Attendance Tracker Project - Technical Documentation

📌 Project Overview

The Attendance Tracker is a full-stack web application that enables organizations to manage employee's attendance effectively. It allows for easy recording, reviewing, and reporting of attendance records. The system supports user registration, login, attendance marking, and administrative capabilities like generating reports.

🛠️ Tech Stack

Frontend:
- React.js – UI library for building responsive interfaces
- Axios – To handle HTTP requests
- Material UI – For styling the interface

Backend:
- Node.js – JavaScript runtime
- Express.js – Web framework for building APIs
- SQL Server (SSMS) – Relational database for storing data

Tools:
- Postman – For API testing
- Git – Version control
- VS Code – Code editor

🧱 System Architecture

High-Level Overview:
[React Frontend] <--> [Express Backend API] <--> [SQL Server Database]

1. React frontend sends HTTP requests via Axios to the Express backend.
2. Express backend handles authentication, business logic, and data operations.
3. SQL Server stores employees, and their attendance records.

🚀 Features

1. User Registration and Login
- Users can create an account or log in securely.

- Passwords are stored securely using hashing.
- JWT tokens are used for authentication.

2. Viewing Attendance Records
- Retrieve and display attendance based on employee name or date range.

3. Admin Features
- Add or remove employees
- View aggregated data
- Manage user roles (future scope)

4. Reporting and Analytics
- Attendance statistics can be shown as exported

Database Schema

Database Schema for GyansysAttendanceDB

This document provides a comprehensive overview of the database schema for **GyansysAttendanceDB**, including database settings, tables, views, stored procedures, user-defined types, indexes, and constraints. The database is designed to manage attendance-related data, including employee information, department details, swipe card transactions, and watchlists.

1. Database Information

Database Name: GyansysAttendanceDB

2. User-Defined Table Types

2.1. tvp_WatchListEmployees

Description: Used to pass a list of employees for watchlist operations.

Columns:

ColumnName	DataType
ID	INT NULL
EmployeeName	NVARCHAR(255) NULL
EmployeeEmail	NVARCHAR(255) NULL

3. Tables

3.1. DeptMaster

Description: Stores department information.

Columns:

ColumnName	DataType
ID	BIGINT IDENTITY(1,1) NOT NULL
DeptCode	NVARCHAR(6) NOT NULL
DeptName	NVARCHAR(30) NOT NULL
DeptType	NVARCHAR(1) NOT NULL
LocCode	NVARCHAR(5) NOT NULL
IsActive	BIT NOT NULL (Default: 1)
CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_DeptMaster_ID on ID
- **Unique**: UK_DeptMaster_DeptCode on DeptCode
- **Unique**: UK_DeptMaster_DeptName on DeptName

3.2. EmpMaster

Description: Stores employee master data.

Columns:

Column Name	Data Type
ID	BIGINT IDENTITY(1,1) NOT NULL
EmpID	NVARCHAR(15) NOT NULL
CardID	NVARCHAR(8) NOT NULL
EmpName	NVARCHAR(50) NOT NULL
FPImage1	NVARCHAR(1640) NULL
FPImage2	NVARCHAR(1640) NULL
DeptCode	NVARCHAR(6) NOT NULL
EmpEmail	NVARCHAR(100) NULL
Address	NVARCHAR(200) NULL
Gender	NVARCHAR(1) NULL
IsActive	BIT NOT NULL (Default: 1)

CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_EmpMaster_ID on ID
- **Unique**: UK_EmpMaster_EmpID on EmpID

3.3. NetXs_Misc

Description: Miscellaneous data from NetXs system, often used for department details.

Columns:

code	NVARCHAR(6) NULL
descr	NVARCHAR(30) NULL
type	NVARCHAR(1) NULL
loccode	NVARCHAR(5) NULL

3.4. NetXs_Emp

Description: Employee data from NetXs system.

Columns:

Column Name	Data Type
EmpID	NVARCHAR(15) NULL
CardID	NVARCHAR(8) NULL
empname	NVARCHAR(50) NULL
TzXsGrp	NVARCHAR(15) NULL
WoXsGrp	NVARCHAR(2) NULL
HoXsGrp	NVARCHAR(2) NULL
loccode	NVARCHAR(5) NULL
FPImage1	NVARCHAR(1640) NULL
FPImage2	NVARCHAR(1640) NULL
emppin	NVARCHAR(6) NULL
expirydate	DATETIME NULL
photo	IMAGE NULL
deptid	NVARCHAR(6) NULL
type	NVARCHAR(6) NULL
sitecode	NVARCHAR(6) NULL
empstatus	NVARCHAR(1) NULL

location	NVARCHAR(15) NULL
email	NVARCHAR(100) NULL
XsInfo	VARCHAR(8000) NULL
SapId	NVARCHAR(20) NULL
Dob	DATETIME NULL
gender	NVARCHAR(1) NULL
address	NVARCHAR(200) NULL
tel1	NVARCHAR(20) NULL
tel2	NVARCHAR(20) NULL
doj	DATETIME NULL
dol	DATETIME NULL
remark	NVARCHAR(200) NULL
MachineName	NVARCHAR(30) NULL
BFlag	NVARCHAR(1) NULL
MFlag	NVARCHAR(1) NULL

3.5. NetXs_GateSetup

Description: Gate setup data from NetXs system.

Columns:

Column Name	Data Type
CID	NVARCHAR(3) NULL
GtNo	NVARCHAR(2) NULL
GtDesc	NVARCHAR(50) NULL
GtType	NVARCHAR(1) NULL
OpenTime	NVARCHAR(2) NULL
DOTL	NVARCHAR(1) NULL
DOTLTime	NVARCHAR(2) NULL
ExVerify	NVARCHAR(1) NULL
loccode	NVARCHAR(5) NULL
AntiPB	NVARCHAR(1) NULL
RdEntry	NVARCHAR(1) NULL
RdExit	NVARCHAR(1) NULL
BioMatric_RdEntry	NVARCHAR(3) NULL
BioMatric_RdExit	NVARCHAR(3) NULL
Authmode	NVARCHAR(1) NULL

3.6. NetXs_Trans

Description: Transaction data from NetXs system (e.g., swipe card transactions).

