Project documentation:Calculating the family expenses using ServiceNow

Team Id: NM2025TMID17823
Team Members:

Team Leader:Munnangi jayasri.R

Team Member 1 Dhanusha.J

Team Member 2 Thilaga.M

Team Member 3 Pavithra.A

Problem Statement:

Tracking family expenses manually is time consuming and error-prone,making it hard to manage budgets effectively

Objective:

To use ServiceNow features to record,organize,and calculate family expenses in a structured and automated way

Skills:
IOT Open Hardware platforms,Data Structures

1. Introduction

This documentation provides a detailed overview of the process for building a custom application in ServiceNow to track and calculate family expenses. The application is designed to be a simple and effective tool for managing household finances by categorizing daily expenditures and totaling them within a main family expenses record. This guide covers the key development steps, including creating custom tables, establishing relationships, and implementing business logic.

2. Project Setup

This section details the initial steps taken to prepare the ServiceNow instance for development.

Step 2.1: Signing Up and Creating a Developer Instance

To begin, we created a personal developer instance (PDI) on the ServiceNow Developer Program. The PDI provides a dedicated environment for development and testing, ensuring that our project work is isolated from any production or shared instances.

Step 2.2: Creating a Custom Update Set

An **Update Set** is a container for customizations and configuration changes. To ensure that all of our project's changes could be easily tracked and moved between different instances, we created a new Update Set specifically for this project.

1. Navigate to the **Filter Navigator** and type Update Sets.
2. Under **Local Update Sets**, click **New**.
3. Fill in the details:
    - **Name:** Family Expenses App
    - **Description:** Captures all customizations for the Family Expenses application.
4. Click **Submit and Make Current** to set this as the active Update Set. All subsequent changes were automatically captured.

3. Creating Custom Tables

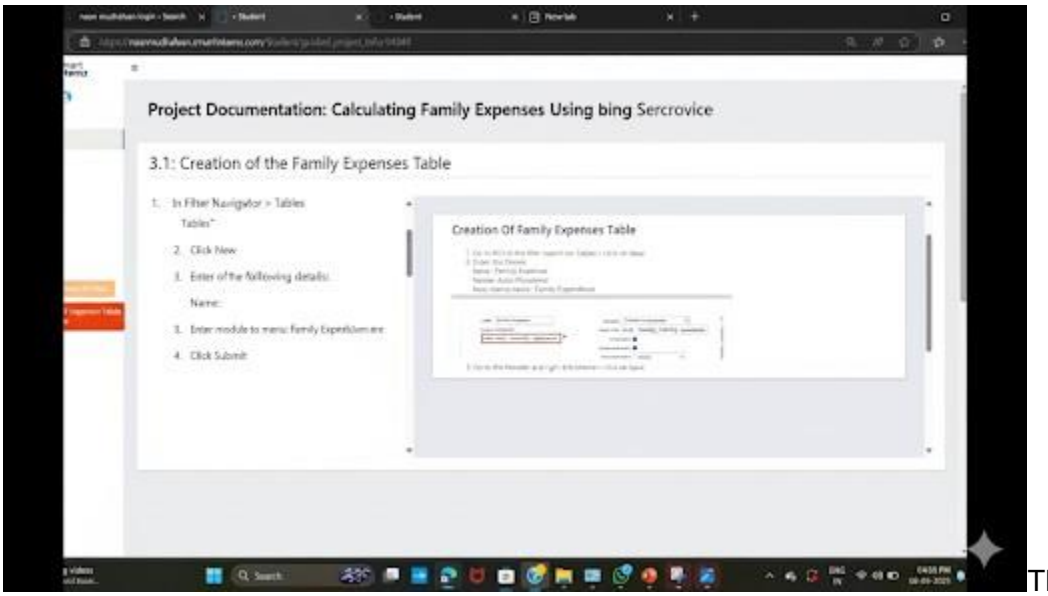
We created two custom tables to store the financial data: one for summarizing family expenses and another for logging individual daily expenditures.

Step 3.1: Creation of the Family Expenses Table

This table serves as the primary record for tracking total monthly or weekly expenses.

1. In the **Filter Navigator**, type Tables and click on **Tables** under **System Definition**.
2. Click **New** to create a new table.
3. Enter the following details:
    - **Label:** Family Expenses
    - **Name:** x_<your_scope>_family_expenses (This is auto-populated)
    - **Add module to menu:** Family Expenditure
4. Click **Submit** to create the table.

