# Time Off Management Application – Complete Documentation

## Application Overview

- **Scope:** `x_1554358_time_off`
- **Application Name:** Time Off Management
- **Purpose:** Custom employee time off request system with automated balance tracking, approval processing, and Service Portal integration

### Key Components Involved

- **UI Action:** Submit for Approval
- **Business Rules:** Validation and balance updates
- **Script Include:** Balance retrieval
- **Flow Designer / Approval Engine:** Approval routing

---

## System Architecture

### Core Tables

**1. Time Off Request (x_1554358_time_off_time_off_request)**
**Purpose:** Stores all employee time off requests.

**Key Fields:**

- `number` – Auto-generated request number
- `requested_for` – Reference to user
- `time_off_type` – Choice: Annual, Sick
- `request_state` – Requested, Approved, Rejected
- `start_date` / `end_date` – Requested time off period
- `total_hours` – Total requested hours
- `available_balance` – Available balance for vacation
- `balance_after_request` – Calculated remaining balance

**2. Time Off Balance (x_1554358_time_off_balance)**
**Purpose:** Stores current available balances per user.

**Key Fields:**

- `user` – Reference to `sys_user`
- `vacation_balance` – Annual leave hours
- `sick_balance` – Sick leave hours
- `last_updated` – Last updated timestamp

**3. Time Off Request Day (x_1554358_time_off_time_off_request_day)**

**Purpose:** Stores individual day-level entries for each time off request.

**Key Fields:**

- `request` – Reference to `time_off_request` record (`number` is the display value)
- `date` – Requested day
- `hours` – Hours requested for that day
- `display_name` – Generated display value based on user and requested hours

---

## Server-Side Logic

### Script Include

#### GetTimeOffBalance

- **Type:** Client Callable (GlideAjax)

**Purpose**

- Retrieves the current user's time off balances from the Time Off Balance table
- Provides balance data to client-side scripts via GlideAjax
- Returns:

    - vacation balance
    - sick balance
- Returns values in JSON format

**Logic**

- Defines a client-callable function: `getBalance()`
- Initializes a result object:
    - `vacation`
    - `sick`
- Queries the Time Off Balance table:
    - filters where `user = gs.getUserID()` (current logged-in user)
- If a matching record is found:
    - retrieves:
        - `vacation_balance`
        - `sick_balance`
    - converts values to strings
    - stores values in the result object
- Returns the result as a JSON string

**Notes**

- Extends `AbstractAjaxProcessor`, allowing it to be called from client scripts via GlideAjax
- Marked as client callable, enabling secure server-side data access from the UI
- Uses `gs.getUserID()` to ensure only the current user's data is retrieved
- Returns data in JSON format, which must be parsed on the client side
- Designed specifically for use by the Populate Time Off Balance Catalog Client Script

**Why This Matters**

- Separates server-side data retrieval from client-side logic
- Improves performance by avoiding full page reloads
- Ensures secure access to balance data without exposing database queries to the client
- Promotes reusable, modular design across the application

**Technical Note**

This Script Include acts as a reusable service layer for retrieving user-specific balance data, demonstrating separation of concerns between client and server logic.

---

## Business Rules

### 1. Deduct Balance on Approval

- **Table:** Time Off Request
- **When:** After Update

#### Condition

- State changes to Approved

#### Purpose

- Updates the Time Off Balance table
- Deducts approved hours from the correct balance

#### Logic

- Identifies the associated Time Off Balance record for the employee
- Determines request type:
  - Annual → subtract from `vacation_balance`
  - Sick → subtract from `sick_balance`
- Deducts the total approved hours from the appropriate balance field

**Notes**

- Runs after update to ensure the request is fully approved before modifying balances
- Assumes a valid Time Off Balance record exists for the user
- Uses GlideRecord to locate the Time Off Balance record
- Performs conditional logic based on request type
- Updates balance fields and saves the record

---

## 2. Auto Create Request Days

- **Table:** Time Off Request
- **When:** After Insert, Update

**Condition**

- Request is in Draft state
- Both Start Date and End Date are populated
- On update, only runs if Start Date or End Date changes