Columns:

Column Name	Data Type
CID	NVARCHAR(3) NULL
GtNo	NVARCHAR(2) NULL
EmpID	NVARCHAR(15) NULL
CardID	NVARCHAR(8) NULL
dt	DATETIME NULL
InOut	NVARCHAR(1) NULL
ErrDesc	NVARCHAR(150) NULL
loccode	NVARCHAR(5) NULL
downloccode	NVARCHAR(5) NULL
status	NVARCHAR(1) NULL
Tid	BIGINT NULL
sitecode	NVARCHAR(3) NULL
deptid	NVARCHAR(6) NULL
type	NVARCHAR(6) NULL
updatedon	DATETIME NULL
empname	NVARCHAR(50) NULL
location	NVARCHAR(15) NULL
VFlag	NVARCHAR(1) NULL

Indexes:

- Non-clustered index on **EmpID** including **CardID**, **dt**, **InOut**, **empname**
- Non-clustered index **NetXs_Trans_Index1** on **dt** including **EmpID**, **deptid**

3.7. DailyAttendanceHistory

Description: Historical daily attendance records.

Columns:

Column Name	Data Type
ID	BIGINT IDENTITY(1,1) NOT NULL
AttDate	DATE NOT NULL
EmpID	NVARCHAR(15) NOT NULL
EmpName	NVARCHAR(50) NOT NULL
CardID	NVARCHAR(8) NOT NULL
DeptID	NVARCHAR(6) NOT NULL
DeptName	NVARCHAR(30) NOT NULL
FirstIn	DATETIME NULL

LastOut	DATETIME NULL
CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_DailyAttendanceHistory_ID on ID
- **Unique**: PK_DailyAttendanceHistory_AttDate_EmpID on AttDate, EmpID

3.8. DebugLog

Description: Logs debugging information.

Columns:

Column Name	Data Type
ID	BIGINT IDENTITY(1,1) NOT NULL
ModuleName	NVARCHAR(255) NULL
MessageLog	NVARCHAR(MAX) NULL
LoggedInUserEmail	NVARCHAR(255) NULL
CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_DebugLog_ID on ID

3.9. DeliveryCenterMaster

Description: Master table for delivery centers.

Columns:

Column Name	Data Type
ID	BIGINT IDENTITY(1,1) NOT NULL
DeliveryCenterCode	NVARCHAR(6) NOT NULL
DeliveryCenterName	NVARCHAR(30) NOT NULL
IsActive	BIT NOT NULL (Default: 1)
CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_DeliveryCenterMaster_ID on ID

- **Unique**: UK_DeliveryCenterMaster_DeliveryCenterCode on DeliveryCenterCode

3.10. DeliveryCenterOwners

Description: Maps owners to delivery centers.

Columns:

Column Name	Data Type
ID	BIGINT IDENTITY(1,1) NOT NULL
DeliveryCenterCode	NVARCHAR(6) NOT NULL
OwnerEmail	NVARCHAR(255) NOT NULL
IsActive	BIT NOT NULL (Default: 1)
CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_DeliveryCenterOwners_ID on ID
- **Foreign Key**: FK_DeliveryCenterOwners_DeliveryCenterCode_DeliveryCenterMaster_DeliveryCenterCode referencing DeliveryCenterMaster(DeliveryCenterCode)

3.11. DeptOwnerMaster

Description: Maps owners to departments.

Columns:

Column Name	Data Type
ID	BIGINT IDENTITY(1,1) NOT NULL
DeptCode	NVARCHAR(6) NOT NULL
DeptName	NVARCHAR(30) NOT NULL
OwnerEmail	NVARCHAR(255) NOT NULL
IsActive	BIT NOT NULL (Default: 1)
CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_DeptOwnerMaster_ID on ID

3.12. DeptSubTypeMapping

Description: Maps department subtypes.

Columns:

Column Name	Data Type
DeptId	VARCHAR(50) NOT NULL
DeptName	VARCHAR(50) NULL
TypeId	VARCHAR(50) NULL

3.13. DeptwiseExpectedCount

Description: Stores expected attendance counts per department.

Columns:

Column Name	Data Type
ID	BIGINT IDENTITY(1,1) NOT NULL
DeptID	NVARCHAR(15) NOT NULL
DeptName	NVARCHAR(30) NOT NULL
ExpectedCount	BIGINT NOT NULL (Default: 0)
IsActive	BIT NOT NULL (Default: 1)
CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_DeptwiseExpectedCount_ID on ID

3.14. DimDate

Description: Dimension table for dates, including holiday and weekday information.

Columns:

Column Name	Data Type
ID	BIGINT IDENTITY(1,1) NOT NULL
AttDate	DATE NOT NULL
AttDay	NVARCHAR(10) NOT NULL
AttWeek	SMALLINT NOT NULL
AttMonth	TINYINT NOT NULL
AttQuarter	TINYINT NOT NULL

AttYear	SMALLINT NOT NULL
IsWeekDay	BIT NOT NULL
IsHoliday	BIT NOT NULL
HolidayText	NVARCHAR(30) NULL
CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_DimDate_ID on ID
- **Unique**: UK_DimDate_AttDate on AttDate

3.15. DimDate_backup

Description: Backup of the DimDate table.

