**Part A – System Requirement Specification**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| R1 |  | The software should provide facility for user to record their expenses or incomes (**AKA, Transaction**). These transactions can be either one-off or recurrence transaction. The software system should provide facility to record both type of transaction in convenience manner. The general details related to a transaction like date and time, the amount, payee and payer should be recorded. There should a separate form in order to record a new transaction. User should be able to specify a particular transaction is either income or an expense. | | | |
|  | R1.1 | The one-off transaction is a particular distinct expense or an income of a particular day. User should be able to give a title or a name, specify the amount, select the date optionally and save the transaction. There should be a date field which is optional for the user when recording one-off transaction. If user does not specify the date field when the transaction is saved, it will be recorded against the current day. Or else, the transaction will be record against the specified date. Having input field to add a comment will be useful for the user to add some additional notes related to transaction. One-off transaction can even be created for past days as well. | | | |
|  | R1.2 | The software should further provide facility to save recurrence expenses of the user. Once user has created a recurrence expense, that particular expense will be counted to the expense of the corresponding futures days until the expense is deleted or it is expired. There can be daily, weekly, monthly or yearly expenses for the user. When creating recurrence expense, user should be able to define two date fields as ‘Effective from Date’ and ‘Effective to Date’. Both these fields can be optional for user. The current date should be populated for the ‘Effective from Date’ by default. If user has not specified ‘Effective to Date’, it should be populated by the system based on the selected ‘Frequency’ of the expense. Recurring expenses can even be created from past days.   * Daily Event - ‘Effective to Date’ is one year from the current day * Weekly Event- ‘Effective to Date’ is one year from the current day * Monthly Event- ‘Effective to Date’ is one year from the current day * Yearly Event- ‘Effective to Date’ is 10 year from the current day | | | |
|  |  |  | |  | |
|  |  | R1.2.1 | | When recording weekly recurrence expense, user should be able to specify the day of the week. It can be either as in number format from 1 to 7 or a selection from Sunday to Saturday. | |
|  |  | R1.2.2 | | When recording monthly recurrence expense, user should be able to specify the day of the month as in number format from 1 to 31st or like ‘First-Sunday’, ‘Second-Wednesday’ etc. | |
|  |  | R1.2.3 | | Similarly, as above two, when recording yearly expense, user should be able to specify the month of the year and the day of the month. | |
|  |  |  | |  | |
| R2 | The software system should provide ability to update a selected transaction. | | | | |
| R3 | The software should provide facility to view incomes of a particular week. This should clearly show the incomes of each day of the week with enough and possible additional details like comments, title and payer etc. User should be able to get a printout of the report. | | | | |
| R4 | Software should provide facility to search and list down the expenses for a given criteria like selected date or date range, expense type, payer or payee. User should be able to generate reports using these data and further, if user want to get a printout of it. The software itself should provide the print functionality. | | | | |