Here's a visual of the process for creating the Family Expenses Table:



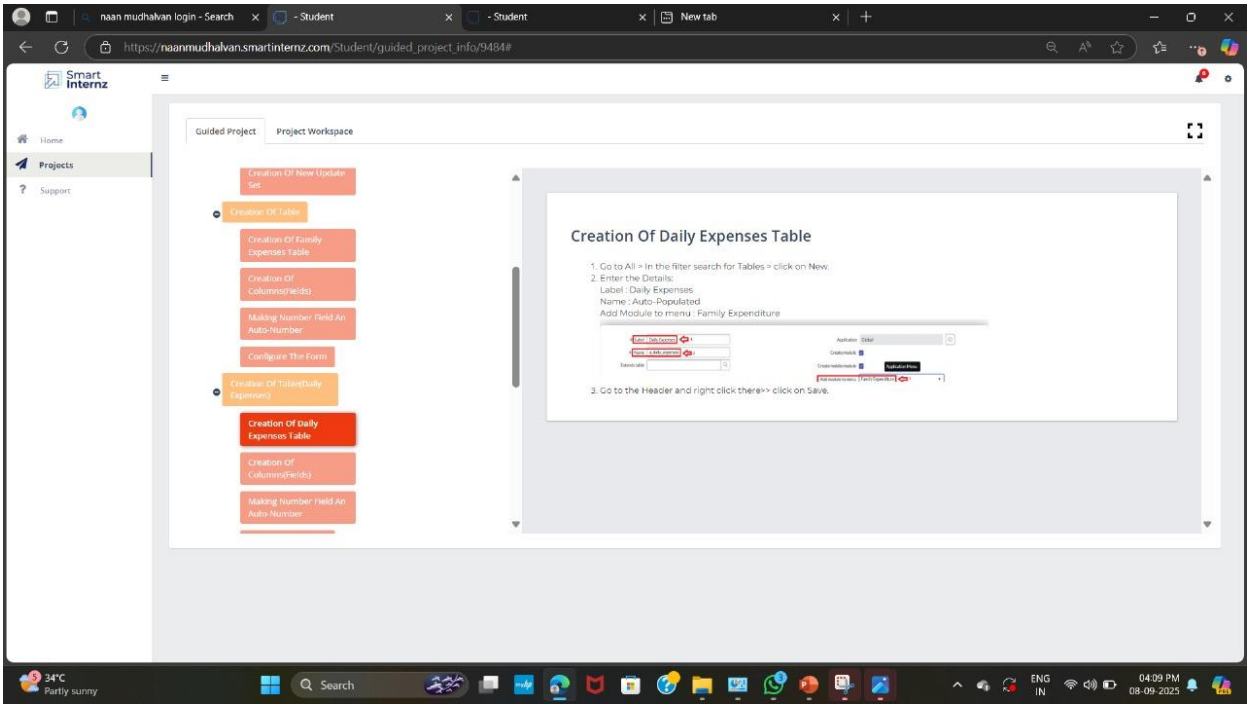
This table includes fields for different expense categories, such as Food, Rent, and Travel, allowing for a detailed breakdown of costs.

Step 3.2: Creation of the Daily Expenses Table

This table stores a list of all individual expenses incurred on a day-to-day basis.

1. In the **Filter Navigator**, type Tables and click on **Tables** under **System Definition`.
2. Click **New** to create a new table.
3. Enter the following details:
    - **Label:** Daily Expenses
    - **Name:** x_<your_scope>_daily_expenses (This is auto-populated)
    - **Add module to menu:** Daily Expenses
4. Click **Submit** to create the table.

Here's how to create the Daily Expenses Table:



This table includes fields for amount, category, and short description, providing the granular data needed to calculate the totals.

