Requirements Engineering

Dance Studio Booking System

Submitted By: Anna Kovalenko (T00243622)

Computing with Games Development

Date Submitted: 19/04/2024

**Table of Contents**

[1. Introduction/overview 4](#_Toc164430423)

[2. Functional Components 5](#_Toc164430424)

[3. User Requirements 6](#_Toc164430425)

[3.1. Dance Studio Booking System will manage Members. 6](#_Toc164430426)

[3.2. Dance Studio Booking System will manage Classes. 6](#_Toc164430427)

[3.3. Dance Studio Booking System will process Bookings. 6](#_Toc164430428)

[3.4. Dance Studio Booking System will perform administrative reporting. 6](#_Toc164430429)

[3.5. Dance Studio Booking System will process Authentication. 6](#_Toc164430430)

[4. System Requirements 7](#_Toc164430431)

[4.1. System Level Use Case Diagram 7](#_Toc164430432)

[4.2. Manage Members 8](#_Toc164430433)

[4.2.1. Add Member 8](#_Toc164430434)

[4.2.2. Update Member 11](#_Toc164430435)

[4.2.3. Delete Member 14](#_Toc164430436)

[4.3. Manage Classes 15](#_Toc164430437)

[4.3.1. Schedule Class 15](#_Toc164430438)

[4.3.2. Modify Class 18](#_Toc164430439)

[4.3.3. Cancel Class 21](#_Toc164430440)

[4.4. Process Booking 23](#_Toc164430441)

[4.4.1. Make Booking 23](#_Toc164430442)

[4.4.2. Cancel Booking 28](#_Toc164430443)

[4.5. Perform Admin 30](#_Toc164430444)

[4.5.1. Classes Overview 30](#_Toc164430445)

[4.5.2. Generate Yearly Revenue Analysis 32](#_Toc164430446)

[4.5.3. Generate Popular Dance Style Analysis 34](#_Toc164430447)

[4.6. Authentication 36](#_Toc164430448)

[4.6.1. Login Member 36](#_Toc164430449)

[4.6.2. Login Admin 38](#_Toc164430450)

[4.6.3. Recover Member Password 40](#_Toc164430451)

[5. System Model 42](#_Toc164430452)

[5.1. Level-0 DFD 42](#_Toc164430453)

[5.2. Level-1 DFD 43](#_Toc164430454)

[5.3. Level-2 DFD (Process P1: Manage Members) 44](#_Toc164430455)

[5.4. Level-2 DFD (Process P2: Manage Classes) 45](#_Toc164430456)

[5.5. Level-2 DFD (Process P3: Process Bookings) 46](#_Toc164430457)

[5.6 Level-2 DFD (Process P4: Perform Admin) 47](#_Toc164430458)

[5.7 Level-2 DFD (Process P5: Authentication) 48](#_Toc164430459)

[6. Data Model (Class Diagram) 49](#_Toc164430460)

[6.1. Class Diagram 49](#_Toc164430461)

[6.2. Relational Schema 50](#_Toc164430462)

[6.3. Database Schema 50](#_Toc164430463)

[7. Conclusion 52](#_Toc164430464)

# Introduction/overview

The Dance Studio Booking System is introduced, focusing on its main parts and what users needed.

The main goals for the project are:

1. Implement functionalities to effectively manage member profiles, enabling easy addition, updating, and deletion of member information within the system.
2. Facilitate the scheduling, modification, and cancellation of dance classes.
3. Enable members to easy make and cancel class bookings.
4. Enable admin to easy view classes, generate yearly revenue analyses, and popular dance style analyses.
5. Implement an authentication process for both members and administrators, features such as member and admin login functionalities, along with password recovery options for members.

It includes key sections: Members, Classes, Bookings, Admin, and Authentication, each crucial in managing various aspects of studio operations.

The system requirements outlined using detailed use case diagrams, defined crucial functionalities for managing members, classes, bookings, administrative tasks, and authentication.

Through Data Flow Diagrams (DFDs), the System Model visually explains how the system would work. Focus on Member interactions outlined processes involving member profile managing, booking classes, and administrative functions.

The Data Model, the Class Diagram, illustrated connections between important classes like Bookings, Classes, Members, and Instructors, ensuring an efficient experience for all involved.

# Functional Components

# User Requirements

## Dance Studio Booking System will manage Members.

* + 1. Dance Studio Booking System will add a member profile.
    2. Dance Studio Booking System will update a member profile.
    3. Dance Studio Booking System will delete a member profile.

## Dance Studio Booking System will manage Classes.

* + 1. Dance Studio Booking System will schedule a class.
    2. Dance Studio Booking System will modify a class.
    3. Dance Studio Booking System will cancel a class.

## Dance Studio Booking System will process Bookings.

* + 1. Dance Studio Booking System will make a booking.
    2. Dance Studio Booking System will cancel a booking.