| R5 | The software should provide facility to delete transactions. When deleting a transaction, a confirmation to delete the transaction should be taken from the user by the system. | | | | |
| R6 | The software should provide facility to view expenses of a particular week. This should clearly show the expenses of each day of the week with enough and possible additional details like comments, title, payee and payer etc. User should be able to get a printout of this report. | | | | |
| R8 | Software should calculate the forecasted value for incomes and expenses on a future date by analyzing the history data available in the system database. User should be able to view these forecasted values for a given date. | | | | |
| R7 | Software should provide time management functionality by allowing user to record their day to day or scheduled tasks or events. These events can be appointments or tasks. The general details related to an event like date, start time, end time, name and comment should be recorded. There should a separate form in order to record a new event. There are two types of events which one-off events and recurring events. User should be able to specify a particular event is either one-off or recurrence. | | | | |
|  | R7.1 | | The one-off events happened just a once on a particular day. User should be able to add a title or a name, date on which the event happens, start time, end time and then save the event. The date field should be mandatory when saving one-off event, and it should be current or future date when creating the event. Having input field to add a comment will also be useful for the user. | | |
|  | R7.2 | | The software should further provide facility to save recurrence events for the user. When creating recurrence event, user should be able to define an expire date on which this particular recurrence event expires. By default, the ‘Start Date’ will be taken as the date on which the recurrence event is created within the system. If user has not specified ‘Expire Date’, by default, the recurrence event will be created based on the selected occurrence. User should be able to create recurrence events on daily, weekly, monthly and yearly basis.   * Daily Event - ‘Effective to Date’ is one year from the current day * Weekly Event- ‘Effective to Date’ is one year from the current day * Monthly Event- ‘Effective to Date’ is one year from the current day * Yearly Event- ‘Effective to Date’ is 10 year from the current day | | |
|  |  | | R7.2.1 | | When creating weekly recurrence event, user should be able to specify the day of the week. It can be either as in number format from 1 to 7 or as a selection from Sunday to Saturday. |
|  |  | | R7.2.2 | | When recording monthly recurrence event, user should be able to specify the day of the month as in number format from 1 to 31st or like ‘First-Sunday’, ‘Second-Wednesday’ etc. |
|  |  | | R7.2.3 | | Similarly, as above two, when recording yearly event, user should be able to specify the month of the year and the day of the month. |
| R8 | The software should provide facility to get a weekly report of events. | | | | |
| R9 | The software system should provide ability to update or delete a selected event. | | | | |
|  | The software should provide search capabilities of events, like search events by date or date range, and search events by type. | | | | |
| R8 | System should provide facility to manage contacts as payer or payee. This includes creating new contact, searching, updating and deleting contacts. Contact name, address and telephone number should be saved with a contact record. | | | | |

