Role-Based Access Control (RBAC) Made Easy: How to Implement It in Your Web App," Medium. [Online]. Available: https://medium.com/@kanishksinghmaurya/role-based-access-control-rbac-made-easy-how-to-implement-it-in-your-web-app-68cb8aabb4f9

[22] "Multi-tenant Data Isolation with PostgreSQL Row Level Security," AWS Database Blog. [Online]. Available: https://aws.amazon.com/blogs/database/multi-tenant-data-isolation-with-postgresql-row-level-security/

[23] "Implementing Fine-Grained Postgres Permissions for Multi-Tenant Applications."

[Online]. Available: https://www.permit.io/blog/implementing-fine-grained-postgres-permissions-for-multi-tenant-applications

[24] "Using WGS 84 and Geometry Calculations to Check if Point is Inside Polygon," GIS Stack Exchange. [Online]. Available: https://gis.stackexchange.com/questions/462416/using-wgs-84-and-geometry-calculations-to-check-if-point-is-inside-polygon

[25] "Documentation: 18: 5.9. Row Security Policies," PostgreSQL Documentation. [Online]. Available: https://www.postgresql.org/docs/current/ddl-rowsecurity.html

[26] "Mastering PostgreSQL Row-Level Security (RLS) for Rock-Solid Multi-Tenancy." [Online]. Available: https://ricofritzsche.me/mastering-postgresql-row-level-security-rls-for-rock-solid-multi-tenancy/

[27] "Why Does FusedLocationProviderApi Never Report Accuracy Better than 10m? Is This Documented?" Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/49289206/why-do