**Purpose**

- Automatically generates individual Time Off Request Day records

- Ensures each applicable workday within the request range is tracked
- Keeps child records synchronized when date ranges change

**Logic**

- Converts start and end dates into comparable string values (`YYYY-MM-DD`)
- Cleanup step:
  - finds existing request day records linked to the request
  - deletes any records that fall outside the updated date range
- Generate new records:
  - loops through each day between start and end dates
  - skips weekends (Saturday and Sunday)
  - for each valid workday:
    - checks if a record already exists
    - if not, creates a new Time Off Request Day record:
      - sets request reference
      - sets date
      - defaults `hours = 8`

**Notes**

- Uses midday time (`12:00:00`) when iterating dates to avoid timezone rollover issues
- Prevents duplicate records by checking existing entries before insert
- Automatically adjusts child records when the request date range is modified
- Assumes a standard 8-hour workday for each generated day
- Weekend exclusion is based on instance configuration (`6 = Saturday`, `7 = Sunday`)

**Technical Note**
This logic mirrors a composite primary key uniqueness pattern by ensuring one record per request per date, similar to enforcing a `(request, date)` constraint in relational databases.

---

### 3. Enforce Time Off Request Days

- **Table:** Time Off Request
- **When:** Before Update

**Condition**

- Request is transitioning into the approval lifecycle
- State changes to Requested or Pending Approval

**Purpose**

- Prevents submission of a Time Off Request without any associated request days
- Ensures required child records exist before approval processing begins

**Logic**

- Detects when the request is moving into a submission/approval state
- Queries the Time Off Request Day table for records linked to the request
- Counts the number of associated day records
- If no records exist:
    - displays an error message to the user
    - aborts the update to prevent submission

**Notes**

- Uses GlideAggregate for efficient record counting
- Runs before update to block invalid submissions before they are saved
- Ensures parent-child data integrity between:
    - Time Off Request (parent)

    - Time Off Request Day (child)
- Prevents edge cases where users attempt to submit requests without defined dates

**Technical Note**
This rule enforces a parent-child dependency similar to requiring related records before allowing a transaction to proceed, ensuring referential completeness at the application level.

———

### 4. Prevent Overlap on Submit

- **Table:** Time Off Request
- **When:** Before Update

**Condition**

- Request state changes out of Draft
- New state is one of:
    - Requested
    - Pending Approval
    - Approved

**Purpose**

- Prevents a user from submitting overlapping time off requests
- Ensures the same employee does not have multiple requests covering the same day during active request states

**Logic**

- Verifies the request is moving out of Draft into a submission or approval state
- Queries all Time Off Request Day records for the current request
- Builds a list of requested dates
- Searches for other Time Off Request Day records that:
  - match any of the same dates
  - belong to a different request
  - belong to the same `requested_for` user
  - are tied to requests in active states:
    - Requested
    - Pending Approval
    - Approved
- If a match is found:
  - shows an error message with the conflicting date
  - aborts the update

**Notes**

- Runs before update so invalid requests are blocked before submission is saved
- Uses dot-walking to check related parent request fields:
  - `request.requested_for`
  - `request.request_state`
- Only checks overlap against requests that are still active in the approval lifecycle
- Ignores the current request itself by excluding matching `sys_id`
- Uses `setLimit(1)` for efficiency since only one conflict is needed to stop submission

**Why This Rule Matters**

- Prevents duplicate or conflicting time off entries for the same employee
- Enforces scheduling integrity at the application level
- Helps avoid approval errors and inaccurate balance usage

**Technical Note**

This rule functions similarly to a uniqueness check across `(requested_for, date)` for active requests, even though the

platform stores the relationship through child day records rather than a composite database key.