**Non-Functional Requirements**

1. System should work on windows.
2. System should provide offline capabilities.
3. After reinstalling the application, the history data should be available for the same login.
4. Concurrent login should be avoided.
5. After uninstalling the application, data stored should permanently be removed. There should not be data retained in the computer.
6. No one should be able to see the data from the computer file system or from the data base. All data should be encrypted.

**Screen Mockups**

Create one-off transaction

Graphical user interface, application

Description automatically generated

Create daily recurrence transaction

Graphical user interface

Description automatically generated

Create weekly recurrence transaction

Graphical user interface

Description automatically generated

Create monthly recurrence transaction

Graphical user interface

Description automatically generated

Create yearly recurrence transaction

Graphical user interface

Description automatically generated

Create one-off event

Graphical user interface

Description automatically generated

Create Daily Recurrence Event

Graphical user interface, application

Description automatically generated

Create Weekly Recurrence Event

Graphical user interface, application

Description automatically generated

Create Monthly Recurrence Event

Graphical user interface

Description automatically generated

**Part B – (1) Use Case Diagram**

Use Case Diagram for Add Operations

Diagram

Description automatically generated

Use Case Diagram for Manage Operations (View/Update/Delete)

Diagram

Description automatically generated

**Part B – (2) Use Case Diagram Description**

Use case description: Record Transaction

|  |  |  |
| --- | --- | --- |
| UC No | UC0001 | |
| UC Name | Record Transaction | |
| UC Description | Recording a new transaction either income or expense. Also, either one-off transaction or recurring transaction. Options are given to the user in order to select ‘Transaction Type’ and ‘Occurrence’. Both income and expense will be recorded as transaction either as one-off transaction or recurring transaction based on the options which user selected. | |
| UC Priority | High | |
| Primary Actor | User | |
| Pre-condition | User with valid authentication status. | |
| Success End Condition | New transaction should be recorded. | |
| Triggering Event | User click on ‘New Transaction’ menu item of the menu bar of main form. | |
| **Main Scenario** | **User** | **System** |
|  |  | 1. Open the ‘New Transaction’ form with default options.   Type = ‘Income’  Occurrence = ‘One-off’ |
|  | 1. Provide a name or a title for the transaction,   Specify a date,  Specify valid amount |  |
|  | 1. User can change transaction type to ‘Expense’ if use want. By default, this is set to ‘Income’ |  |
|  | 1. Click on ‘save’ button |  |
|  |  | 1. Validate user inputs |
|  |  | 1. Persist transaction in the database |
|  |  | 1. Acknowledge user with success status |
| **Variation** 1 |  |  |
|  |  | (1). Main Scenario Step 1 |
|  | (2) Main Scenario Step 2,3 |  |
|  | (3). Select ‘Recurrence’ as occurrence type |  |
|  |  | (4). Shows up ‘Frequency’ drop down with ‘Daily’ default value. |
|  | (5). Click on ‘save’ button with default frequency which is daily |  |
|  |  | (6). Main Scenario Step 5, 6, 7 |
| **Variation 2** |  |  |
|  |  | (1). Main Scenario Step 1 |
|  | (2). Main Scenario Step 2,3 |  |
|  | (3). Select ‘Recurrence’ as occurrence type |  |
|  |  | (4). Shows up ‘Frequency’ drop down with ‘Daily’ default value. |
|  | (5) Change the ‘Frequency’ to ‘Weekly’ |  |
|  |  | (6) Shows up a new form element to specify the day of the week. |
|  | (7). Click on ‘save’ button |  |
|  |  | Main Scenario Step 5,6,7 |
|  |  |  |
| **Variation 3** |  |  |
|  |  | 1. Main Scenario Step1 |
|  | (2). Main Scenario Step 2,3 |  |
|  | (3). Select ‘Recurrence’ as occurrence type |  |
|  |  | 4). Shows up ‘Frequency’ drop down with ‘Daily’ default value. |
|  | (5) Change the ‘Frequency’ to ‘Monthly’ |  |
|  |  | (6) Shows up a new form element to specify the day of the month. |
|  | (7) Click on ‘save’ button |  |
|  |  | (8).Main Scenario Step 5,6,7 |
|  |  |  |
| **Variation 4** |  |  |
|  |  | 1. Main Scenario Step 1 |
|  | (2). Main Scenario Step 2,3  Variation 1 Step (6) |  |
|  | (3). Select ‘Recurrence’ as occurrence type |  |
|  |  | 4). Shows up ‘Frequency’ drop down with ‘Daily’ default value. |
|  | (5) Change the ‘Frequency’ to ‘Yearly’ |  |
|  |  | (6) Shows up a new form element to specify the day of the year. |
|  | (7) Click on ‘save’ button |  |
|  |  | (8).Main Scenario Step 5,6,7 |
| Alternative Scenario 1 |  | At 3, System was unable to find the book for a given keyword. The keyword might be wrong. Go to 3 with different keyword. |
| Alternative Scenario 2 |  | At 3, System was unable to find the book. The book is not available in the current stock. Then  <<extends>> Create Customer Order |
| Exceptional Scenario 1 | A network or server failure. User should be notified with error message. | |
| Inclusions |  | |
| Extensions | Create Customer Order ???? | |