## Dance Studio Booking System will perform administrative reporting.

* + 1. Dance Studio Booking System will provide all created classes information.
    2. Dance Studio Booking System will produce a yearly revenue analysis.
    3. Dance Studio Booking System will produce popular dance styles analysis.

## Dance Studio Booking System will process Authentication.

* + 1. Dance Studio Booking System will allow member to login.
    2. Dance Studio Booking System will allow admin to login.
    3. Dance Studio Booking System will provide a password recovery feature for member.

# System Requirements

Members Module is responsible for adding, editing, and deleting member profiles. Store member information such as first name, second name, contact details and booked classes.

Classes Module is responsible for creating, modifying, and cancelling dance classes. Assign dance type, date, time, instructors, set classes capacities and prices.

Bookings Module is responsible for allowing members to book classes. Handle class reservations and cancellations.

Admin Module is responsible for displaying all created classes and their information, generating reports on yearly revenue based on bookings, as well as analysing classes attendance to identify popular dance styles.

Authentication Module is responsible for allowing members and administrators to login and providing recovery password feature for members.

## System Level Use Case Diagram

The following system level use case diagram illustrates the high-level system requirements.

Admin

Member

## Manage Members

This module provides functions to add, update, and delete member profiles, streamlining member data management within the system.

### Add Member

This function adds details of a member in the system. Each member is identified by a unique MemberID.

Member

<<includes>>

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Add Member | |
| **Use Case Id** | 1 | |
| **Priority** | High | |
| **Source** | Member | |
| **Primary Business Actor** | Member | |
| **Other Participating Actors** |  | |
| **Description** | The function creates member profile and adds data to Members file. | |
| **Preconditions** |  | |
| **Trigger** |  | |
| **Expected Scenario** | **Member** | **System** |
|  | **Step 1:** Start Add Profile function.  **Step 3:** Enter the required Data:   * Firstname (string) * Lastname (string) * Gender (string) * DOB (string) * Email (string) * Phone (int) * Password (string)   **Step 4:** Confirm Create. | **Step 2:** Display UI.  **Step 5:** Validate the data:   * All fields must be entered. * First name cannot be null or empty. * First name cannot contain numbers or symbols. * Second name cannot be null or empty. * Second name cannot contain numbers or symbols. * Email cannot be null or empty. * Email must be in valid format([xxxx@xxx.xxx](mailto:xxxx@xxx.xxx)) * Email cannot contain word ‘admin’. * Email cannot exist in the database. * Phone cannot be null or empty. * Phone number must contain >= 9 numbers. * Phone cannot contain letters or symbols. * Age must be over 18 and above 65. * Password cannot be null or empty. * Password must be >= 8 symbols.   **Step 6:** Assign MemberID.  **Step 7:** Save data in the Members File:   * MemberID (int) * Firstname (string) * Lastname (string) * Gender (string) * DOB (string) * Email (string) * Phone (int) * Password (string)   **Step 8:** Display Confirmation Message.  **Step 10:** Display Login UI. |
| **Alternate Scenarios** | **Member** | **System** |
| **Invalid Data Entered** |  | **Step 5:** Invalid data detected.  **Step 6:** Display an appropriate message.  **Step 7:** Return to step 3. |
| **Conclusions** | The member profile was created. | |
| **Post conditions** | The member can book classes. | |
| **Business Rules** | Member must be over 18 and above 65 years of age. | |
| **Implementation Constraints** |  | |

### Update Member

This function updates details of a member in the system.

Member

<<includes>>

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Update Member | |
| **Use Case Id** | 2 | |
| **Priority** | Medium | |
| **Source** | Member | |
| **Primary Business Actor** | Member | |
| **Other Participating Actors** |  | |
| **Description** | The function update member’s details and makes updates in the Members file. | |
| **Preconditions** | The member must be login. | |
| **Trigger** |  | |
| **Expected Scenario** | **Member** | **System** |
|  | **Step 1:** Start Update Member Profile function.  **Step 4:** Update required data:   * Firstname (string) * Lastname (string) * Gender (string) * DOB (string) * Phone (int) * Email (string) * OldPassword (string) * NewPassword (string)   **Step 5:** Confirm Save. | **Step 2:** Display UI.  **Step 3:** Retrieve data from Members file:   * Firstname (string) * Lastname (string) * Gender (string) * DOB (string) * Email (string) * Phone (int)   **Step 6:** Validate the data:   * Data must be entered: * Firstname (string) * Lastname (string) * Gender (string) * DOB (string) * Phone (int) * Email (string) * Entering data for both Password and New Password are optional. * First name cannot be null or empty. * First name cannot contain numbers or symbols. * Second name cannot be null or empty. * Second name cannot contain numbers or symbols. * Email cannot be null or empty. * Email must be in valid format([xxxx@xxx.xxx](mailto:xxxx@xxx.xxx)) * Email cannot contain word ‘admin’. * Email cannot exist in the database. * Phone cannot be null or empty. * Phone number must contain >= 9 numbers. * Phone cannot contain letters or symbols. * Age must be over 18 and above 65. * Old Password = Password of current member in Members file. * New Password must have >= 8 digits.   **Step 7:** Update data in the Members File:   * Firstname (string) * Lastname (string) * Gender (string) * DOB (string) * Phone (int) * Email (string) * Password (string)   **Step 8:** Display Confirmation Message.  **Step 9:** Display Member Profile UI. |
| **Alternate Scenarios** | **Member** | **System** |
| **Invalid Data** |  | **Step 6:** Invalid data detected.  **Step 7:** Display an appropriate message.  **Step 8:** Return to step 4. |
| **Conclusions** | The data of the member was updated. | |
| **Post conditions** | The member has new information in profile and in the Members file. | |
| **Business Rules** | Member must be over 18 and above 65 years of age. | |
| **Implementation Constraints** |  | |