---

### 5. Prevent Overlapping Time Off

- **Table:** Time Off Request Day
- **When:** Before Insert, Update

#### Condition

- `request` is populated
- `date` is populated

#### Purpose

- Prevents duplicate time off day entries for the same user on the same date
- Ensures a user cannot have overlapping active requests at the individual day level

#### Logic

- Verifies the current Time Off Request Day record has both:
  - a parent request
  - a date

- Retrieves the parent Time Off Request record
- Identifies:
  - the `requested_for` user from the parent request
  - the selected request day date
- Queries other Time Off Request Day records that:
  - have the same date
  - belong to the same user
  - are linked to requests in active states:
    - Requested
    - Pending Approval
    - Approved
  - are not the current record
- If a matching record exists:
  - displays an error message
  - aborts the insert or update

**Notes**

- Runs at the child table level, which provides a more granular overlap check than the parent request rule

- Uses the parent request to determine the employee associated with the request day
- Prevents conflicts even if request day records are manually inserted or modified outside the normal request submission flow
- Excludes the current record using `sys_id` to avoid false positives during updates
- Only blocks overlaps against requests that are still active in the approval lifecycle

**Why This Rule Matters**

- Adds a second layer of protection against duplicate or conflicting time off dates
- Helps preserve scheduling accuracy even when child records are edited directly
- Supports data integrity independently of the parent request submission process

**Technical Note**

This rule acts like an application-level uniqueness check on time off days for a given user and date, using related request state and employee data through dot-walking.

---

## 6. Recalculate Total Hours

- **Table:** Time Off Request Day
- **When:** After Insert, Update, Delete

**Condition**

- A related parent Time Off Request record exists

**Purpose**

- Recalculates the total requested hours whenever request day records are added, modified, or removed
- Keeps the parent request's `total_hours` field synchronized with its child day entries

**Logic**

- Determines the parent request ID:
  - uses `current.request` for insert and update
  - uses `previous.request` for delete
- Queries all Time Off Request Day records linked to that request
- Sums the values in the `hours` field
- Retrieves the parent Time Off Request record
- Updates the parent record's `total_hours` field with the calculated sum

**Notes**

- Runs after database changes so the recalculation reflects the latest inserted, updated, or deleted child record state
- Uses GlideAggregate with `SUM(hours)` for efficient total calculation
- Supports delete operations by using the parent reference from `previous` when `current` is no longer available
- Maintains parent-child consistency between:
  - Time Off Request
  - Time Off Request Day
- If no child day records remain, the total is set to `0`

**Why This Rule Matters**

- Prevents manual mismatches between request days and total requested hours
- Ensures the parent request always reflects the true sum of its child records
- Simplifies reporting, approvals, and balance deduction logic by keeping `total_hours` accurate

**Technical Note**

This rule functions like a roll-up calculation, where child row values are

aggregated and stored on the parent record for easier access and processing.

**Maintenance Note**
Updating the parent request may trigger additional business rules on the Time Off Request table, so related rule interactions should be considered during maintenance.

———

### 7. Set Available Balance

- **Table:** Time Off Request
- **When:** Before Update

**Condition**

- `total_hours` is populated

**Purpose**

- Retrieves the employee's current available balance for the selected time off type
- Calculates the projected balance remaining after the request is applied
- Keeps balance-related fields on the request current before the record is saved

**Logic**

- Checks that the request has a `total_hours` value
- Queries the Time Off Balance table for the record associated with the `requested_for` user
- Determines which balance field to use based on `time_off_type`:
  - Annual → `vacation_balance`
  - Sick → `sick_balance`
- Reads the current available balance from the balance record
- Reads the requested total hours from the request
- Sets:
  - `available_balance = current balance`
  - `balance_after_request = current balance − requested hours`

**Notes**

- Runs before update so calculated balance values are stored on the request before save
- Uses the employee's Time Off Balance record as the source of truth for available hours
- Supports different balance calculations depending on request type

- Does not directly validate whether the request exceeds available balance; it calculates and displays the values for downstream use
- If no matching balance record is found, the fields are not updated

**Why This Rule Matters**

- Gives users and approvers visibility into available balance at the time of request processing
- Supports approval decisions by showing how the request impacts remaining time off
- Keeps calculated balance fields aligned with the employee's current balance record

**Technical Note**

This rule behaves like a pre-save calculation step, pulling related balance data into the request record so it is immediately available for forms, approvals, and reporting.