**Use case description: Record Event**

|  |  |  |
| --- | --- | --- |
| UC No | UC0002 | |
| UC Name | Record Event | |
| UC Description | Recording a new event either as one-off or recurrence. Also, either as tasks or appointments. Based on the occurrence option selected by the user, the event will be either one-off or recurrence. Both one-off and recurrent events can be created from a single form under a single use case. | |
| UC Priority | High | |
| Primary Actor | User | |
| Pre-condition | User with valid authentication status. | |
| Success End Condition | New event should be recorded. | |
| Triggering Event | User click on ‘New Event’ menu item of the menu bar of main form. | |
| **Main Scenario** | **User** | **System** |
|  |  | 1. Open the ‘Event’ form with default options.   Type = ‘Task’  Occurrence = ‘One-off’ |
|  | 1. Provide a name or a title for the event, Start Date, Start Time and End Time and also the Type as ‘task’ or ‘appointment’ |  |
|  | 1. Click on ‘save’ button |  |
|  |  | 1. Validate user inputs |
|  |  | 1. Validate overlapping event |
|  |  | 1. Persist event in the database |
|  |  | 1. Acknowledge user with success status |
| **Variation** 1 | Create daily recurrence event | |
|  |  | (1). Main Scenario Step 1 |
|  | (2). Main Scenario Step 2 |  |
|  | (3). Change the ‘Occurrence’ option to ‘Daily’ |  |
|  |  | (4). Shows up ‘Expire Date’ field with on year ahead default value. |
|  | (5). Change ‘Expire Date’ if needed |  |
|  | (6). Click on ‘save’ button |  |
|  |  | (7). Main Scenario Step 4, 5, 6,7 |
| **Variation 2** | Create weekly recurrence event | |
|  |  | (1). Main Scenario Step 1 |
|  | (2). Main Scenario Step 2 |  |
|  | (3). Change the ‘Occurrence’ option to ‘Weekly’ |  |
|  |  | (4). Shows up ‘Day of Week’ drop down with ‘Sunday’ default value.  Shows up ‘Expire Date’ field with on year ahead default value. |
|  | (5) Specify the ‘Day of Week’  Change the ‘Expire Date’ if needed. |  |
|  | (6). Click on ‘save’ button |  |
|  |  | (7) Main Scenario Step 4, 5, 6,7 |
|  |  |  |
| **Variation 3** | Create monthly recurrence event | |
|  |  | 1. Main Scenario Step1 |
|  | (2). Main Scenario Step 2 |  |
|  | (3). Change the ‘Occurrence’ option to ‘Monthly’ |  |
|  |  | (4). Shows up input fields to specify the day of the month.  Shows up ‘Expire Date’ field with on year ahead default value. |
|  | (5) Specify the ‘Day of Month’  Change the ‘Expire Date’ if needed. |  |
|  | (6). Click on ‘save’ button |  |
|  |  | (7) Main Scenario Step 4, 5, 6,7 |
|  |  |  |
| **Variation 4** | Create yearly recurrence event | |
|  |  | 1. Main Scenario Step 1 |
|  | (2). Main Scenario Step 2 |  |
|  | (3). Change the ‘Occurrence’ option to ‘Yearly’ |  |
|  |  | (4). Shows up input fields to specify the month of the year and the day of the month.  Shows up ‘Expire Date’ field with on year ahead default value. |
|  | (5) Specify the month of the year and the day of the month.  Change the ‘Expire Date’ if needed. |  |
|  | (6). Click on ‘save’ button |  |
|  |  | (7) Main Scenario Step 4, 5, 6,7 |
| Alternative Scenario 1 |  |  |
| Alternative Scenario 2 |  |  |
| Exceptional Scenario 1 | A network or server failure. User should be notified with error message. | |
| Inclusions |  | |
| Extensions | RecordRecurringEvent | |

**Use case description: Update Event**

|  |  |  |
| --- | --- | --- |
| UC No | UC0003 | |
| UC Name | Update Event | |
| UC Description | User should be able to update a selected event. Most of the steps for this use case are almost similar to UC0001 which is ‘Record Event’ use case. | |
| UC Priority | High | |
| Primary Actor | User | |
| Pre-condition | User with valid authentication status. | |
| Success End Condition | An existing event should be updated. | |
| Triggering Event | User performs a search and click on ‘Edit’ on top of a selected record. | |
| **Main Scenario** | **User** | **System** |
|  | (1). <<Include>> Search Event |  |
|  | (2). Select event and click ‘Edit’ button. |  |
|  |  | (3). Opens ‘Event’ form with pre-populated data from the selected record. |
|  | (4). Change the data as required by the user |  |
|  | (5). Click on ‘save’ button |  |
|  |  | (6) Main Scenario Step 4, 5, 6,7 |
|  |  |  |
|  |  |  |
| **Variation** 1 | Update daily recurrence event | |
|  |  | Steps are same as ‘Record Event’ |
| **Variation 2** | Update weekly recurrence event | |
|  |  | Steps are same as ‘Record Event’ |
| **Variation 3** | Update monthly recurrence event |  |
|  |  | Steps are same as ‘Record Event’ |
| **Variation 4** | Update yearly recurrence event |  |
|  |  | Steps are same as ‘Record Event’ |
|  |  |  |
| Exceptional Scenario 1 | A network or server failure. User should be notified with error message. | |
| Inclusions | Search Event | |
| Extensions |  | |

