PES UNIVERSITY
RR CAMPUS

UE20CS352
OOAD MINI-PROJECT
REPORT

TITLE

CALENDAR MANAGEMENT SYSTEM

TEAM DETAILS:

1. Abhijeet Umakant Wadiyar: PES1UG20CS006
2. Aditya Mahesh Hombal: PES1UG20CS020
3. Aneela Gurram: PES1UG20CS049
4. Animesh Agarwal: PES1UG20CS053

SYNOPSIS

Introduction:

The Calendar Management System is a Java-based desktop application that allows users to manage their personal and professional schedules. The system enables users to create, edit and delete events on their calendars, set reminders and notifications, and view schedules in different formats such as day, week, and month.

The system uses a GUI interface - SWING to interact with the user and stores the event data in a serialised format using the SerializableStorage class. The data is organised using a Hashtable data structure, which allows for efficient retrieval and modification of events.

The system also includes a feature for importing and exporting calendar data in CSV format, making it easy to share event data with other users or applications. Additionally, the system provides a backup and restore functionality, allowing users to safeguard their calendar data in case of data loss or corruption.

Overall, the Calendar Management System is a user-friendly and efficient application that provides a simple yet effective way for users to manage their schedules and stay organised.

Purpose:

The purpose of this project is to build a calendar management system. Calendars are useful tools for keeping track of upcoming meetings, deadlines, and milestones. They can help us visualise our schedule and remind us of important events, such as holidays and vacation time.

Scope:

The purpose of this project is to place all tasks for an individual's multiple projects or tasks and appointments in one location. Additionally the user must be able to see whether the calendar for any month of any year.
Above all, we hope to provide a nice user experience.

Product Perspective:

Calendars are one of the basic necessities for anyone. Integrating an event management system with a calendar is easily one of the best applications for a calendar management system. There are many existing event management systems and calendars but this project integrates these features in a single location. This could further be bundled with other products targeting office management somewhat like microsoft office.

Product Functions:

The major functions for this calendar management system are:

1. ***Ability to add and manage events/tasks***
2. ***Ability to view the calendar for a given month/year.***
3. ***Ability to view personal time plots.***
4. ***Ability to View people attending a particular meeting.***
5. ***Save and Load data***

ARCHITECTURE PATTERN :

MVC Architecture Pattern :

MVC stands for Model-View-Controller,a software design pattern that separates the application logic into three interconnected components

Model represents the data and the business logic. View represents the user interface.

Controller acts as an intermediary between the Model and the View.

This pattern is used in web and mobile applications as it provides a clear separation of concerns and promotes modularity, reusability, and scalability.

Advantages of MVC Pattern :

Separation of Concerns:** **By dividing the application into three separate components, the MVC pattern ensures that each component is responsible for a specific task, reducing the complexity of the application.

Modularity and Reusability:** **The separation of concerns in the MVC pattern makes it easier to reuse and modify the components without affecting the others.

Testability:**The MVC pattern makes it easier to test the application as each component can be tested separately, reducing the risk of introducing bugs into the application.

Scalability:**The MVC pattern promotes modularity and reusability, making it easier to add new features and scale the application as needed.

Framework used:

***Java Swing** is a graphical user interface (GUI) widget toolkit for Java. It provides a set of components, such as buttons, text boxes, and menus, that can be used to create interactive graphical applications in Java. Swing was developed as part of the Java Foundation Classes (JFC) and has been a part of Java since Java 2 (JDK 1.2) was released in 1998.*

Swing provides a powerful set of features for creating complex user interfaces, including support for layout managers, event handling, and custom graphics. It is platform-independent, which means that Swing applications can run on any platform that supports Java.