Columns:

Column Name	Data Type
ID	BIGINT IDENTITY(1,1) NOT NULL
AttDate	DATE NOT NULL
AttDay	NVARCHAR(10) NOT NULL
AttWeek	SMALLINT NOT NULL
AttMonth	TINYINT NOT NULL
AttQuarter	TINYINT NOT NULL
AttYear	SMALLINT NOT NULL
IsWeekDay	BIT NOT NULL
IsHoliday	BIT NOT NULL
HolidayText	NVARCHAR(30) NULL
CreatedOn	DATETIME NOT NULL
ModifiedOn	DATETIME NOT NULL

3.16. EmpManagerMap

Description: Maps employees to their managers.

Columns:

Column Name	Data Type
EmpId	VARCHAR(10) NULL

Manager	VARCHAR(100) NULL

3.17. Intg_LastCloudSyncInfo

Description: Stores the last synchronization date with the cloud.

Columns:

- **LastCloudSyncDate** DATETIME NULL

3.18. NetXs_Emp_Copy

Description: Copy of NetXs_Emp table for testing or backup purposes.

Columns: (Same as NetXs_Emp, see 3.4)

3.19. NetXs_Emp_CopyNew

Description: Another copy of NetXs_Emp table.

Columns: (Same as NetXs_Emp, see 3.4)

3.20. NetXs_Testing_Emp

Description: Testing version of NetXs_Emp table.

Columns: (Same as NetXs_Emp, see 2496)

3.21. NetXs_Testing_Trans

Description: Testing version of NetXs_Trans table.

Columns: (Same as NetXs_Trans, see 3.6)

3.22. RoleMapping

Description: Maps roles to employees.

Columns:

Column Name	Data Type

ID	BIGINT IDENTITY(1,1) NOT NULL
RoleID	INT NOT NULL
EmployeeEmail	NVARCHAR(255) NOT NULL
IsActive	BIT NOT NULL (Default: 1)
CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_RoleMapping_ID on ID
- **Foreign Key**: FK_RoleMapping_RoleID_RoleMaster_RoleID referencing RoleMaster(RoleID)

3.23. RoleMaster

Description: Master table for roles.

Columns:

Column Name	Data Type
ID	BIGINT IDENTITY(1,1) NOT NULL
RoleID	INT NOT NULL
RoleName	NVARCHAR(100) NOT NULL
IsActive	BIT NOT NULL (Default: 1)
CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_RoleMaster_ID on ID
- **Unique**: UK_RoleMaster_RoleID on RoleID
- **Unique**: UK_RoleMaster_RoleName on RoleName

3.24. TypeMaster

Description: Master table for types (e.g., department types).

Columns:

Column Name	Data Type
TypeCode	VARCHAR(10) NOT NULL
TypeName	VARCHAR(250) NOT NULL
IsActive	BIT NOT NULL
DeptId	INT NOT NULL
CreatedOn	DATETIME NULL
ModifiedOn	DATETIME NULL

Constraints:

- **Primary Key**: PK_TypeMaster on TypeCode

3.25. Updated

Description: Temporary or transitional table for employee updates.

Columns:

Column Name	Data Type
EmpID	FLOAT NULL
Email	NVARCHAR(255) NULL
Department	NVARCHAR(255) NULL
DeptId	FLOAT NULL
Type	NVARCHAR(255) NULL
Net X Class & Practice	NVARCHAR(255) NULL

3.26. UpdatedNew

Description: Extended version of Updated table.

Columns:

Column Name	Data Type
EmpID	FLOAT NULL
Email	NVARCHAR(255) NULL

Department	NVARCHAR(255) NULL
DeptId	FLOAT NULL
Type	NVARCHAR(255) NULL
Net X Class & Practice	NVARCHAR(255) NULL
TypeId	NVARCHAR(255) NULL
F8	NVARCHAR(255) NULL
F9	NVARCHAR(255) NULL
F10	NVARCHAR(255) NULL
F11	NVARCHAR(255) NULL
F12	NVARCHAR(255) NULL

3.27. WatchListEmployees

Description: Stores employees associated with watchlists.

Columns:

Column Name	Data Type
ID	BIGINT IDENTITY(1,1) NOT NULL
WatchListID	BIGINT NOT NULL
EmployeeID	NVARCHAR(15) NOT NULL (Default: '0')
EmployeeName	NVARCHAR(255) NOT NULL
EmployeeEmail	NVARCHAR(255) NOT NULL
IsActive	BIT NOT NULL (Default: 1)
CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_WatchListEmployees_ID on ID
- **Unique**: UK_WatchListEmployees_EmployeeEmail on WatchListID, EmployeeEmail
- **Foreign Key**: FK_WatchListEmployees_WatchListID_WatchListMaster_ID referencing WatchListMaster(ID)

3.28. WatchListMaster

Description: Master table for watchlists.

Columns:

Column Name	Data Type

ID	BIGINT IDENTITY(1,1) NOT NULL
WatchListName	NVARCHAR(100) NOT NULL
WatchListDescription	NVARCHAR(255) NOT NULL (Default: '')
WatchListPrimaryOwnerName	NVARCHAR(255) NOT NULL
WatchListPrimaryOwnerEmail	NVARCHAR(255) NOT NULL
WatchListSecondaryOwnerName	NVARCHAR(255) NOT NULL (Default: '')
WatchListSecondaryOwnerEmail	NVARCHAR(255) NOT NULL (Default: '')
IsActive	BIT NOT NULL (Default: 1)
CreatedOn	DATETIME NOT NULL (Default: GETDATE())
ModifiedOn	DATETIME NOT NULL (Default: GETDATE())

Constraints:

- **Primary Key**: PK_WatchListMaster_ID on ID
- **Unique**: UK_WatchListMaster_WatchListName on WatchListName

4. Views

4.1. v_ActiveEmployees

Description: Provides a list of active employees with department details.

4.2. v_DeptMaster

Description: Provides department details from NetXs_Misc.

4.3. v_EmpMaster

Description: Provides employee details with department names.