### Delete Member

This function allows deletes details of a member in the system.

Member

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Delete Member | |
| **Use Case Id** | 3 | |
| **Priority** | Low | |
| **Source** | Member | |
| **Primary Business Actor** | Member | |
| **Other Participating Actors** |  | |
| **Description** | The function deletes members data in the Members file. The member cannot login again with the same details. | |
| **Preconditions** | The member must be login. The member must be in Update Member section. | |
| **Trigger** |  | |
| **Expected Scenario** | **Member** | **System** |
|  | **Step 1:** Start Delete Member function.  **Step 3:** Confirm OK. | **Step 2:** Display appropriate messages.  **Step 4:** Delete member’s data from the Members file.  **Step 5:** Display message.  **Step 6:** Display Log In UI. |
| **Alternate Scenarios** | **Member** | **System** |
|  |  |  |
| **Conclusions** | The member profile was deleted. | |
| **Post conditions** | The member cannot login. | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

## Manage Classes

This module provides functions to schedule, modify, and cancel classes.

### Schedule Class

This function schedule class in the system. Each class is identified by a unique ClassID.

Admin

<<includes>>

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Schedule Class | |
| **Use Case Id** | 4 | |
| **Priority** | High | |
| **Source** | Admin | |
| **Primary Business Actor** | Admin | |
| **Other Participating Actors** |  | |
| **Description** | The function schedules a class and adds data to the Classes file. | |
| **Preconditions** | The admin must be login. | |
| **Trigger** |  | |
| **Expected Scenario** | **Admin** | **System** |
|  | **Step 1:** Start Schedule Class function.  **Step 5:** Enter the required Data:   * Name (string) * Type (string) * Date (string) * Time (string) * Instructor (string) * Capacity (int) * Price (float)   **Step 6:** Confirm Add. | **Step 2:** Display UI.  **Step 3:** Retrieve data from the Instructors file:   * Instructor (string)   **Step 4:** Retrieve data from the Class\_Types file:   * Type (string)   **Step 7:** Validate the data:   * All fields must be entered. * Name cannot be null or empty. * Type must be selected. * Time must be in right format. * Time cannot be null or empty. * Time must be available. * Time cannot be in the past. * Date must be available. * Date cannot be in the past. * Instructor must be selected. * Capacity cannot be null or empty. * Capacity <=30. * Price cannot be null or empty. * Price in valid format (00.00).   **Step 8:** Assign ClassID.  **Step 9:** Assign AvailablePlaces.  **Step 10:** Save data in the Classes File:   * ClassID (int) * Name (string) * TypeID (string) * Date (string) * Time (string) * InstructorID (int) * Capacity (int) * AvailablePlaces(int) * Price (float)   **Step 11:** Increase Quantity\_of\_classes by 1 in the Class\_types file.  **Step 12:** Display Confirmation Message.  **Step 13:** Display Classes Overview UI. |
| **Alternate Scenarios** | **Admin** | **System** |
| **Invalid Data Entered** |  | **Step 7:** Invalid input data.  **Step 8:** Display an appropriate error message.  **Step 9:** Return to step 5. |
| **Conclusions** | The class was created. | |
| **Post conditions** | The member can book a class. | |
| **Business Rules** | Class must be scheduled for an available date and time.  Capacity is maximum 30 people. | |
| **Implementation Constraints** |  | |

### Modify Class

This function modifies details of a class in the system.