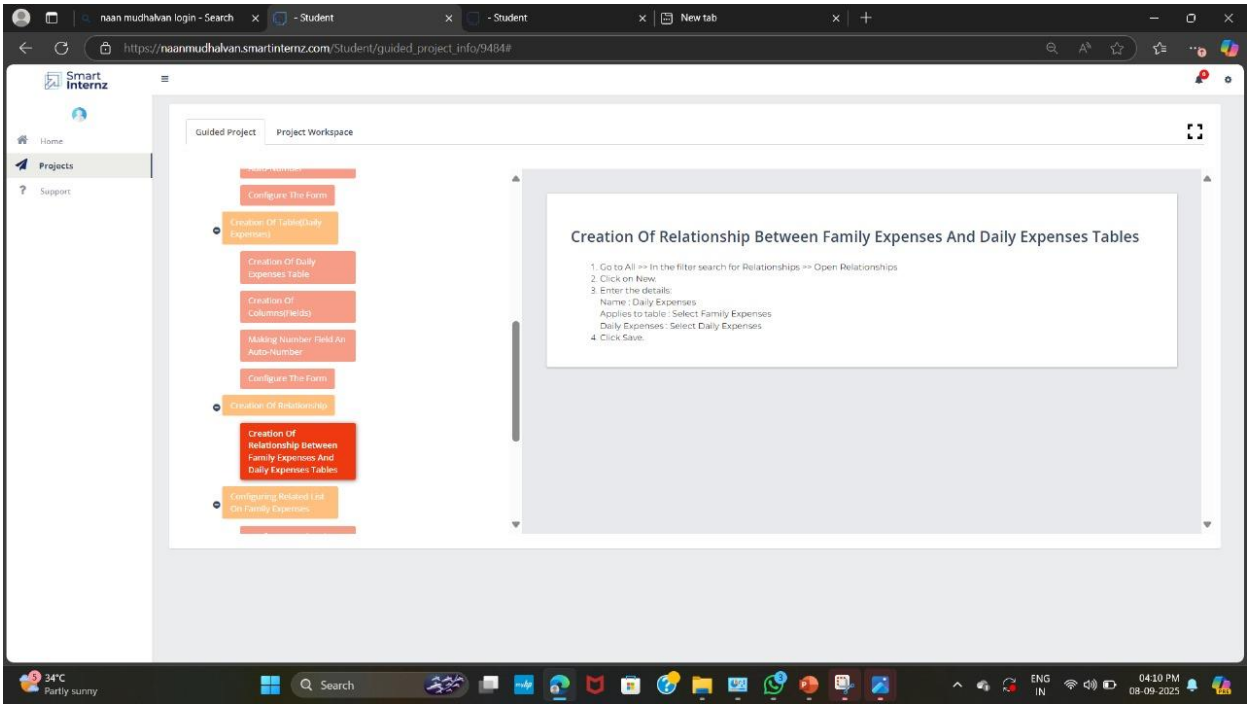
4. Establishing Relationships

To link the two tables and ensure data integrity, we created a relationship between Daily Expenses and Family Expenses. This relationship allows us to view all related daily expense records directly from a single Family Expenses record.

Step 4.1: Creating the Relationship

1. In the **Filter Navigator**, type Relationships and click on **Relationships** under **System Definition**.
2. Click **New**.
3. Configure the relationship as follows: ○ **Name:** Daily Expenses
   - ○ **Applies to table:** Family Expenses
   - ○ **Queries from table:** Daily Expenses
   - ○ **Query with:** current.addQuery('family_expense', parent.sys_id); (This ensures that only records related to the parent Family Expenses record are displayed).
4. Click **Submit** to save the relationship.

Here's an image depicting the creation of the relationship:

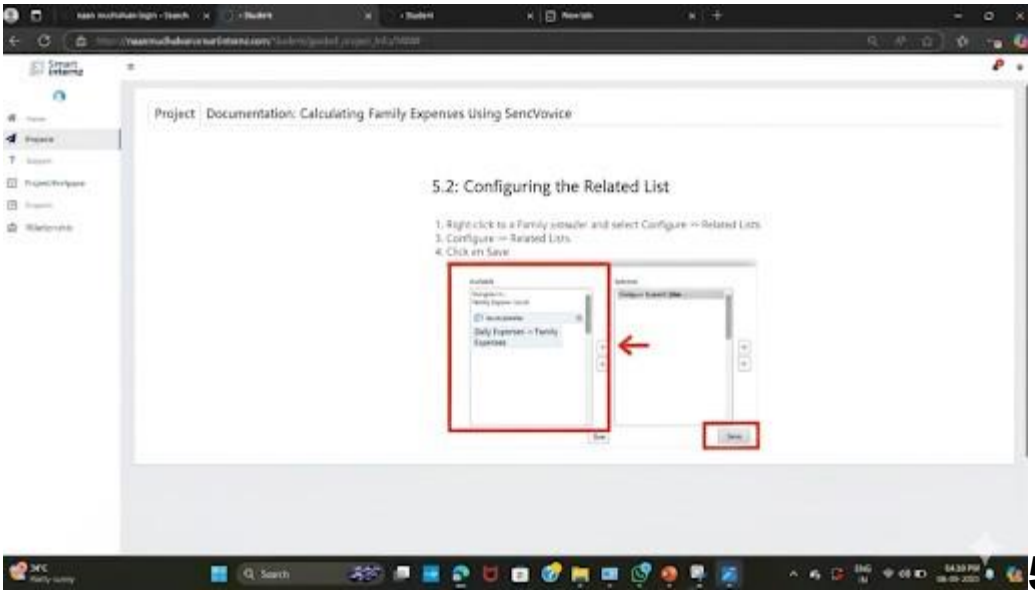


Step 4.2: Configuring the Related List

To make the relationship visible on the Family Expenses form, we configured a related list.

1. Navigate to a Family Expenses record.
2. Right-click the form header and select **Configure > Related Lists**.
3. Move **Daily Expenses -> Family Expenses** from the **Available** slushbucket to the **Selected** slushbucket.
4. Click **Save**.

This action adds a related list tab at the bottom of the Family Expenses form, allowing users to see and manage all daily expenses associated with that particular record. Here's an image showing how to configure the Related List:

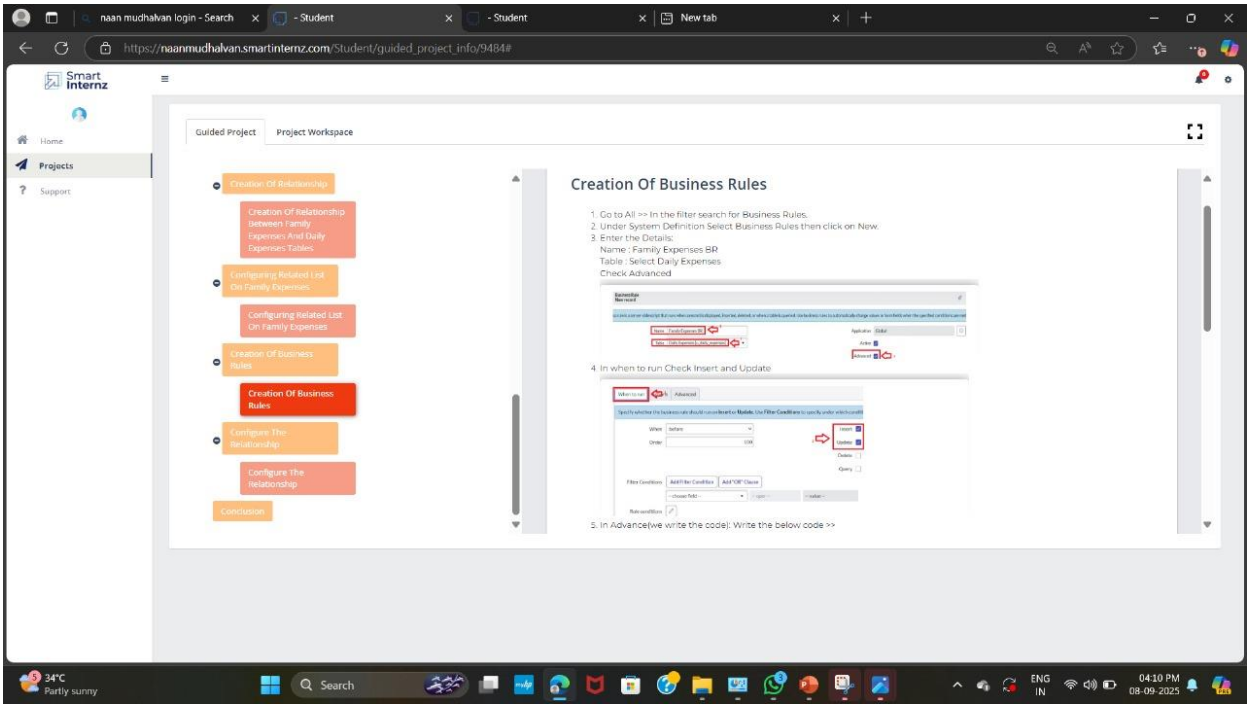


5.

Implementing Business Rules

Business rules are server-side scripts that run when a record is inserted, updated, or deleted. We used a business rule to automatically calculate and update the total expenses in the Family Expenses table whenever a new Daily Expenses record is created or updated.