4.4. v_SwipeCardTrans

Description: Provides swipe card transaction details with employee and gate information.

5. Stored Procedures

5.1. sp_CreateWatchList

Description: Creates a new watchlist and inserts associated employees.

Parameters:

StoredProcColumnName	DataType
@param_LoggedInUserEmail	NVARCHAR(255)
@param_WatchListName	NVARCHAR(100)
@param_WatchListDescription	NVARCHAR(255)
@param_WatchListPrimaryOwnerName	NVARCHAR(255)
@param_WatchListPrimaryOwnerEmail	NVARCHAR(255)
@param_tvp_WatchListEmployees	tvp_WatchListEmployees READONLY

5.2. sp_DeleteWatchList

Description: Soft-deletes a watchlist by setting IsActive to 0.

Parameters:

StoredProcColumnName	DataType
@param_LoggedInUserEmail	NVARCHAR(255)
@param_WatchListID	BIGINT

5.3. sp_DeliveryCenter_DailyAttendanceReport

Description: Generates a daily attendance report for a delivery center.

Parameters:

StoredProcColumnName	DataType
@param_Date	DATETIME
@param_DeliveryCenterCode	NVARCHAR(6)

5.4. sp_GetDeliveryCenterOwners

Description: Retrieves owners of a delivery center.

Parameters:

- **@param_DeliveryCenterCode** NVARCHAR(6)

5.5. sp_GetDepartments

Description: Retrieves active departments excluding the CEO department.

Parameters: None

5.6. sp_GetDeptOwners

Description: Retrieves owners of a specific department.

Parameters:

- **@param_DeptCode** NVARCHAR(6)

5.7. sp_GetDeptwiseAttendance

Description: Retrieves department-wise attendance for a given date.

Parameters:

- **@param_Date** NVARCHAR(10)

5.8. sp_GetEmpAttendance

Description: Retrieves attendance details for an employee on a specific date.

Parameters:

StoredProcColumnName	Data type
@param_EmpID	NVARCHAR(10)
@param_Date	NVARCHAR(10)

5.9. sp_GetEmployeeAttendance

Description: Retrieves employee attendance based on various parameters.

Parameters:

StoredProcColumnName	Data type
@param_OperationId	NVARCHAR(2)
@param_DeptId	NVARCHAR(10) = NULL

@param_EmpId	NVARCHAR(10) = NULL
@param_Date	NVARCHAR(10) = NULL
@param_Year	SMALLINT = NULL
@param_Month	SMALLINT = NULL

5.10. sp_GetLastSyncDateWithNetxs

Description: Retrieves the last synchronization date with Netxs.

Parameters: None

5.11. sp_GetMonthlyDeptAttendance

Description: Retrieves monthly attendance for a department.

Parameters:

StoredProcColumnName	Data type
@param_DeptId	NVARCHAR(15)
@param_Year	SMALLINT
@param_Month	SMALLINT

5.12. sp_GetUserRoles

Description: Retrieves roles for a logged-in user.

Parameters:

- **@param_LoggedInUserEmail** NVARCHAR(255)

5.13. sp_GetWatchList

Description: Retrieves watchlists for a user.

Parameters:

- **@param_LoggedInUserEmail** NVARCHAR(255)

5.14. sp_GetWatchListByID

Description: Retrieves details of a specific watchlist.

Parameters:

StoredProcColumnName	Data type
@param_LoggedInUserEmail	NVARCHAR(255)
@param_WatchListID	BIGINT

5.15. sp_IntegrateDailySwipeData

Description: Integrates daily swipe data from JSON input.

Parameters:

- **@param_swipejson** NVARCHAR(MAX)

5.16. sp_IntegrateDepartmentData

Description: Integrates department data from JSON input.

Parameters:

- **@param_deptjson** NVARCHAR(MAX)

5.17. sp_IntegrateEmployeeData

Description: Integrates employee data from JSON input.

Parameters:

- **@param_employeejson** NVARCHAR(MAX)

5.18. sp_IntegrateGateData

Description: Integrates gate data from JSON input.

Parameters:

- **@param_gatejson** NVARCHAR(MAX)

5.19. sp_Manager_DailyAttendanceReport

Description: Generates a daily attendance report for managers.

Parameters:

StoredProcColumnName	Data type
@param_Date	DATETIME
@param_DeptCode	NVARCHAR(6)

5.20. sp_PopulateDailyAttendanceHistory

Description: Populates the DailyAttendanceHistory table from transaction data.

Parameters: None

5.21. sp_SearchEmployeesByName

Description: Searches for employees by name.

Parameters:

- **@param_EmpName** NVARCHAR(100)

5.22. sp_UpdateWatchList

Description: Updates an existing watchlist.

Parameters:

StoredProcColumnName	Data type
@param_LoggedInUserEmail	NVARCHAR(255)
@param_WatchListID	BIGINT
@param_WatchListName	NVARCHAR(100)
@param_WatchListDescription	NVARCHAR(255)
@param_WatchListPrimaryOwnerName	NVARCHAR(255)
@param_WatchListPrimaryOwnerEmail	NVARCHAR(255)
@param_tvp_WatchListEmployees	tvp_WatchListEmployees READONLY

5.23. sp_WatchListDetailsForToday

Description: Retrieves watchlist details for the current day.

Parameters:

StoredProcColumnName	Data type
@param_LoggedInUserEmail	NVARCHAR(255)
@param_Date	NVARCHAR(10)

6. Indexes

Table: NetXs_Trans

- Non-clustered index on **EmpID** including **CardID**, **dt**, **InOut**, **empname**
- Non-clustered index **NetXs_Trans_Index1** on **dt** including **EmpID**, **deptid**

7. Constraints

- **Primary Keys**: Defined on all tables with identity columns (e.g., ID).
- **Unique Constraints**: Enforced on fields like DeptCode, EmpID, etc., to ensure data integrity.
- **Foreign Keys**:
  - **DeliveryCenterOwners(DeliveryCenterCode)** references **DeliveryCenterMaster(DeliveryCenterCode)**
  - **RoleMapping(RoleID)** references **RoleMaster(RoleID)**
  - **WatchListEmployees(WatchListID)** references **WatchListMaster(ID)**
- **Default Constraints**: Set for columns like **IsActive**, **CreatedOn**, and **ModifiedOn** to auto-populate values.

