Geo spread: Time Zone Handling

✅ Objective:

To **standardize time capture and display** across all geographies, ensuring consistent and accurate tracking of HR events (like leave, approvals, onboarding) regardless of user location.

🔹 .NET

What to do:

- Use `DateTime.UtcNow` in:
  - API calls
  - Data persistence (SQL Server, Dataverse)
- Store **all system timestamps in UTC**.

How it helps:

- Ensures a single source of truth for all time-based data.
- Prevents confusion in global systems with users in different time zones.

- Provides consistent backend time logic for workflows and reports.

🔹 Power Automate

What to do:

- Use the `convertTimeZone()` function to:
  - Convert UTC to **user's local time** before sending emails, setting deadlines, etc.
- Always **input UTC timestamps**, convert only for **presentation**.

How it helps:

- Sends accurate local time in notifications, approvals, and reminders.
- Prevents SLA breaches and deadline misalignments.
- Supports global teams working in different time zones.

🔹 Power Apps

What to do:

- Fetch user time zone from:
    - Azure AD
    - SharePoint User Profile
- Use time zone metadata to format and display date-time fields.
- Internally **store timestamps in UTC** using `Patch()` or forms.

How it helps:

- Ensures users see correct times based on their region.
- Prevents incorrect input/display of event times (e.g., interview at 9 AM).
- Maintains consistency between UI and stored data.

🔹 SharePoint

What to do:

- Use SharePoint User Profile Service to fetch time zone settings for users.
- Ensure SharePoint-based data entry also logs in **UTC** (e.g., via Power Automate).

How it helps:

- Centralizes user time zone data.
- Supports accurate scheduling, reminders, and task tracking on lists and calendars.

🔹 Power BI

What to do:

- Model and visualize time-based data using **UTC as the base**.
- Allow dynamic conversion to **viewer’s time zone** in visuals using filters/slicers.

How it helps:

- Presents accurate dashboards globally.
- Prevents misleading timelines or event comparisons across time zones.
- Improves analytical clarity and SLA compliance tracking.

✅ System-wide Benefits

- Prevents **workflow errors**, missed deadlines, and **attendance misinterpretation**.
- Supports **global HR operations** with uniform timestamp logic.
- Increases **data integrity and SLA compliance**.
- Minimizes manual adjustments or timezone-specific bugs in reports and workflows.

Role-Based Access Control (RBAC)

✅ Objective:

To ensure **granular and secure access** to HRIS components and data, allowing users to interact only with information and actions appropriate to their role.

🔹 SharePoint

What to do:

- Define **SharePoint Groups** or use **Microsoft 365 Security Groups**.

- Apply permissions at:
  - **Site level** (e.g., HR portal)
  - **List/library level** (e.g., performance reviews)
  - **Item level** (e.g., individual employee records)
- Align groups with roles defined in the RBAC matrix.

How it helps:

- Prevents unauthorized access to sensitive HR documents and data.
- Allows fine-grained control over visibility and actions.
- Ensures compliance with internal data protection policies.

🔹 .NET

What to do:

- Implement **middleware** to enforce RBAC on each API request.
- Use **claims-based identity** (via Azure AD or OAuth tokens) to extract user roles.

- Check roles against an **RBAC matrix** to control access to endpoints, data, and operations.

How it helps:

- Centralized, backend-enforced security logic.
- Stops unauthorized data access even if UI is bypassed.
- Enables secure, scalable integration with external systems.

🔹 Power Apps

What to do:

- Use role-checking functions like:
    - `User().Email`
    - `Office365Users.MyProfile().JobTitle`
- Dynamically control visibility, access, and editability:
    - Show/hide buttons
    - Enable/disable forms

    - Restrict data views
- Integrate with security groups or custom roles in Dataverse.

How it helps:

- Ensures that users only see and interact with content relevant to their role.
- Prevents accidental data exposure on the frontend.
- Enhances UX by removing irrelevant or unauthorized elements.

🔹 Power Automate

What to do:

- Branch logic in flows based on user role.
- Route approvals or data only to **authorized users**.
- Check permissions before sending sensitive data or triggering updates.

How it helps:

- Prevents misrouted approvals or leaks of confidential data.

- Supports compliance-driven workflows.
- Adapts business logic based on organizational hierarchy.

🔹 Power BI

What to do:

- Implement **Row-Level Security (RLS)** based on roles (from Azure AD or Dataverse).
- Filter data dynamically based on viewer’s role or department.
- Hide sensitive dashboards or metrics from unauthorized viewers.

How it helps:

- Ensures secure and relevant data insights per role.
- Protects confidential HR metrics (e.g., salaries, attrition).
- Reduces information overload by tailoring views.

✅ System-wide Benefits

- **Protects sensitive HR data** like compensation and exit records.
- Meets **regulatory and compliance** requirements (e.g., GDPR).
- Prevents **access violations** and **data breaches**.
- Builds a **trustworthy and secure environment** for users.

Department Segregation and Workflow Customization

✅ Objective:

To implement **department-specific data visibility, workflows, and UI logic** across the HRIS so that each department (e.g., Finance, IT, Sales, HR) sees and interacts only with relevant content and processes.

🔹 SharePoint

What to do:

- Add **Department metadata columns** to key lists/libraries (e.g., employee files, forms).

- Use **folders** or **custom views** filtered by department metadata.
- Organize document templates and records by department.

How it helps:

- Simplifies access and navigation for department users.
- Ensures users only see documents relevant to their department.
- Reduces clutter and improves performance in large lists/libraries.

🔹 .NET

What to do:

- Use **authenticated user’s claims/roles** to determine department context.
- Apply **department-based filtering** in API queries (e.g., only return Sales data for Sales users).
- Secure endpoints so users cannot query other departments’ data.

How it helps:

- Enforces backend-level data security and isolation.

- Improves performance by fetching only relevant records.
- Enables scalable, modular service architecture for departmental logic.

🔹 Power Apps

What to do:

- Include **Department field** in all relevant data sources (employee profiles, forms, workflows).
- Use `If()` and `Filter()` functions in galleries and forms to:
    - Show only department-specific records.
    - Dynamically change UI elements based on department.
- Use `Office365Users.MyProfile().Department` or custom lookup for department detection.

How it helps:

- Tailors the app interface to each department's needs.
- Prevents users from accessing or editing unrelated data.
- Enhances usability and reduces user error.

🔹 Power Automate

What to do:

- Implement **conditional branches** in flows based on department metadata.
- Use **department-specific approval chains**, notification logic, or templates.
- Route requests and updates based on department logic.

How it helps:

- Enables highly customized workflows for each department.
- Reduces complexity by modularizing flows.
- Improves maintainability and scalability of business logic.

🔹 Power BI

What to do:

- Filter dashboards by **Department metadata** using slicers or RLS (Row-Level Security).
- Create **department-specific reports** for performance, HR metrics, etc.

How it helps:

- Provides personalized insights to each department.
- Prevents data overload and information leaks.
- Improves clarity and decision-making.

✅ System-wide Benefits

- Allows departments to **operate independently** while staying on a unified HRIS.
- Enhances **performance and scalability** through filtering and modularization.
- Delivers a **personalized, role-aware user experience**.
- Simplifies future enhancements by avoiding reengineering.