Admin

<<includes>>

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Modify Class | |
| **Use Case Id** | 5 | |
| **Priority** | Medium | |
| **Source** | Admin | |
| **Primary Business Actor** | Admin | |
| **Other Participating Actors** |  | |
| **Description** | The function updates details of a class in the Classes file. | |
| **Preconditions** | The admin must be login. | |
| **Trigger** |  | |
| **Expected Scenario** | **Admin** | **System** |
|  | **Step 1:** Start Modify Class function.  **Step 4:** Select type.  **Step 6:** Select class.  **Step 7:** Confirm Select.  **Step 12:** Update the required data:   * Name (string) * Type (string) * Date (string) * Time (string) * Instructor (string) * Capacity (int) * Price (float) | **Step 2:** Display UI (Select Class part).  **Step 3:** Retrieve Types from the Class\_Types file.  **Step 5:** Retrieve data from Classes file:   * Class\_ID(int) * Name (string) * TypeID (string) * Date (string) * Time (string) * AvaliablePlaces(int) * InstructorID (int) * Capacity (int) * Price (float)   **Step 8:** Display UI (Update Current Class Details part).  **Step 9:** Retrieve data from Instructors file:   * Instructor (String)   **Step 10:** Retrieve data from the Class\_Types file:   * Type (string)   **Step 11:** Retrieve all details of the selected class from the Classes file and load in UI:   * Name (string) * TypeID (string) * Date (string) * Time (string) * InstructorID (int) * Capacity (int) * Price (float)   **Step 13:** Validate the data:   * All fields must be entered. * Name cannot be null or empty. * Type cannot be null or empty. * Type must be selected. * Time must be in right format. * Time cannot be null or empty. * Time must be available. * Time cannot be in the past. * Date must be available. * Date cannot be in the past. * Instructor must be selected. * Capacity cannot be null or empty. * Capacity <=30. * Price cannot be null or empty. * Price in valid format (00.00).   **Step 14:** Update data in the Classes file:   * Name (string) * TypeID (string) * Date (string) * Time (string) * InstructorID (int) * Capacity (int) * Price (float)   **Step 15:** Display Confirmation Message.  **Step 16:** Reset UI.  **Step 17:** Hide Update Current Class Details part. |
| **Alternate Scenarios** | **Admin** | **System** |
| **Invalid Data Entered** |  | **Step 13:** Invalid data.    **Step 14:** Display an appropriate message.    **Step 15:** Return to step 12. |
| **Conclusions** | The Class details was updated. | |
| **Post conditions** | The admin can use updated class. | |
| **Business Rules** | The Class must be scheduled for an available date and time.  Capacity is maximum 30 people. | |
| **Implementation Constraints** |  | |

### Cancel Class

This function deletes details of a class in the system.

Admin

<<includes>>

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Cancel Class | |
| **Use Case Id** | 6 | |
| **Priority** | Low | |
| **Source** | Admin | |
| **Primary Business Actor** | Admin | |
| **Other Participating Actors** |  | |
| **Description** | The function deletes details of a class in the Classes file. | |
| **Preconditions** | The admin must be login. | |
| **Trigger** |  | |
| **Expected Scenario** | **Admin** | **System** |
|  | **Step 1:** Start Cancel Class function.  **Step 4:** Select type.  **Step 6:** Select class.  **Step 7:** Confirm Cancel. | **Step 2:** Display UI.  **Step 3:** Retrieve Type from the Class\_Types file.  **Step 5:** Retrieve data from Classes file:   * Class\_ID (int) * Name (string) * TypeID (string) * Date (string) * Time (string) * AvaliablePlaces(int0 * InstructorID (int) * Capacity (int) * Price (float)   **Step 8:** Validation:   * The class cannot be cancelled if someone has already booked it.   **Step 9:** Delete all required data from the Classes file:   * ClassID (int) * AvailablePlaces (int) * Name (string) * TypeID (String) * Date (string) * Time (string) * InstructorID (int) * Capacity (int) * Price (float)   **Step 10:** Update qty\_of\_classes by -1.  **Step 11:** Display message.  **Step 12:** Reset UI. |
| **Alternate Scenarios** | **Admin** | **System** |
| **Booked class** |  | **Step 8:** The class was booked.  **Step 9:** Display an appropriate message.  **Step 10:** Return to step 6. |
| **Conclusions** | The Class was cancelled. | |
| **Post conditions** | The class does not exist and cannot be booked. | |
| **Business Rules** | The class can only be cancelled if nobody has booked it. | |
| **Implementation Constraints** |  | |

## Process Booking

This module provides functions to make and cancel bookings.

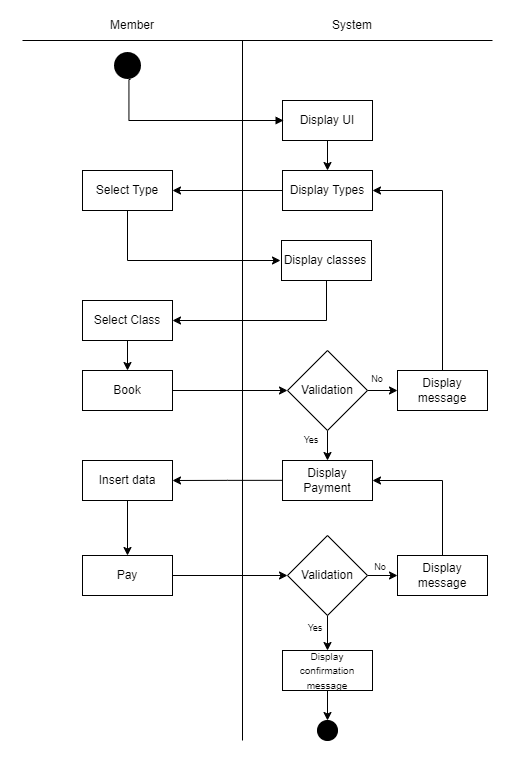
### Make Booking

This function makes and adds details of a booking in the system. Each booking is identified by a unique BookingID.