Setup Instructions

Prerequisites:
- Node.js & npm
- SQL Server Management Studio (SSMS)

Backend Setup:

cd nodejswebapi

npm install

Create a .env file with:

DB_SERVER=localhost

DB_USER=your_db_user

DB_PASSWORD=your_password

DB_NAME=AttendanceTracker

npm start

Frontend Setup:

cd reactuiwebapp

npm install

npm start

Backend Technical Documentation for Gyansys Attendance Application

API Documentation

Middleware

- verifyToken: Ensures a valid JWT token is present in requests.

- authController.js: Provides a `/get-token` endpoint to issue JWT tokens.

Authentication

Use `/attendance/api/get-token` to retrieve a JWT token.

Pass it in the Authorization header as `Bearer <token>`.

API Documentation for server.js

Overview

This API is part of the Attendance Tracker system, built using Node.js and Express. It provides endpoints for managing attendance data, employee information, department data, gate data, watchlists, and user roles. The API interacts with a SQL Server database via stored procedures and uses JSON Web Tokens (JWT) for authentication.

- Base URL: /attendance
- Port: Configured via the PORT environment variable (set in .env)
- Authentication: Most endpoints require a valid JWT token in the Authorization header as Bearer <token>. Tokens are generated via the /api/get-token endpoint.
- Content Type: application/json
- Database: SQL Server (via poolPromiseATDB)
- Error Handling: 500 responses with { "error": "Internal Server Error" } for unexpected failures.

Table of Contents

Authentication

Token Generation

- Endpoint: POST /attendance/api/get-token
- Returns: { token: <JWT_TOKEN> }
- Public: Yes

Usage

- Header: Authorization: Bearer <token>

Endpoints

#	Endpoint	Method	Description
1	/api/get-token	POST	Generate JWT token
2	/api/get-last-sync-date	POST	Get last sync date
3	/api/integration-daily-swipe-data	POST	Integrate swipe data
4	/api/integration-employee-data	POST	Integrate employee data
5	/api/integration-department-data	POST	Integrate department data
6	/api/integration-Gate-data	POST	Integrate gate data
7	/api/employees?name=John	GET	Search employees by name
8	/api/attendance/:empId/:date	GET	Get employee attendance
9	/api/dept?date=YYYY-MM-DD	GET	Department-wise attendance
10	/api/attendance/:empId/:year/:month	GET	Daily attendance history

11	/api/get-employee-attendance/:operationId/:date/:deptId/	GET	Param-based attendance
12	/api/get-employee-attendance?query	GET	Query-based attendance
13	/api/userroles?email=example	GET	Get user roles
14	/api/watchlist/:email/:date	GET	Get watchlist (today)
15	/api/watchlist/:email	GET	Get watchlist
16	/api/watchlist/:email/:watchListId	DELETE	Delete watchlist
17	/api/watchlistdetails/:email/:id	GET	Watchlist by ID
18	/api/watchlist/:watchlistId	PUT	Update watchlist
19	/api/watchlist/create	POST	Create new watchlist

Error Handling

- 401 Unauthorized: Invalid or missing JWT
- 400 Bad Request: Missing required parameters
- 500 Internal Server Error: Server/database issues

Frontend Technical Documentation for Gyansys Attendance Application

Table of Contents

1. Introduction

The **Gyansys Attendance Application** is a web-based solution designed to track and manage employee attendance within an organization. It provides features such as real-time attendance monitoring, watchlist management, department-wise reporting, and employee history tracking. The application is built using **React** with **Material-UI (MUI)** for the frontend, integrated with a backend API for data management, and secured using **Microsoft Azure AD (MSAL)** for authentication.

This document provides a comprehensive overview of the application's technical architecture, components, dependencies, and usage instructions.

2. System Overview

The Gyansys Attendance Application enables administrators to:

- Authenticate users via Microsoft Azure AD.
- View real-time attendance data through dashboards and tables.
- Manage watchlists for specific employees or groups.
- Generate department-wise and employee-specific attendance reports.
- Search for employees and view their attendance history.

The application is designed for scalability, maintainability, and ease of use, leveraging modern web technologies and a modular architecture.

4. Components

4.1 MFALogin.js

Purpose: Handles user authentication using Microsoft Azure AD.

Key Features:

- Initiates a redirect-based login flow using MSAL's loginRedirect method.
- Displays a welcome message and a login button for unauthenticated users.
- Uses @mui/icons-material for visual elements. **Dependencies**:
- @azure/msal-react for authentication.
- react-router-dom for navigation.
- @mui/material for UI components. **Inputs**:
- username: Determines whether to show the login button (hidden if a user is authenticated). **Outputs**:
- Redirects to the Azure AD login page upon clicking the login button.

4.2 EditWatchlistForm.js

Purpose: Allows users to edit an existing watchlist.

Key Features:

- Fetches watchlist details and employee suggestions from the backend API.
- Validates form inputs (e.g., watchlist name, description, owner, and employees).
- Supports autocomplete for selecting owners and employees using MUI's Autocomplete component.
- Submits updated watchlist data via a PUT request. **Dependencies**:
- axios for API calls.
- @mui/material for form components.

- react-router-dom for navigation and URL parameters.
- @azure/msal-react for user email retrieval. **Inputs**:
- username: Used for API authentication.
- id: Watchlist ID from URL parameters. **Outputs**:
- Updated watchlist data sent to the backend.
- Navigation to the watchlist page upon successful submission or cancellation.