---

## 8. Set Display Name

- **Table:** Time Off Request Day
- **When:** Before Insert, Update

**Condition**

- `request` is populated

**Purpose**

- Generates a readable display value for each Time Off Request Day record
- Makes child day records easier to identify in lists, related lists, and reference views

**Logic**

- Confirms the request day record has a parent request reference
- Retrieves the related Time Off Request record
- Gets the display value of the `requested_for` user from the parent request
- Gets the request day hours value
- Sets the child record's `display_name` field using a formatted label:
  - `User Name (Xh)`

**Example**

- `John Doe | yyy-mm-dd | (8h)`

**Notes**

- Runs before insert/update so the display name is populated before the record is saved
- Uses the parent request to derive the employee name rather than storing duplicate user data directly on the child table
- Improves readability of request day records in the UI
- The script retrieves the date display value, which is used in the final display name format
- If the request reference is missing or invalid, the display name is not set

**Why This Rule Matters**

- Makes child day records much easier to recognize at a glance
- Improves admin and developer usability when working with related lists and record lookups
- Provides a cleaner human-readable label than relying on raw system identifiers

**Technical Note**

This rule supports UI usability by creating a derived display field, while the actual relationship to the parent request continues to be maintained through the reference field and `sys_id`.

---

## 9. TOR - Draft Guidance

- **Table:** Time Off Request
- **When:** Display

### Condition

- `request_state` is Draft

### Purpose

- Provides on-screen guidance to users while the request is still in Draft status
- Helps users understand the next steps before submitting the request for approval

### Logic

- Checks whether the request is currently in Draft state
- If true, displays an informational message to the user with the recommended next steps:
  - review or adjust hours in Time Off Request Days
  - click Submit for Approval when ready

### Notes

- Runs on display, so the message appears when the record is opened rather than during insert or update
- Uses `gs.addInfoMessage()` to provide a non-blocking instructional message
- Improves usability by guiding the user through the request process at the correct stage
- Only appears while the request remains in Draft status

**Why This Rule Matters**

- Reduces user confusion during request entry
- Reinforces the intended workflow without requiring separate documentation or training
- Helps ensure users review child request day records before submission

**Technical Note**

This is a UI guidance rule rather than a data-validation rule. It does not modify data or block actions; it simply improves the user experience during the draft phase.

---

## 10. Validate Day Date Within Request Range

- **Table:** Time Off Request Day

- **When:** Before Insert, Update

### Condition

- `request` is populated
- `date` is populated

### Purpose

- Ensures each request day stays within the parent request's start and end date range
- Prevents duplicate day entries for the same request and date
- Blocks weekend dates from being added as request days

### Logic

- Verifies the record has both:
  - a parent request
  - a date
- Checks for an existing Time Off Request Day record with the same:
  - `request`
  - `date`
  - but a different `sys_id`

- If a duplicate exists:
  - displays an error message
  - aborts the action
- Retrieves the parent Time Off Request record
- If the parent request has no start or end date yet, does not block the save
- Compares the request day date against the parent request range:
  - if the day is before `start_date`
  - or after `end_date`
  - displays an error and aborts
- Determines the day of week for the selected date
- If the date falls on Saturday or Sunday:
  - displays an error message
  - aborts the action

**Notes**

- Runs before insert/update so invalid child day records are stopped before being saved
- Prevents duplicate day rows for the same request/date combination
- Uses the parent request as the source of truth for the allowed date

range

- Uses string comparison on `YYYY-MM-DD` values, which works correctly for date-only comparison in this format
- Uses local noon time when calculating weekday to avoid timezone rollover issues
- Adds multiple layers of validation:
  - duplicate prevention
  - date-range enforcement
  - weekend restriction

**Why This Rule Matters**

- Protects the integrity of the Time Off Request Day table
- Ensures child records stay aligned with the parent request dates
- Prevents invalid scheduling entries from being saved manually or through list editing
- Reinforces weekday-only request behavior at the record level

**Technical Note**

This rule combines several application-level validation checks that, in a traditional relational system, might be split across unique constraints, range validation, and business-process rules.