Member

<<includes>>

<<extends>>



|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Make Booking | |
| **Use Case Id** | 7 | |
| **Priority** | High | |
| **Source** | Member | |
| **Primary Business Actor** | Member | |
| **Other Participating Actors** |  | |
| **Description** | The function adds details of a booking in the Bookings file. | |
| **Preconditions** | The member must be login. | |
| **Trigger** |  | |
| **Expected Scenario** | **Member** | **System** |
|  | **Step 1:** Start Make Booking function.  **Step 4:** Select type.  **Step 6:** Select class.  **Step 7:** Confirm Book.  **Step 10:** Enter required data:   * Card number (int) * Card holder (string) * Expire date (string) * CVC (int)   **Step 11:** Confirm Pay. | **Step 2:** Display UI (Book a class part).  **Step 3:** Retrieve Types from the Class\_Types file.  **Step 5:** Check the date of the classes and **r**etrieve data only today and future classes in the data from the Classes file:   * Class\_ID (int) * Name (string) * Date (string) * Time (string) * Instructor (string) * Price (float)   **Step 8:** Validation:   * Check if AvailablePlaces is not 0. * If the member is already booked the class.   **Step 9:** Display UI (Payment part).  **Step 12:** Validate data:   * All fields must be entered. * Card number cannot be null or empty. * Card number must be numeric. * Card number must contain 16 numbers. * Card Holder cannot be null or empty. * Card holder cannot contain numbers or symbols. * Month must be selected. * Year must be selected. * CVC cannot be null or empty. * CVC cannot contain letters or symbols. * CVC must contain 3 numbers.   **Step 13:** Update availablePlaces in the Classes file.  **Step 14:** Assign BookingID.  **Step 15:** Add data to the Bookings file:   * BookingID (int) * MemberID (int) * ClassID (int) * Card\_number (int) * Card\_holder (string) * Payment\_date (string) * Price (float)   **Step 16:** Confirmation message.  **Step 17:** Update member profile UI add booked class data in data grid:   * Class\_ID * Name (string) * Date (string) * Time (string) * Instructor (string) * Price (float)   **Step 18:** Display Member Profile. |
| **Alternate Scenarios** | **Customer** | **System** |
| **Full Class** |  | **Step 8:** Availability equals 0.  **Step 9:** Display message.  **Step 10:** Return to step 6. |
| **Invalid Bookings details** |  | **Step 12:** Invalid data.  **Step 13:** Display an appropriate message.  **Step 14:** Return to step 10. |
| **Conclusions** | The member booked class. | |
| **Post conditions** | The member can attend the class. | |
| **Business Rules** | The class can be booked if there are available spots. | |
| **Implementation Constraints** |  | |

### Cancel Booking

This function cancel booking in the system.

Member

<<includes>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Cancel Booking | |
| **Use Case Id** | 8 | |
| **Priority** | Low | |
| **Source** | Member | |
| **Primary Business Actor** | Member | |
| **Other Participating Actors** |  | |
| **Description** | The function deletes booking information from the Bookings files. | |
| **Preconditions** | The member must be login. | |
| **Trigger** |  | |
| **Expected Scenario** | **Member** | **System** |
|  | **Step 3:** Select class.  **Step 4:** Confirm Cancel. | **Step 1:** Display UI.  **Step 2:** Retrieve data of the booked classes from Classes file:   * Class\_ID (int) * Name(string) * Date (string) * Time (string) * Instructor (string) * Price (float)   **Step 5:** Display an appropriate message.  **Step 6:** Remove data from Bookings file:   * BookingID (int) * MemberID (int) * ClassID (int) * Card\_number (int) * Card\_holder (string) * Payment\_date (string) * Price (float)   **Step 7:** Change AvailablePlaces in the Classes file**.**  **Step 8:** Confirmation message.  **Step 9:** Display Member profile UI. |
| **Alternate Scenarios** | **Member** | **System** |
| **Invalid request** |  | **Step 6:** Invalid request.  **Step 7:** Display an appropriate message.  **Step 8:** Return to step 3. |
| **Conclusions** | The member canceled booking. | |
| **Post conditions** | The member doesn’t have the class. | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

## Perform Admin

This module provides functions to view all classes, analyses yearly revenue and popular dance styles.