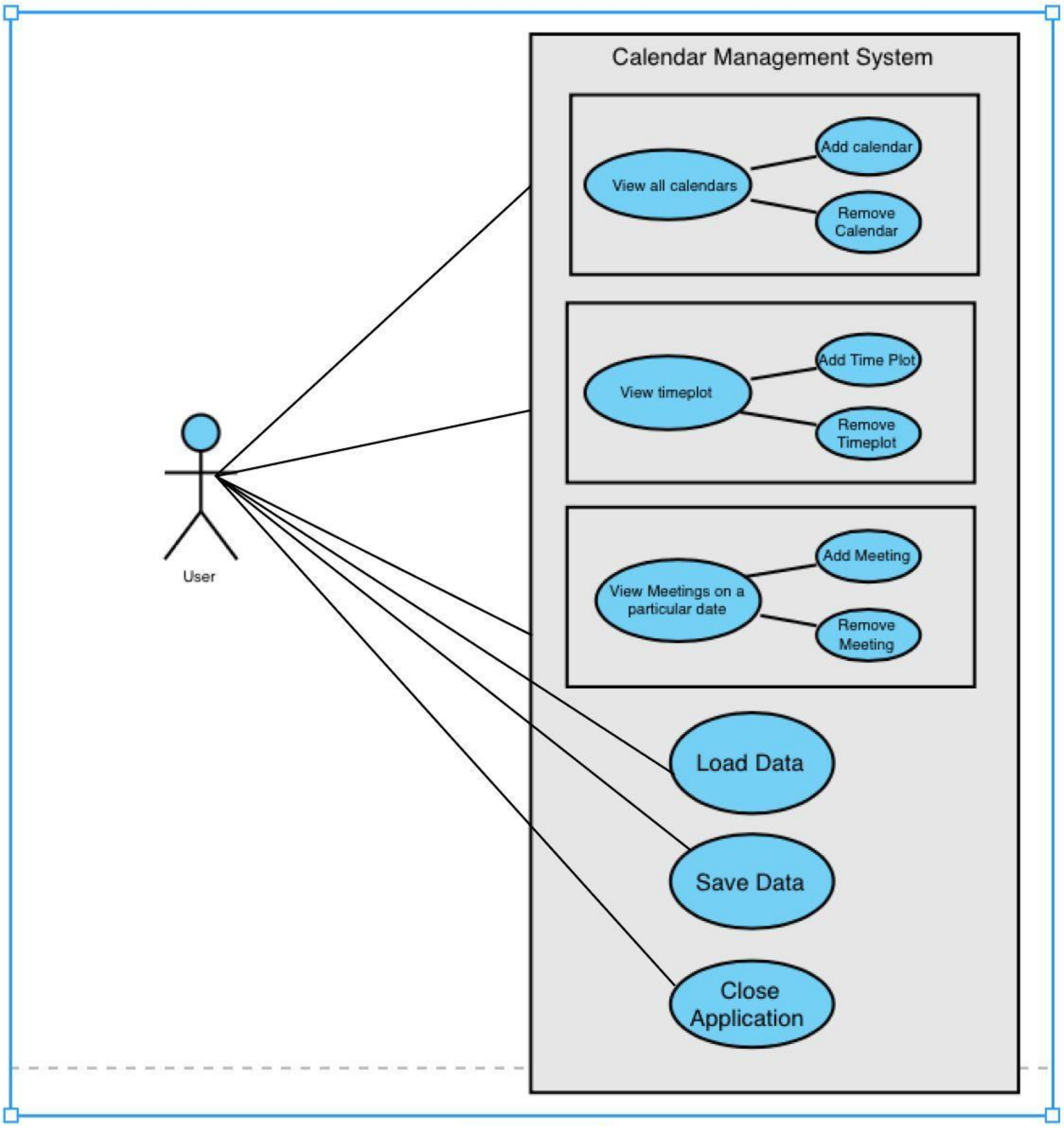
Swing has been used to create a wide range of applications, including desktop applications, web applications, and mobile applications. It has become a popular choice for developing desktop applications in Java because of its rich set of features and its ability to create modern and visually appealing user interfaces.

System Features :