---

## 11. Validate End After Start (Server)

- **Table:** Time Off Request
- **When:** Before Insert, Update

### Condition

- Both `start_date` and `end_date` are populated

### Purpose

- Ensures the request end date is not before the start date
- Prevents invalid date ranges from being saved

### Logic

- Checks that both `start_date` and `end_date` have values
- Converts both values to GlideDateTime objects
- Compares the numeric values of each date
- If `end_date` is earlier than `start_date`:
  - displays an error message
  - aborts the insert or update

**Notes**

- Runs before insert/update to block invalid data before it is saved
- Uses `getNumericValue()` for accurate date comparison
- Handles both new records and updates to existing records
- Allows equal dates (same start and end day), which supports single-day requests

**Why This Rule Matters**

- Prevents logically invalid time off requests
- Ensures downstream logic (day generation, balance calculation, approvals) operates on valid date ranges
- Protects overall data integrity of the application

**Technical Note**

This rule enforces a basic range validation constraint similar to a database check constraint (for example, `end_date >= start_date`) at the application level.

---

## Client-Side Logic

### Client Scripts

### 1. Enforce Time Off Type

- **Type:** onSubmit
- **Field:** N/A

#### Condition

- Runs when the user submits the form

#### Purpose

- Ensures the user selects a valid Time Off Type before the record is submitted
- Prevents submission if the field is still set to the default placeholder value

#### Logic

- On form submission:
  - checks the value of `time_off_type`
- If the value is `'select'`:
  - displays an error message on the field
  - stops the form from submitting
- If a valid value is selected:

- allows submission to continue

**Notes**

- Uses `g_form.showFieldMsg()` to display a field-level error message
- Returns `false` to block submission when validation fails
- Provides immediate client-side validation before any server-side processing occurs
- This script validates the Time Off Type only; it does not set field values or call the server

**2. Validate End After Start**

- **Type:** onChange
- **Field:** End Date

**Condition**

- Runs when the End Date field value changes
- Does not run during initial form load

**Purpose**

- Validates that the selected End Date is not earlier than the Start Date

- Provides immediate feedback to the user when an invalid date range is entered

**Logic**

- Triggers when the End Date field changes
- Exits immediately if the form is still loading
- Retrieves the values of:
  - `start_date`
  - `end_date`
- Clears any existing field message on End Date
- If either date is blank, does nothing
- If `end_date` is earlier than `start_date`:
  - displays an error message on the End Date field

**Notes**

- Uses `g_form.showFieldMsg()` to display a field-level validation message
- Uses `g_form.hideFieldMsg()` to remove old messages before re-checking the dates
- Provides immediate client-side guidance, but does not block submission by itself

- This client-side check is complemented by the server-side Business Rule:
  - Validate End After Start (Server)
- Improves user experience by catching errors before submission

---

## UI Policies

### 1. Hide Work Notes for Requesters

- **Table:** Time Off Request
- **Type:** UI Policy

#### Condition

- Applies to users in the requester role/context (non-admin / non-fulfiller users)

#### Purpose

- Hides the Work Notes field from requesters
- Prevents end users from viewing or interacting with internal-only notes

#### Logic

- When the UI Policy condition is met:
  - sets the Work Notes field visibility to false
  - hides the field from the form for applicable users

**Notes**

- Uses a UI Policy Action:
  - `work_notes → Visible = false`
- Typically used to separate:
  - internal communication (Work Notes)
  - user-facing communication (for example, Additional Comments)
- Improves security and user experience by restricting access to internal fields
- Behavior depends on the defined UI Policy condition

**Why This Matters**

- Ensures internal notes remain restricted to administrators or approvers
- Prevents accidental exposure of backend or sensitive information
- Aligns with standard ServiceNow practice of separating internal vs external communication

## 2. Lock Form When Not Draft

- **Table:** Time Off Request
- **Type:** UI Policy

### Condition

- `request_state` is not Draft

### Purpose

- Prevents users from modifying key fields once the request has moved out of Draft
- Ensures data integrity after submission into the approval lifecycle

### Logic

- When the request state is no longer Draft:
  - sets the following fields to read-only:
    - `time_off_type`
    - `short_description`
    - `requested_for`
    - `start_date`

- `end_date`
- `number`
- `description`

**Notes**

- Uses multiple UI Policy Actions with `Read-Only = true`
- Applies immediately on form load and when the state changes
- Prevents edits without requiring server-side enforcement
- Focuses on restricting modification of core request details after submission
- Complements Business Rules that control state transitions and validation