**Use case description: Delete Event**

|  |  |  |
| --- | --- | --- |
| UC No | UC0004 | |
| UC Name | Delete Event | |
| UC Description | User should be able to delete a selected event. | |
| UC Priority | Low | |
| Primary Actor | User | |
| Pre-condition | User with valid authentication status. | |
| Success End Condition | Event will be deleted from the system. | |
| Triggering Event | User performs a search and click on ‘Delete’ on top of a selected record. | |
| **Main Scenario** | **User** | **System** |
|  | (1). <<Include>> Search Event |  |
|  | (2). Select event and click ‘Delete’ button. |  |
|  |  | (3). System will ask for confirmation with ‘Ok’, ‘Cancel’ button. |
|  | (4). User selects ‘Ok’ button. |  |
|  |  | (5). Delete the record from the DB or mark the record as inactive. |
|  |  | (6). Acknowledge the user with success message. |
|  |  |  |
| **Variation 1** | **Abort event delete operation** | |
|  | (1). Main scenario step 1,2 |  |
|  |  | (2). Main scenario step 3 |
|  | (3). User selects ‘Cancel’ button |  |
|  |  | (4). Close the confirmation window. |
| Alternative Scenario 1 |  |  |
| Alternative Scenario 2 |  |  |
| Exceptional Scenario 1 | A network or server failure. User should be notified with error message. | |
| Inclusions | Search Event | |
| Extensions |  | |

**Use case description: Search Event**

|  |  |  |
| --- | --- | --- |
| UC No | UC0005 | |
| UC Name | Search Event | |
| UC Description | Search events by specifying different search criteria like date or date range, event type, event occurrent etc. | |
| UC Priority | High | |
| Primary Actor | User | |
| Pre-condition | User with valid authentication status. | |
| Success End Condition | List of matching events for the selected search criteria. If no matching events, empty result. | |
| Triggering Event | User click on ‘Manage Event’ menu item of the menu bar of main form. | |
| **Main Scenario** | **User** | **System** |
|  | (1). User click on ‘Manage Event’ menu item from the menu bar. |  |
|  |  | (2). Open ‘Manage Event’ screen with search form and a grid with data against the default search criteria. |
|  | (3). Specify the search criteria |  |
|  | (4). Click on ‘Search’ button |  |
|  |  | (5). List up all matching events for the defined criteria. |
| Exceptional Scenario 1 | A network or server failure. User should be notified with error message. | |
| Inclusions |  | |
| Extensions | Update Event, Delete Event | |

**Use case description: Add Contact**

|  |  |  |
| --- | --- | --- |
| UC No | UC0006 | |
| UC Name | Add Contact | |
| UC Description | Allows to create a contact with details like name, telephone number, address etc. | |
| UC Priority | Medium | |
| Primary Actor | User | |
| Pre-condition | User with valid authentication status. | |
| Success End Condition | New contact should be recorded. | |
| Triggering Event | User click on ‘New Contact’ menu item of the menu bar of main form. | |
| **Main Scenario** | **User** | **System** |
|  |  | 1. Open the ‘New Contact’ form with default options.   Type = ‘Payer’ |
|  | 1. File the form |  |
|  | 1. Click on ‘Save’ button |  |
|  |  | 1. Validate user inputs |
|  |  | 1. Persist contact in the database |
|  |  | 1. Acknowledge user with success status |

**Use case description: Update Contact**

|  |  |  |
| --- | --- | --- |
| UC No | UC0007 | |
| UC Name | Update Contact | |
| UC Description | User should be able to update contact. | |
| UC Priority | Low | |
| Primary Actor | User | |
| Pre-condition | User with valid authentication status. | |
| Success End Condition | Existing contact should be updated. | |
| Triggering Event | User click on ‘Edit’ button after selecting a contact from the grid | |
| **Main Scenario** | **User** | **System** |
|  | 1. Include <<Search Contact>> |  |
|  | 1. Select contact and click on ‘Edit’ button |  |
|  |  | 3. Open contact form with populated data from selected contact. |
|  | 4. Change the data and click on ‘save’ button |  |
|  |  | 1. Validate user inputs |
|  |  | 1. Persist contact data in the database |
|  |  | 1. Acknowledge user with success status |
|  |  |  |
| Alternative Scenario 2 |  | At 3, System was unable to find the book. The book is not available in the current stock. Then  <<extends>> Create Customer Order |
| Exceptional Scenario 1 | A network or server failure. User should be notified with error message. | |
| Inclusions | Search Contact | |