### Classes Overview

This function displays all exciting classes in the system.

Admin

<<includes>>

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Classes Overview | |
| **Use Case Id** | 9 | |
| **Priority** | High | |
| **Source** | Admin | |
| **Primary Business Actor** | Admin | |
| **Other Participating Actors** |  | |
| **Description** | The function displays all existing classes in the Classes file. | |
| **Preconditions** | The admin must be login. | |
| **Trigger** |  | |
| **Expected Scenario** | **Admin** | **System** |
|  | **Step 4:** Select type. | **Step 1:** Display UI.  **Step 2:** Validate if the Class\_Types file Quantity\_of\_classes != 0.  **Step 3:** Retrieve Type from the Class\_Types file.  **Step 5:** Retrieve data from Classes file:   * Class\_ID (int) * Name(string) * Type (string) * Date (string) * Time (string) * AvaliablePlaces (int) * Instructor (string) * Capacity (int) * Price (float) |
| **Alternate Scenarios** | **Admin** | **System** |
| **No data found** |  | **Step 2:** No data found.  **Step 3:** Display an appropriate message.  **Step 4:** Reset UI. |
| **Conclusions** | The admin can view all existing classes in the system. | |
| **Post conditions** | The admin has the option to select the types of classes they wish to view. | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

### Generate Yearly Revenue Analysis

This function performs yearly revenue analyse in the system.

Admin

<<includes>>

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Generate Yearly Revenue Analysis | |
| **Use Case Id** | 10 | |
| **Priority** | Medium | |
| **Source** | Admin | |
| **Primary Business Actor** | Admin | |
| **Other Participating Actors** |  | |
| **Description** | The function performs yearly revenue analyse in the selected year and display result. | |
| **Preconditions** | The admin must be login. | |
| **Trigger** |  | |
| **Expected Scenario** | **Admin** | **System** |
|  | **Step 1:** Start Generate Yearly Revenue Analysis function.    **Step 5:** Select year.  **Step 6:** Confirm Calculate. | **Step 2:** Display UI.  **Step 3:** Validate if data exists in the Bookings file.  **Step 4:** Insert Year from the code.  **Step 7:** Calculate yearly revenue using data from the Bookings file:   * Price (float) * Payment\_date (string)   **Step 8:** Generate message with yearly revenue.  **Step 9:** Create chart with the analysis. |
| **Alternate Scenarios** | **Admin** | **System** |
| **No data found** |  | **Step 5:** No data Found.    **Step 6:** Display message.  **Step 7:**  Reset UI. |
| **Conclusions** | The yearly revenue was calculated. | |
| **Post conditions** | The admin can see yearly revenue. | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

Example of Yearly Revenue Analysis

### Generate Popular Dance Style Analysis

This function performs Popular Dance Style Analyse in the system.

Admin

<<includes>>

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Generate Popular Dance Style Analysis | |
| **Use Case Id** | 11 | |
| **Priority** | Medium | |
| **Source** | Admin | |
| **Primary Business Actor** | Admin | |
| **Other Participating Actors** |  | |
| **Description** | The function performs Popular Dance Style Analyse in the system and displays result. | |
| **Preconditions** | The admin must be login. | |
| **Trigger** |  | |
| **Expected Scenario** | **Admin** | **System** |
|  | **Step 1:** Start Generate Popular Dance Style Analysis function.  **Step 3:** Confirm Find. | **Step 2:** Display UI.  **Step 4:** Validate if the Class\_Types file Quantity\_of\_classes != 0.  **Step 5:** Calculate Popular Dance Style using data from the Class\_Types file:   * Quantity\_of\_classes (int)   **Step 6:** Display message with result.  **Step 7:** Create a chart with the analysis. |
| **Alternate Scenarios** | **Admin** | **System** |
| **No data found** |  | **Step 3:** No data found.  **Step 4:** Display error message.  **Step 5:** Reset UI. |
| **Conclusions** | The Popular Dance Style was calculated. | |
| **Post conditions** | The admin can see what the most popular dance style is. | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

Example of Popular Dance Styles Analysis

## Authentication

This module provides functions to login Member and Admin and allow to member recover password.

### Login Member

The function login member in the system.

Member

<<includes>>

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Login Member | |
| **Use Case Id** | 12 | |
| **Priority** | High | |
| **Source** | Member | |
| **Primary Business Actor** | Member | |
| **Other Participating Actors** |  | |
| **Description** | The function login member and check if the member exists in the Members file. | |
| **Preconditions** | The member must have an account. | |
| **Trigger** |  | |
| **Expected Scenario** | **Member** | **System** |
|  | **Step 2:** Enter required data:   * Email (string) * Password (string)   **Step 3:** Confirm Log In. | **Step 1:** Display UI.  **Step 4:** Validate data with Members file:   * Email (string) * Password (string)   **Step 5:** Display Member Profile UI. |
| **Alternate Scenarios** | **Member** | **System** |
| **Invalid data** |  | **Step 4:** Invalid data.  **Step 5:** Display an appropriate error message.  **Step 6:** Reset UI. |
| **Conclusions** | The member logged in in the system. | |
| **Post conditions** | The member can book class. | |
| **Business Rules** | The member must exist in the Members file. | |
| **Implementation Constraints** |  | |

### Login Admin

The function login admin in the system.