**Why This Matters**

- Preserves the integrity of submitted requests
- Prevents users from changing request details after entering the approval process
- Aligns with standard workflow behavior where submitted records become locked

### 3. Make Request State Read-Only for Users

- **Table:** Time Off Request
- **Type:** UI Policy

**Condition**

- No condition (applies to all records and states)

**Purpose**

- Prevents users from manually changing the Request State field
- Ensures state transitions are controlled by system logic (for example, Business Rules, UI Actions, or workflows)

**Logic**

- Applies the UI Policy to all records
- Sets the `request_state` field to:
  - `Visible = true`
  - `Read-Only = true`

**Notes**

- No condition means this policy is always enforced
- Prevents users from bypassing the intended workflow by manually editing the state

- State changes are expected to occur through:
    - UI Actions (for example, Submit for Approval)
    - Business Rules
    - Approval processes
- This is a UI-level restriction and does not prevent server-side updates

**Why This Matters**

- Maintains control over the request lifecycle
- Prevents invalid or unauthorized state transitions
- Ensures consistency between UI behavior and backend process logic

---

## UI Actions

### Submit for Approval

- **Table:** Time Off Request
- **Type:** UI Action (Form Button)

**Condition**

- Available when the request is in Draft state

**Purpose**

- Allows users to submit a Time Off Request into the approval process
- Transitions the request from Draft to Requested
- Triggers downstream validation and business logic

**Logic**

- When the button is clicked:
  - sets `request_state` to Requested
  - updates the current record
- Displays a confirmation message to the user:
  - “Your time off request has been submitted for manager approval.”
- Redirects the user back to the updated record

**Notes**

- Executes server-side when the form button is clicked
- Relies on Business Rules to enforce validation, including:
  - ensuring request day records exist
  - preventing overlapping requests
- Uses `current.update()` to persist the state change

- Uses `gs.addInfoMessage()` to provide user feedback
- Uses `action.setRedirectURL(current)` to return the user to the record

#### Why This Matters

- Serves as the primary entry point into the approval lifecycle
- Centralizes the state transition logic for submitting requests
- Works in conjunction with Business Rules to enforce data integrity and validation

---

## Service Catalog Integration

### Record Producer

- **Name:** Time Off Request
- **Category:** Time Off

#### Variables

- Time Off Type (Annual / Sick)
- Start Date
- End Date
- Total Hours

**Display Fields (Read-only)**

- Available Vacation Balance
- Available Sick Balance

**Purpose**

- Provides a user-friendly interface for submitting Time Off Requests
- Captures required request details before creating a record
- Displays current balance information to assist user decision-making

**Behavior / Functionality**

- When the form loads:
  - the user’s available balances are retrieved and displayed
- When submitting:
  - required fields are validated
  - a Time Off Request record is created
  - the request is initialized in Draft state

**Supporting Components**

- **Catalog Client Script (onLoad):**

    - Populates available balance fields
- **Catalog Client Script (onSubmit):**
    - Validates Time Off Type selection
- **Script Include:**
    - `GetTimeOffBalance` retrieves balance data
- **Record Producer Script:**
    - Maps variables to the Time Off Request table
    - Sets the initial request state

Balance values are retrieved dynamically using a client-server interaction (GlideAjax).

---

## Approval Process

### Flow / Process Overview

#### Trigger

- Time Off Request is updated
- Specifically when:
    - `request_state` is set to Requested via UI Action

**Purpose**

- Routes submitted Time Off Requests to the appropriate manager for approval
- Controls the request lifecycle from submission through approval or rejection

**Process Flow**

1. **User submits request**
    - User clicks the Submit for Approval UI Action
    - `request_state` is set to Requested
2. **Pre-approval validation**
    - Business Rules execute to ensure:
        - at least one request day exists
        - no overlapping requests exist
        - request data is valid
3. **Request enters approval state**
    - Flow Designer is triggered
    - Approval task is generated for the user's manager
4. **Manager reviews request**