**Use case description: Search Contact**

|  |  |  |
| --- | --- | --- |
| UC No | UC0008 | |
| UC Name | Record Transaction | |
| UC Description | Use should be able to search contact for a given criteria. | |
| UC Priority | Low | |
| Primary Actor | User | |
| Pre-condition | User with valid authentication status. | |
| Success End Condition | List of contacts matched to given search criteria. | |
| Triggering Event | User click on ‘Manage Contact’ menu item of the menu bar of main form. | |
| **Main Scenario** | **User** | **System** |
|  | (1). Specify search criteria |  |
|  | (2). Click on ‘search’ button |  |
|  |  | (3). Displays list of contacts matching to specified search criteria. |
| Exceptional Scenario 1 | A network or server failure. User should be notified with error message. | |
| Extensions | Delete Contact, Update Contact | |

**Part C: (1) Classes or CRC tables**

*CRC Table for Event*

|  |  |  |  |
| --- | --- | --- | --- |
| **Class Name** | **Type** | **Responsibility** | **Collaboration** |
| Event | Model | Entity class to map EVENT database table | EventInstance |
| EventInstance | Model | Representation of specific, unique and static event on a particular day. One to one matching with Event for one-off event and many-to-one mapping with Event for recurrence events. | Event |
| AddEditEventForm | View | View class which contains UI elements to add or update event  +save() | EventController |
| SearchEventForm | View | View class which contains a search form and a data grid  +search()  +delete()  +initUpdate()  +export()  +print() | EventController |
| ViewWeeklyEventForm | View | View class which displays weekly events  +print()  +export()  +previousWeek()  +nextWeek() | EventController |
| EventController | Controller | Controller class which handles requests coming from the view and also responses coming from the model.  +add()  +update()  +delete()  +searchEventByCriteria()  +getEventById()  +fetchWeeklyEvent() | AddEditEventForm  SearchEventForm  ViewWeeklyEventForm  EventService |
| EventService | Model | Class which contains all business logic related to events  +add()  +update()  +delete()  +searchEventByCriteria()  +getEventById()  +fetchWeeklyEvent() | Event  EventInstance  EventDAO |
| EventDAO | Model | Class which access the data base  +add()  +update()  +delete()  +searchEventByCriteria()  +getEventById()  +fetchWeeklyEvent() | DBAccessManager |
| DBAccessManager | Model | A class which executes all database CRUD operations for the entire application  +executeQuery()  +executeUpdate() |  |
|  |  |  |  |

*CRC Table for Transaction*

|  |  |  |  |
| --- | --- | --- | --- |
| **Class Name** | **Type** | **Responsibility** | **Collaboration** |
| Transaction | Model | Entity class to map TRANSACTION database table | TransactionInstance |
| TransactionInstance | Model | Representation of specific, unique and static transaction on a particular day. One to one matching with ‘Transaction’ for one-off transaction and many-to-one mapping with ‘Transaction’ for recurrence transactions. | Transaction |
| AddEditTransactionForm | View | View class which contains UI elements to add or update transaction  +save() | TransactionController |
| SearchTransactionForm | View | View class which contains a search form and a data grid  +search()  +delete()  +initUpdate()  +export()  +print()  +viewForecast() | TransactionController |
| ViewWeeklyTransactionForm | View | C# form which displays weekly transactions  +print()  +export()  +previousWeek()  +nextWeek() | TransactionController |
| TransactionController | Controller | Controller class which handles requests coming from the view and also responses coming from the model.  +add()  +update()  +delete()  +searchTransactionByCriteria()  +getTransactionById()  +fetchWeeklyTransaction()  +getForecastedData() | AddEditTransactionForm  SearchTransactionForm  ViewWeeklyTransactionForm  TransactionService |
| TransactionService | Model | Class which contains all business logic related to transactions  +add()  +update()  +delete()  +searchTransactionByCriteria()  +getTransactionById()  +fetchWeeklyTransaction()  +getForecastedData() | Transaction  TransactionInstance  TransactionDAO |
| TransactionDAO | Model | Database access operations  +add()  +update()  +delete()  +searchTransactionByCriteria()  +getTransactionById()  +fetchWeeklyTransaction()  +getForecastedData() | DBAccessManager |
| DBAccessManager | Model | Execute CRUD operations specific to the DBMS  +executeQuery()  +executeUpdate() |  |
| ForecastedValueJob | Model | Calculating and updating forecasted values for future dates. This is a back-end Job which runs end of the day.  +calculateForecastedValue() |  |