Admin

<<includes>>

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Login Member | |
| **Use Case Id** | 13 | |
| **Priority** | High | |
| **Source** | Admin | |
| **Primary Business Actor** | Admin | |
| **Other Participating Actors** |  | |
| **Description** | The function login admin and check if the admin exists in the Members file. | |
| **Preconditions** | The admin must exist in the Members file. | |
| **Trigger** |  | |
| **Expected Scenario** | **Admin** | **System** |
|  | **Step 2:** Enter required data:   * Email (string) * Password (string)   **Step 3:** Confirm Log In. | **Step 1:** Display UI.  **Step 4:** Validate data with Members file:   * Email (string) * Password (string)   **Step 5:** Display Classes Overview UI. |
| **Alternate Scenarios** | **Admin** | **System** |
| **Invalid data** |  | **Step 4:** Invalid data.  **Step 5:** Display an appropriate error message.  **Step 6:** Reset UI. |
| **Conclusions** | The admin logged in in the system. | |
| **Post conditions** | The admin can interact with the system. | |
| **Business Rules** | The admin must exist in Members file. | |
| **Implementation Constraints** |  | |

### Recover Member Password

The function allows member to reset password of account.

Member

<<includes>>

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Recover Member Password | |
| **Use Case Id** | 14 | |
| **Priority** | Medium | |
| **Source** | Member | |
| **Primary Business Actor** | Member | |
| **Other Participating Actors** |  | |
| **Description** | The function allows member to reset password of an account. The system will update member password in the Members file. | |
| **Preconditions** | The member must have an account. | |
| **Trigger** |  | |
| **Expected Scenario** | **Member** | **System** |
|  | **Step 1:** Start Recover member Password function.  **Step 3:** Enter the required Data:   * Email (string) * New Password (string) * Confirm Password (string) | **Step 2:** Display UI.  **Step 4:** Validate data:   * Email must exist in the Members file. * Email cannot be update if it contains word ‘admin’ * Confirm Password must be the same as New Password.   **Step 5:** Update password in the Members file.  **Step 6:** Display confirmation message.  **Step 7:** Display log in UI. |
| **Alternate Scenarios** | **Member** | **System** |
| **Invalid data** |  | **Step 4:** Invalid data.  **Step 5:** Display an appropriate message.  **Step 6:** Reset UI. |
| **Conclusions** | The member changed password. | |
| **Post conditions** | The member can login with new password. | |
| **Business Rules** | The email must exist in the Members file. | |
| **Implementation Constraints** |  | |

# System Model

The following dataflow diagrams have been produced for the system, demonstrating interactions and processes involving actor Member with the system. The processes include: Add member, Update Member, Delete Member, Schedule Class, Modify class, Cancel class, Make Booking, Cancel Booking, Classes Overview, Generate yearly Revenue Analysis, Generate Popular dance Style Analysis, Login Member, Login Admin, Recover Member Password.