5.1: Creating the Business Rule



5.2: Adding the Script

1. In the **Filter Navigator**, type Business Rules and click on **Business Rules** under **System Definition**.
2. Click **New**.
3. Enter the following details:
    - **Name:** Update Family Expenses
    - **Table:** Daily Expenses
    - **When to run:** Select **after** for the Insert and Update operations.
    - **Filter Condition:** None, as it applies to all records.
    - **Advanced:** Check this box to enable the scripting section. Here's

an image showing the creation of the business rule:

The following script was placed in the **Advanced** tab to execute the calculation logic.

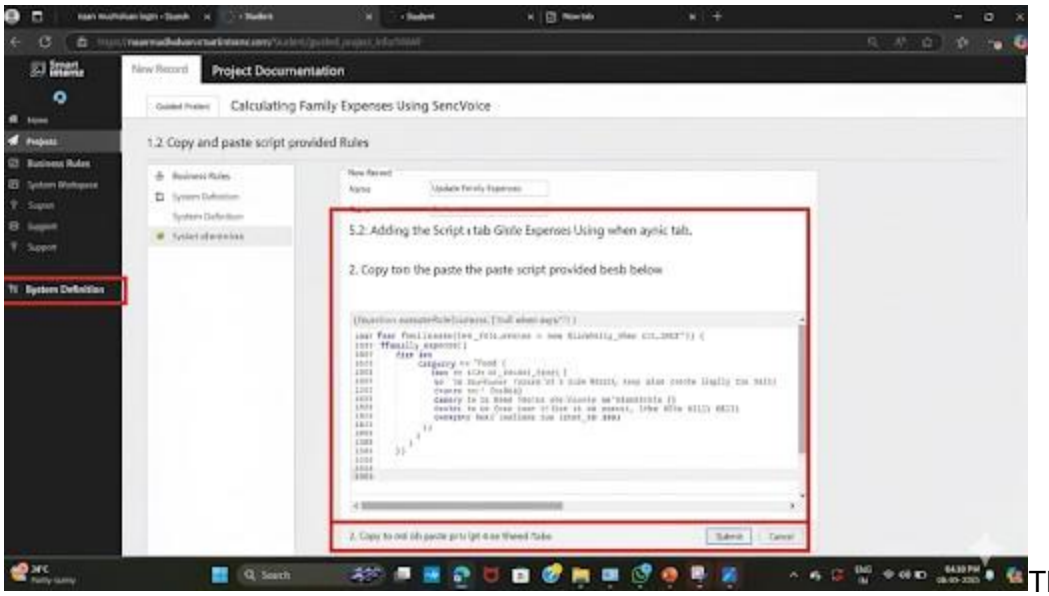
```
(function executeRule(current, previous /*null when async*/) {

    var familyExpenses = new
GlideRecord('x_<your_scope>_family_expenses');
familyExpenses.get(current.family_expense);
     if (familyExpenses.isValidRecord())
{          // Increment the total expense
amount
familyExpenses.setValue('total_amount',
familyExpenses.getValue('total_amount') + current.amount);
```

```
        // Update the category fields based on the daily expense
category         var category =
current.category.getDisplayValue();         if (category ==
'Food') {             familyExpenses.setValue('food',
familyExpenses.getValue('food') + current.amount);         }
else if (category == 'Rent') {
familyExpenses.setValue('rent', familyExpenses.getValue('rent')
+ current.amount);         } else if (category == 'Travel') {
familyExpenses.setValue('travel',
familyExpenses.getValue('travel') + current.amount);
        }
        familyExpenses.update();
    }

})(current, previous);
```

Here is an image showing the script for the business rule:



This script performs the following actions:

1. It retrieves the Family Expenses record associated with the current Daily Expenses record.
2. It adds the amount of the daily expense to the total_amount of the family expense record.
3. It then updates the specific category field (e.g., food, rent, travel) based on the category selected for the daily expense.
4. Finally, it saves the changes to the Family Expenses record, ensuring the totals are always up-to-date.

6. Conclusion

By following these steps, we successfully built a simple yet functional application within the ServiceNow platform. The application effectively automates the calculation of family expenses by leveraging custom tables, established relationships, and a business rule to perform real-time updates. This project demonstrates the power and flexibility of ServiceNow's low-code development environment for creating practical business applications.