*CRC Table for Contact*

|  |  |  |  |
| --- | --- | --- | --- |
| **Class Name** | **Type** | **Responsibility** | **Collaboration** |
| Contact | Model | Entity class to map CONTACT database table |  |
| AddEditContactForm | View | View class which contains UI elements to add or update contact  +save() | ContactController |
| SearchContactForm | View | View class which contains a search form and a data grid  +search()  +delete()  +initUpdate()  +export()  +print() | ContactController |
| ContactController | Controller | Controller class which handles requests coming from the view and also responses coming from the model.  +add()  +update()  +delete()  +searchContactByCriteria()  +getContactById() | AddEditContactForm  SearchContactForm  ViewWeeklyContactForm  ContactService |
| ContactService | Model | Class which contains all business logic related to transactions  +add()  +update()  +delete()  +searchContactByCriteria()  +getContactById() | Contact  ContactDAO |
| ContactDAO | Model | Database access operations  +add()  +update()  +delete()  +searchContactByCriteria()  +getContactById() | DBAccessManager |
| DBAccessManager | Model | Execute CRUD operations specific to the DBMS  +executeQuery()  +executeUpdate() |  |
|  |  |  |  |

**Part C: (2) Domain Model**

Diagram

Description automatically generated

When considering the nature of a transaction, whether it is income or expense, it can be considered as an attribute of a transaction. Because, this particular attribute itself does not make a significant impact for the design of the system which is for class diagram or database table or the implementation of the functionality.

The occurrence type is a key attribute which significantly impact for the design of the system in many of the aspects, like classes, database tables and the implementation.

Also, when it comes to the event, similarly, task or appointment can be considered as an attribute of the event entity. But again, the event occurrence is a key attribute when designing the application.

Please see the “Design Decision” section in order to understand ‘EventInstance’ and ‘TransactionInstance’.

**Part D: Collaboration**

Sequence Diagram – Record Event

Diagram

Description automatically generated

Sequence Diagram – Record Transaction

Diagram

Description automatically generated

Sequence Diagram – Search Event

Diagram

Description automatically generated

Sequence Diagram – Search Transaction

Diagram

Description automatically generated

Sequence Diagram – Delete Event

Diagram

Description automatically generated

Sequence Diagram – Delete Transaction

Diagram

Description automatically generated

Sequence Diagram – Update Event

Diagram

Description automatically generated

Use case diagram – Add Contact

Diagram

Description automatically generated

**Part E – Activity** One of the following will be used for the implementation

*Option 01- MLP- ANN*

Multi-layer Perceptron is a class of Artificial Neural Network (MLP-ANN) which is used to forecast the values for future dates by using historical time series set of data. The system will integrate with an MLP-ANN sub module which calculates forecasting transaction values for incomes and expenses for future. The MLP-ANN model will run daily basis and calculate values for next 365 days. The model is developed using R which get the total transaction data from TRANSACTION\_TOTAL table and calculated values are saved in FORECASTING\_TOTAL table.

Diagram

Description automatically generated

*Option 02- Moving Average*

Moving average is one of the mechanisms which is used to calculate future values based on historical time series set of data. This technique simply takes the average of the data set within a defined range from previous data set.