4.3 Datatable.js

Purpose: Displays a tabular view of department-wise attendance data with sorting and filtering capabilities.

Key Features:

- Fetches department and watchlist data based on the selected date.
- Supports sorting on columns (e.g., Expected Count, Reported Count, Absent Count, Achievement Percentage).
- Includes tabs for switching between swipe information and watchlist views.
- Integrates with UserInformation and WatchListForAdmin components for detailed views.
- Uses a date picker (DateComponent) for selecting the attendance date. **Dependencies**:
- axios for API calls.
- @mui/material for UI components.
- react-router-dom for navigation.
- react-icons for icons.
- react-spinners for loading animations.
- dayjs for date formatting. **Inputs**:
- User email from MSAL for API authentication.
- Selected date from UserContext. **Outputs**:
- A table displaying department-wise attendance metrics.
- Links to department and employee history reports.

4.4 Dashboard.js

Purpose: Provides a comprehensive overview of attendance data with visualizations. **Key Features**:

- Similar to Datatable.js but includes additional visualizations (pie chart and bar chart) using AttendancePieChart and BarChart components.
- Displays department-wise attendance data with sorting and filtering.
- Includes tabs for swipe information and watchlist views.
- Supports date selection for filtering data. **Dependencies**:
- Same as Datatable.js, plus D3.js for chart rendering. **Inputs**:
- User email and selected date from UserContext. **Outputs**:

- A dashboard with tables and charts summarizing attendance data.

4.5 Updatepage.js

Purpose: A placeholder page indicating that a feature is under development. **Key Features**:

- Displays a "Coming Soon" message with a button to navigate back to the home page. **Dependencies**:
- @mui/material for UI components.
- react-router-dom for navigation. **Inputs**: None. **Outputs**:
- A static page with a navigation button.

4.6 EmployeeHistory.js

Purpose: Displays the attendance history of a specific employee for a selected month and year. **Key Features**:

- Fetches employee attendance data based on empId, year, and month from URL parameters.
- Supports searching for employees by name with autocomplete suggestions.
- Allows filtering by year and month using a TextField with type="month".
- Displays a table with columns for date, day, type, department, employee name, first in, last out, duration, and remarks. **Dependencies**:
- axios for API calls.
- @mui/material for UI components.
- react-router-dom for navigation and URL parameters.
- dayjs for date formatting.
- react-icons for status icons. **Inputs**:
- empId, year, month, empName from URL parameters. **Outputs**:
- A table showing the employee's attendance history.

4.7 DepartmentMonthWiseReport.js

Purpose: Generates a month-wise attendance report for a specific department. **Key Features**:

- Fetches attendance data for a department based on deptId, year, and month.
- Dynamically generates table columns based on the days of the selected month.
- Supports searching for departments using AutoCompleteInput.
- Allows filtering by year and month. **Dependencies**:
- axios for API calls.
- @mui/material for UI components.
- react-router-dom for navigation and URL parameters.
- dayjs for date manipulation. **Inputs**:
- deptId, year, month from URL parameters. **Outputs**:

- A dynamic table showing employee attendance for each day of the month.

4.8 UserInformation.js

Purpose: Displays detailed information about a selected employee, including swipe data. **Key Features**:

- Shows employee details (name, ID, email, department, gender) and swipe information (in/out times, floor/door).
- Highlights absent status with a red background.
- Includes links for emailing or chatting with the employee via Microsoft Teams.
  Dependencies:
- @mui/material for UI components.
- react-icons for icons.
- react-spinners for loading animations. **Inputs**:
- selectedItem: Employee data.
- selectedItemAllEntries: Array of swipe entries.
- error, fetchHistoryError, employeeDetailsLoading: For error handling and loading states.
  Outputs:
- A card displaying employee and swipe information.

4.9 Loginpage.js

Purpose: Provides a fallback login page for manual authentication (not using Azure AD). **Key Features**:

- Accepts name, email, and password inputs.
- Validates credentials against hardcoded values stored in UserContext.
- Stores authentication status in localStorage. **Dependencies**:
- @mui/material for UI components.
- react-router-dom for navigation. **Inputs**:
- User input for name, email, and password. **Outputs**:
- Navigation to the home page upon successful login or an error alert for invalid credentials.

4.10 DepartmentDaywiseReport.js

Purpose: Displays a day-wise attendance report for a specific department. **Key Features**:

- Fetches attendance data for a department based on operationId, date, and departmentId.
- Displays a table with columns for date, day, type, department, employee name, first in, last out, duration, and remarks. **Dependencies**:
- @mui/material for UI components.
- react-router-dom for navigation and query parameters.
- dayjs for date formatting.

- react-icons for status icons. **Inputs**:
- operationId, date, departmentId, deptName from query parameters. **Outputs**:
- A table showing attendance details for the department on the specified date.

4.11 WatchListForAdmin.js

Purpose: Displays watchlist data in an accordion-based table format. **Key Features**:

- Organizes watchlists into expandable accordions by name.
- Shows employee details (name, in-time, out-time) with links to email and Microsoft Teams.
- Uses MUI Accordion and Table components for a structured view. **Dependencies**:
- @mui/material
- react-icons**Inputs**:
- groupedWatchlist: Object containing watchlists grouped by name.
- selectedItem: Currently selected employee (optional). **Outputs**:
- A scrollable table within accordions displaying watchlist data.

4.12 Watchlistform.js

Purpose: Allows creation of new watchlists. **Key Features**:

- Provides a form for entering watchlist name, description, owner, and employees.
- Uses MUI Autocomplete for owner and employee selection.
- Validates inputs and displays errors.
- Shows success/error notifications via Snackbar.
- Submits data via a POST request. **Dependencies**:
- axios
- @mui/material
- react-router-dom
- @azure/msal-react**Inputs**:
- username: User identifier for API authentication. **Outputs**:
- New watchlist created and saved to the backend.
- Navigation to the watchlist page on success.