- Manager can:
  - Approve → request moves forward
  - Reject → request is denied

5. **Post-approval processing**
   - On approval:
     - a Business Rule updates Time Off Balance
     - approved hours are deducted from the appropriate balance

**Notes**

- The approval process is initiated via a UI Action, not automatically on insert
- Business Rules enforce validation before approval begins
- Approval logic is handled through Flow Designer
- Balance updates occur only after approval to ensure accuracy

**Why This Matters**

- Ensures proper authorization before time off is granted
- Prevents invalid or conflicting requests from entering the approval process
- Maintains accurate employee time off balances

---

# Service Portal Implementation

## Portal Overview

- **Portal URL:** `/sp?id=time_off`
- Provides a centralized interface for users to:
  - submit time off requests
  - view existing requests
  - interact with time off data visually

## Portal Pages

### 1. Time Off Portal (Homepage)

- **URL:** `/sp?id=time_off`

#### Purpose

- Serves as the main entry point for the Time Off application within the Service Portal

#### Features

- Navigation tiles providing quick access to:

- My Requests
- Calendar
- New Request

**Notes**

- Designed for simple navigation and user accessibility
- Acts as the central hub for all Time Off-related actions

### 2. My Time Off Requests

- **URL:** `/sp?id=my_time_off_requests`

**Purpose**

- Displays a list of time off requests for the logged-in user

**Features**

- List view of user requests
- Clickable records for quick access
- Displays the selected request in a form view

**Notes**

- Uses a custom or configured list widget to filter records by current user
- Integrates with the Form widget for record display

#### 3. Calendar Page

- **URL:** `/sp?id=time_off_calendar`

**Purpose**

- Provides a visual representation of user time off

**Features**

- Calendar-style display of time off entries
- Clickable records for navigation
- Direct access to underlying request records

**Notes**

- Enhances usability by offering a visual alternative to list-based views
- Useful for identifying scheduling conflicts or patterns

## Widget Architecture

#### Simple List Widget

**Purpose**

- Displays time off requests for the logged-in user

**Features**

- Filters records by user
- Displays requests in a list format
- Adds URL parameters when a record is selected:
  - `sl_sys_id`
  - `sl_table`

**Notes**

- Acts as the primary navigation mechanism for selecting records
- Passes context to other widgets via URL parameters

### Form Widget

**Purpose**

- Displays the selected Time Off Request record

**Features**

- Reads URL parameters:

    - `sl_sys_id`
    - `sl_table`
- Loads and displays the corresponding record
- Updates dynamically when a new record is selected

**Notes**

- Works in conjunction with the Simple List widget
- Enables a master-detail style interface within the portal

---

## Data Flow Example

### Request Lifecycle

This flow illustrates how user input is processed through client-side validation, server-side logic, and approval workflows to produce a finalized time off request.

1. **User initiates request**
    - User accesses the Time Off Request Record Producer via Service Portal
    - Catalog Client Scripts:
        - populate available balances (GlideAjax)

    - validate required inputs

2. **Record is created (Draft state)**
    - Record Producer Script:
        - maps variables to the Time Off Request table
        - sets `request_state = Draft`
3. **Request details are generated**
    - Business Rule:
        - creates Time Off Request Day records based on the date range
    - Business Rule:
        - calculates total requested hours
4. **User submits request**
    - User clicks the Submit for Approval UI Action
    - System:
        - updates `request_state` to Requested
        - triggers validation Business Rules:
            - ensures request days exist
            - prevents overlapping requests
5. **Approval process begins**
    - Flow Designer:

    - detects the request state change
    - routes approval to the user’s manager

6. **Manager decision**
    - Manager reviews the request and:
        - Approves → request continues
        - Rejects → request is denied
7. **Post-approval processing**
    - On approval:
        - a Business Rule deducts approved hours from:
            - `vacation_balance` or `sick_balance`
        - updated balance is saved to the Time Off Balance table
8. **Record reflects final state**
    - Request remains stored with:
        - updated balance values
        - final approval status
    - Data is available in:
        - Service Portal
        - lists and reports