In this implementation, average of last 7 days is taken as the forecasted value for the current date. These values are pre-calculated using a back end daily Job for next 365 days. Since, these are pre-calculated values, data fetch operations will be faster than calculating this value on the fly. The following orange color values are forecasted values. Only last 3 days average has been considered for the following example scenario.

|  |  |  |
| --- | --- | --- |
| **DATE** | **TOTAL INCOME** | **NEXT\_DAY\_FORECASTING** |
| 2021-03-01 | 2300 |  |
| 2021-03-02 | 3000 |  |
| 2021-03-03 | 5000 |  |
| 2021-03-04 | 2650 | 3433.333333 |
| 2021-03-05 | 4500 | 3550 |
| 2021-03-06 | 4250 | 4050 |
| 2021-03-07 | 3600 | 3800 |
| 2021-03-08 | 2750 | 4116.666667 |
| 2021-03-09 | 3125 | 3533.333333 |
| 2021-03-10 | 3900 | 3158.333333 |
| 2021-03-11 | 5100 | 3258.333333 |
| 2021-03-12 |  | 4041.666667 |
| 2021-03-13 |  | 4500 |
| 2021-03-14 |  | 5100 |

**Diagram

Description automatically generated**

**Part F – Report**

(1) Design Decisions

(i) Database table structure is created in the same way for both one-off transaction and recurring transaction. For both type of transactions, there is a header record which is saved in the TRANSACTION table and the details are saved in TRANSACTION\_INSTANCE table. One record in the TRANSACTION\_INSTANCE represents a unique transaction for unique day. For one-off transaction, there is one to one relationship from TRANSACTION to TRANSACTION\_INSTANCE table. For recurring transactions, there is one to many relationships from TRANSACTION to TRANSACTION\_INSTANCE table. Two model classes are created to map these two tables.

The following sample table structure will explain this further.

Table, Excel

Description automatically generated

As you can see in above table, transaction with id 1 is one of transaction. It has one mapping record in TRANSACTION\_INSTANCE table. Transaction with id 2 is recurring transaction. It has related 5 records in TRANSACTION\_INSTANCE.

Similar database design is applied for events as well. It also has a table as EVENT and EVENT\_INSTANCE.

(ii) TRANSACTION \_INSTANCE and EVENT\_INSTANCE tables are populated with in a separate thread which is initiated on the event of transaction create, update or delete. This will improve the performance of data fetch operations and also report view and generation, because this minimizes on the fly data process and complex queries.

(iii) Updating daily total transactions are also calculated within another thread which is initiated on the event of transaction create, update or delete. Both the total daily income and expenses are saved in a table called TRANSACTION\_TOTAL as follows. This will improve the performance when displaying the total transaction on the screen, since the total values are pre-calculated. System needs only to fetch total values from the database without doing complex calculations and performing complex queries with the data fetch.

Table

Description automatically generated

(iv) Tasks or Appointments are considered as attribute of Events, since this attribute itself does not make significant difference.

(v) As explain above, data model for recurring and one-off has been designed in similar way. This reduces the complexity of the design significantly and improve the performance.

(vi) Calculating total transaction values will be done in asynchronous way within separate threads. Those will not be calculated within the request thread of updating transaction. This will improve the performance when viewing summary or reports etc. In order to achieve this, an event should be triggered from the back end of the application in order to update statistical summary data like total expenses or income for a day. We can use event driven messaging mechanism or simply threaded programs.

(vii) ‘Income’ or ‘Expense’ was considered as simple attribute of a particular transaction. This does not make significant difference for transaction. If we consider attributes which we want to save along with recording an income or an expense, most of the attributes are common. We can consider this ‘Income’ and ‘Expense’ as a simple attribute of a transaction. When it comes to the occurrence, either ‘one-off’ or ‘recurrence’, it significantly deviates the nature of the transaction, UI design, the data that should be stored in the database.

(2) Why this design is more suitable?

1. Most of the calculations are done in an asynchronous manner which will improve the response time of create/update operations.
2. Data view and report operations will be fast because the required calculations are not being done on the fly with the user request. Those are pre-calculated from separate threads.
3. Keeping consistence database table structure for both one-off and recurring transactions and events, will reduce the complexity of the development.
4. Forecasting values are pre-calculated. This will improve the performance of the forecasting view operations.