4.13 Watchlist.js

Purpose: Manages the list of watchlists with options to edit or delete. **Key Features**:

- Fetches and displays all watchlists for the authenticated user.
- Provides buttons to edit or delete watchlists.
- Uses a confirmation dialog for delete actions.
- Includes a link to create a new watchlist. **Dependencies**:
- axios

- @mui/material
- react-router-dom
- @azure/msal-react
- react-icons**Inputs**:
- User email from MSAL. **Outputs**:
- A table listing watchlists with action buttons.

Common Folder

4.14 Tabs.js

Purpose: Renders a tab panel for switching between content views.
Key Features:

- Conditionally displays content based on the active tab index.
- Uses MUI Box for layout and accessibility attributes for tab navigation.
  Dependencies:
- @mui/material
- prop-types
  Inputs:
- children: Content to display.
- value: Current tab index.
- index: Tab panel index.
  Outputs:
- A hidden or visible tab panel based on the active tab.

4.15 Navbar.js

Purpose: Provides a navigation bar with links and user information.
Key Features:

- Displays the application title, navigation links (Home, Admin, Report), and user name.
- Includes a dropdown menu for admin-related pages (Configuration Master, Admin Master, Department Master).
- Conditionally shows a watchlist link based on UserContext.
  Dependencies:
- @mui/material
- react-router-dom
- @azure/msal-react

- react-icons

Inputs:

- User data from UserContext and MSAL.

Outputs:

- A responsive navigation bar with links and user details.

4.16 AutoCompleteInput.js

Purpose: Provides an autocomplete input field with action buttons.

Key Features:

- Supports searching with autocomplete suggestions.
- Includes customizable buttons (Fetch, Search, Fetch History, Clear) based on props.
- Uses MUI Autocomplete for suggestions and input handling.

Dependencies:

- @mui/material

Inputs:

- query, setQuery, suggestions, handleSearch, handleFetchHistory, handleReset, label, isFetch, isSearch, isFetchHistory, isClear.

Outputs:

- An autocomplete input with action buttons.

4.17 Layout.js

Purpose: Defines the overall page layout with a navbar, content area, and footer.

Key Features:

- Structures the application with a fixed-height navbar and footer.
- Ensures the content area is responsive and fills available space.

Dependencies:

- @mui/material

Inputs:

- children: Main content of the page.

Outputs:

- A consistent layout with navbar, content, and footer.

4.18 DynamicTable.js

Purpose: Renders a sortable table for dynamic data.

Key Features:

- Supports sorting on columns using MUI TableSortLabel.
- Displays a loading animation (FadeLoader) when no data is available.

- Dynamically renders columns and rows based on props.

Dependencies:

- @mui/material
- react-spinners

Inputs:

- columns: Array of column definitions.
- data: Array of row data.

Outputs:

- A sortable table or loading animation.

4.19 Footer.js

Purpose: Displays a footer with copyright information.

Key Features:

- Shows the current year and company name from constraints.
- Uses a custom SCSS stylesheet for styling.

Dependencies:

- @mui/material

Inputs: None.

Outputs:

- A footer with copyright text.

4.20 DateComponent.js

Purpose: Provides a date picker for selecting dates.

Key Features:

- Uses MUI DatePicker with dayjs for date handling.
- Supports customizable views (e.g., year, month, day).
- Renders a compact date picker with no underline styling.

Dependencies:

- @mui/material
- @mui/x-date-pickers
- dayjs

Inputs:

- value: Selected date.
- onchange: Callback for date changes.
- views: Array of allowed views (optional).

Outputs:

- A date picker component.

src/services/TokenServices.js

Purpose:

This module handles the secure retrieval and management of an API token for authenticated requests to the attendance tracker backend. It generates a timestamp-based HMAC signature and stores the retrieved token in localStorage while also configuring Axios for future authenticated API calls.

Token Retrieval Logic

Function: getToken

Description:

Fetches an authentication token from the backend using a timestamp and HMAC signature for request validation.

Steps:

Generate Timestamp:

timestamp = Date.now().toString()

Generate HMAC Signature:

Uses HMAC-SHA256 algorithm with the API ID as the secret to generate a secure signature:

```
CryptoJS.HmacSHA256(timestamp, secret)
```

POST Request to Backend: Sends the timestamp in the request body and the HMAC signature in the header:

```
POST /get-token

Headers:

X-Signature: <HMAC_SIGNATURE>
```

Token Storage & Configuration:

Saves the received token in localStorage under the key apiToken.

Sets the default Authorization header for Axios using this token.

Error Handling: Logs and rethrows any error encountered during the process.

Dependencies:

axios – For HTTP requests.

crypto-js – For HMAC-SHA256 signature generation.

Environment Variables Required:

REACT_APP_API_ID – The API secret key used to generate the HMAC signature.

REACT_APP_ATTENDANCE_TRACKER_API_URL – The base URL of the attendance tracker API.

src/routes/ApplicationRoutes.js

Purpose:

Defines all client-side routes for the React single-page application using React Router v6. This routing setup maps URLs to specific UI components and handles route-based access.

Core Features:

1. Dynamic Route Management

Uses <Routes> and <Route> from react-router-dom to render components based on URL paths.

Supports both static and dynamic routes (e.g. /EmpHistory/:empId/:year/:month).

2. Context Integration

Retrieves username, userRoles, and isAutheriseUser from UserContext using useContext.

Passes username and userRoles as props to route components where needed.

3. Error Handling

Wraps the entire routing configuration in an ErrorBoundary to catch and display any runtime UI errors.