## Level-0 DFD

Dance Studio Booking System

Member

Booking details

Class details

## Level-1 DFD

Types details

Instructors file

Class

details

P2

Manage Classes

P4

Perform Admin

D2

Classes file

D3

Bookings file

Bookings

details

P1

Manage Members

P3

Process Bookings

Member

P5

Authentication

D1

Members file

Members details

Perform booking

Member/Admin

details

Members

details

MemberID

ClassID

Class details

ClassID

Login

Show member details

D4

Instructors details

D5

Class\_Types file

Types details

Types details

## Level-2 DFD (Process P1: Manage Members)

Member details

P1.2

Update Member

P1.3

Delete Member

Members file

D1

P1.1

Add Member

Member

Member details

Member details

MemberID

Member details

Update Member details

Delete member’s data

Appropriate message

Update details request

Member

Appropriate message

Appropriate message

Delete details request

## Level-2 DFD (Process P2: Manage Classes)

Instructor details

Instructors file

D5

Class\_Types file

Types details

D4

P2.2

Modify Class

P2.3

Cancel Class

Classes file

D2

P2.1

Schedule Class

Class details

ClassID

Class details

Updated Class details

Class details

Delete class details

Instructor details

Types details

## Level-2 DFD (Process P3: Process Bookings)

D5

Class\_Types file

Classes file

Cancel booking request

Appropriate message

Appropriate message

Member

P3.1

Make Booking

P3.2

Cancel Booking

Bookings file

D3

Booking details

Booking details

Booking details

Delete data

D2

Class details

Types details

## 5.6 Level-2 DFD (Process P4: Perform Admin)

P4.2

Generate Yearly Revenue Analysis

P4.3

Generate Popular Dance Styles Analysis

Bookings file

D3

Class\_Types file

D5

Price

Payment date

Quantity of classes

P4.1

Classes Overview

Classes details

D2

Classes file

## 5.7 Level-2 DFD (Process P5: Authentication)

System access

Member

P5.1

Login Member

Member details

Member details

P5.3

Recover Member Password

Members file

D1

P5.2

Login Admin

Admin details

Member details

New password

Appropriate message

# Data Model (Class Diagram)

The Class Diagram of Dance Studio Booking System includes classes such as Bookings, Classes, Members and Instructors. The Members class is connected to the Bookings class, allowing members to make bookings. The Bookings class is associated with the Classes class, representing the booked dance classes. The Classes class is connected with the Instructors class.

## Class Diagram

Instructors

- Instructor\_ID\*: int

- FullName: string

- Email: string

- Phone: int

1

Bookings

- Booking\_ID\*: int

- Card\_holder: string

- Card\_number: int

- Payment\_date: date

- Price: float

Members

- Member\_ID\*: int

- Firstname: string

- Lastname: string

- Gender: string

- Email: string

- DOB: date

- Phone: int

- Password: string

Classes

- Class\_ID\*: int

- Name: string

- Date: date

-Time: string

- AvialablePlaces: int

- Capacity: int

- Price: float

0..\*

Make booking

Is has

1

0..\*

has

0..\*

1

Class\_Types

- Type\_ID\*: int

- Type: string

- Qty\_of\_classes: int

Is has

1

0..\*

## Relational Schema

**Classes (**Class\_ID, Name, DateCode, AvialablePlaces, Capacity, Price, AvialableTime, Instructor\_ID, Type\_ID**)**

**Instructors (**Instructor\_ID, FullName, Email, Phone**)**

**Class\_Types (**Type\_ID, Type, Qty\_of\_classes**)**

**Members (**Member\_ID, Firstname, Lastname, Gender, Email, DOB, Phone, Password**)**

**Bookings (**Booking\_ID, Card-holder, Card\_number, Price, Payment\_date, Member\_ID, Class\_ID**)**

## Database Schema

Schema: DanceStudioBookingSYS

Relation: Members

Attributes:

Member\_ID: numeric (5)

Firstname: string (25) NOT NULL

Lastname: string (25) NOT NULL

Gender: string (8) NOT NULL

Email: string (50) NOT NULL

Phone: numeric (12) NOT NULL

DOB: Date NOT NULL

Password: string (20) NOT NULL

Primary Key: Member\_ID

Relation: Classes

Attributes:

Class\_ID: numeric (5)

Name: string (30) NOT NULL

DateCode: Date NOT NULL

TimeCode: string (6) NOT NULL

AvaliablePlaces: numeric (2) NOT NULL

Capacity: numeric (2) NOT NULL

Price: numeric (5,2) NOT NULL CHECK>0

Instructor\_ID: numeric (5)

Type\_ID: varchar2(2)

Primary Key: Class\_ID

Foreign Key: Instructor\_ID references Instructors

Foreign Key: Type\_ID reference Types

Relation: Instructors

Attributes:

Instructor\_ID: numeric (5)

FullName: string (50) NOT NULL

Email: string (50) NOT NULL

Phone: numeric (13) NOT NULL

Primary Key: Instructor\_ID

Relation: Class\_Types

Attributes:

Type\_ID: string (2)

Type: string (10) NOT NULL

Qty\_of\_classes: numeric (3)

Primary Key: Type\_ID

Relation: Bookings

Attributes:

Booking\_ID: numeric (5)

Card\_holder: string (25) NOT NULL

Card\_number: numeric (16) NOT NULL

Payment\_date: Date NOT NULL

Price: numeric (5,2) NOT NULL CHECK>0

Member\_ID: numeric (5)

Class\_ID: numeric (5)

Primary Key: Booking\_ID

Foreign Key: Member\_ID references Members

Foreign Key: Class\_ID references Classes

# Conclusion

The core components of a Dance Booking System – Members, Classes, Bookings, Administration, and Authentication-form the backbone of the system, ensuring smooth studio operations.

A thorough examination of system requirements, identified the core functionality needed to manage Members, Classes, Bookings, Administrative tasks, and Authentication procedure, ensured that the system could effectively meet the diverse needs of its members.

Visualizing system processes using data flow diagrams (DFDs) provided a clear and concise view of how the system interacts with its main actor – Member. These diagrams outline critical processes including member profile management, class booking, and administrative functions to ensure a streamlined experience.

The Data Model, specifically depicted through a class diagram, demonstrates the relationships between classes such as Bookings, Classes, Members and Instructors.

In conclusion, through meticulous planning and hard work, I have achieved my goal for the project.