---

# Technical Implementation Notes

## Key Lessons Learned

### Business Rules vs Flow Designer

- **Business Rules** are best suited for:
  - data validation
  - calculations
  - enforcing data integrity
- **Flow Designer** is better suited for:
  - process automation
  - approvals and task routing

### GlideAjax Usage

- Required for retrieving server-side data from client scripts
- Enables dynamic UI behavior without page reloads
- Supports separation of concerns between:
  - client-side UI logic
  - server-side data access

### Service Portal Timing Considerations

- Catalog item variables may not be immediately available on form load
- Requires delayed execution techniques such as:
  - `setTimeout()`
  - retry/polling logic
- Important when dynamically setting field values

### Data Integrity Practices

- Always validate and sanitize numeric values:
  - use `parseInt()` or equivalent
- Handle null or undefined values safely to prevent errors
- Implement validation at multiple layers:
  - client-side (UX)
  - server-side (enforcement)

### Layered Validation Strategy

- **Client Scripts:**
  - provide immediate user feedback
- **Business Rules:**
  - enforce validation on the server

- **UI Policies:**
  - control field behavior and accessibility

---

## Security & Access Control

### Roles

#### Users

- Can create Time Off Requests via the Record Producer
- Can view and manage their own requests
- Cannot:
  - modify request state directly
  - view internal fields (for example, Work Notes)
  - edit restricted records after submission

#### Managers / Administrators

- Can review submitted requests
- Can approve or reject requests
- Can view all Time Off Request records
- Have elevated access to:

- approval actions
- administrative fields
- balance data (as applicable)

## Access Control (ACL) Considerations

### Record-Level Security

- Users are restricted to viewing only their own Time Off Requests
- Enforced through:
  - ACLs (recommended)
  - query filters / business logic

### Balance Table Protection

- Time Off Balance table is restricted to prevent direct user edits
- Balance updates are controlled exclusively through:
  - Business Rules
  - server-side logic

### Approval Permissions

- Approval actions are restricted to:
  - managers

    - authorized roles
- Prevents unauthorized users from:
    - approving
    - rejecting
    - modifying request state

**Notes**

- UI Policies are used to restrict field visibility and editability in the UI
- Business Rules enforce server-side validation and prevent unauthorized data changes
- Security is implemented in multiple layers:
    - UI (visibility and read-only controls)
    - Client Scripts (user guidance)
    - Server-side logic (enforcement)

**Security Design Approach**

- Follows a layered security model:
    - UI restrictions for usability
    - client-side validation for immediate feedback
    - server-side enforcement for data integrity

---

## Future Enhancement Opportunities

The following enhancements represent potential future improvements to extend functionality, improve usability, and support scalability of the application.

### Manager Dashboard for Approvals

- Create a dedicated dashboard for managers to:
  - view pending approvals
  - approve or reject requests in a centralized interface
- Could include:
  - summary metrics
  - quick action buttons

### Time Off Accrual Automation

- Implement automated accrual of vacation and sick time
- Use scheduled jobs or Flow Designer to:
  - increment balances based on tenure or policy
- Reduces manual balance maintenance

### Negative Balance Warnings

- Add validation to warn users when a request would result in a negative balance
- Could be implemented as:
  - client-side warning
  - server-side enforcement
- Improves user awareness and prevents invalid submissions

### Reporting and Analytics

- Develop reports and dashboards to provide insights into:
  - employee time off usage
  - department trends
  - balance utilization
- Enables better workforce planning and decision-making

### Mobile-Friendly Enhancements

- Optimize Service Portal pages for mobile devices
- Improve usability for:
  - submitting requests
  - reviewing approvals

- Enhance accessibility for users on the go

---

## Conclusion

The Time Off Management Application provides a complete, end-to-end solution for managing employee leave requests within ServiceNow. It combines:

- custom data modeling
- Business Rule–driven automation
- client-side enhancements using GlideAjax
- Service Portal–based user experience
- Flow Designer–driven approval processing

The application demonstrates practical implementation of both frontend and backend ServiceNow development concepts, including:

- client-server interaction
- workflow automation
- layered validation
- secure data handling

Overall, the system is designed to maintain data integrity, enforce business logic, and deliver a user-friendly experience for both employees and managers.