4. Theme Management

Uses @emotion/react's <ThemeProvider> to apply specific themes to pages like:

DepartmentMonthWiseReport , Watchlistform

Route Overview

Path	Component	Features
/	Datatable	Default landing route
/dashboard	Dashboard	Accepts username
/EmpHistory/:empId/:year/:month	EmployeeHistory	Dynamic employee report
/watchlist	Watchlist	Personalized with username
/watchlistform	Watchlistform with Theme	Accepts username, userRoles
/watchlistform/:id	EditWatchlistForm	Edit functionality
/Updatepage	Updatepage	For user updates
/EmployeeStatus	EmployeeStatus	Static route
/DepartmentDayWiseReport	DepartmentDayWiseReport	Static route
/DepartmentMonthWiseReport/:operationId/:deptId/:year/:month/:subDeptId	DepartmentMonthWiseReport with Theme	Deep dynamic route
/DepartmentMonthWiseReport/:operationId/:deptId/:year/:month	OldDeptMonthWiseReport with Theme	Alternative view
/Login	Loginpage	Accepts username
*	PageNotFound	Catch-all route for 404s

constraints/index.js

Purpose:

This file defines a centralized object named constraints that stores static content strings used across the application, such as labels, button texts, and UI messages. This promotes maintainability and consistency in the UI/UX.

Key Features:

Centralized Text Management

All static strings for navigation bar, footer, data table, and other UI elements are stored in one place, making it easier to update or internationalize the app.

Readability & Maintainability

By using structured keys (e.g., DATATABLE.BUTTON.SEARCH), the code becomes more readable and reduces hard-coded values across components.

Testing

Strategy:
- Manual testing with frontend and Postman
- Unit tests (to be added in future versions)

Tools:
- Postman (for API testing)
- Jest (optional for unit tests)

Deployment

Suggested Stack:
- Frontend: Azure Cloud

- Backend: Azure Cloud

CI/CD (Optional):
- GitHub Actions for automated testing/deployment

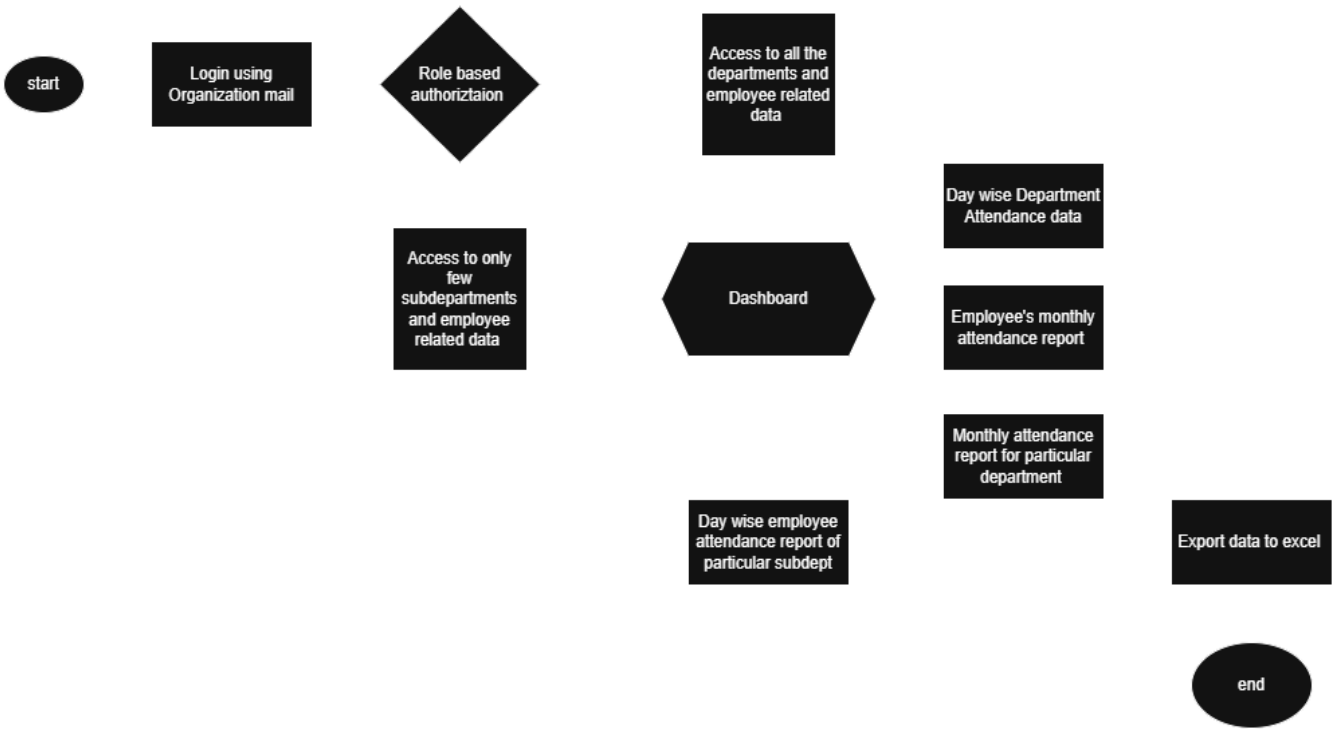
Troubleshooting & FAQ

Q1: SQL Server connection error?
- Check your .env file for correct DB credentials
- Ensure SQL Server is running and accessible on specified port

Q2: CORS issue when calling API from frontend?
- Ensure CORS is enabled in server.js

Q3: Getting 401 Unauthorized?
- Ensure you're including the correct JWT token in the request headers

User Flow Diagram


start
Login using Organization mail
Role based authoriztaion
Access to all the departments and employee related data
Access to only few subdepartments and employee related data
Dashboard
Day wise Department Attendance data
Employee's monthly attendance report
Monthly attendance report for particular department
Day wise employee attendance report of particular subdept
Export data to excel
end