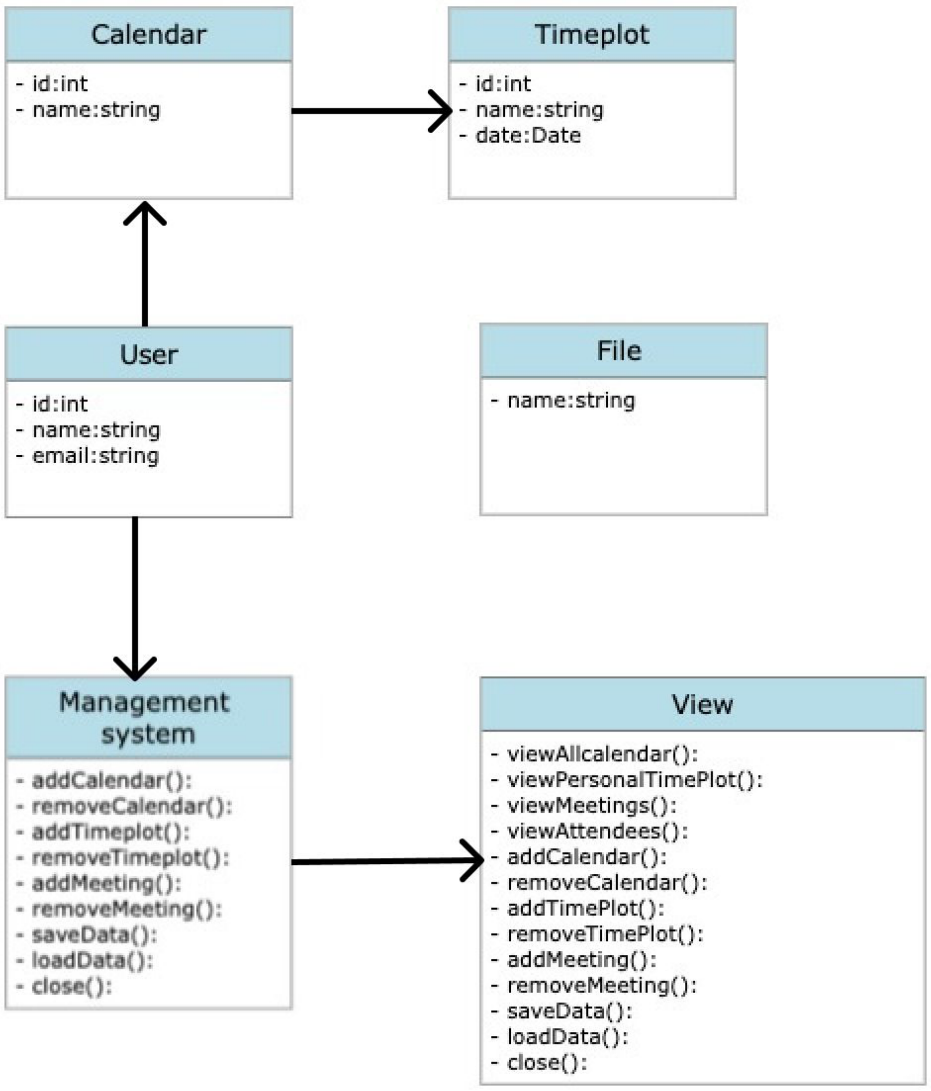
1. ***View all calendars:* This use case allows the user to view all the calendars present in the system. This use case is initiated when the user opens the EMS application or clicks on the "Show All Calendars" button.**
2. ***View personal time plot:* This use case allows the user to view the personal time plot for a particular calendar. This use case is initiated when the user clicks on the "TimePlots"-"Personal" option.**
3. ***View meetings on a particular date:* This use case allows the user to view all the meetings scheduled on a particular date. This use case is initiated when the user clicks on the "Meetings"-"YYYYMMDD" option.**
4. ***View people attending a particular meeting:* This use case allows the user to view all the people attending a particular meeting. This use case is initiated when the user clicks on the "Meeting"-"HHMM" option.**
5. ***Add calendar:* This use case allows the user to add a new calendar to the system. This use case is initiated when the user clicks on the "Add +" button under the "Calendars"-"All" section.**
6. ***Remove calendar:* This use case allows the user to remove an existing calendar from the system. This use case is initiated when the user clicks on the "Remove -" button under the "Calendars"-"All" section.**
7. ***Add time plot:* This use case allows the user to add a new time plot to an existing calendar. This use case is initiated when the user clicks on the "Add +" button under the "Time Plots"-"somename" section.**
8. ***Remove time plot:* This use case allows the user to remove an existing time plot from an existing calendar. This use case is initiated when the user clicks on the "Remove -" button under the "Time Plots"-"some name" section.**

9. ***Add meeting:* This use case allows the user to add a new meeting to an existing time plot. This use case is initiated when the user clicks on the "Add +" button under the "Meetings"-"somedate" section.**
10. ***Remove meeting:* This use case allows the user to remove an existing meeting from an existing time plot. This use case is initiated when the user clicks on the "Remove -" button under the "Meetings"-"somedate" section.**
11. ***Save data:* This use case allows the user to save the data present in the system to a file. This use case is initiated when the user clicks on the "Save" button under the "Left" section.**
12. ***Load data:* This use case allows the user to load the data present in a file into the system. This use case is initiated when the user clicks on the "Load =" button under the "Right" section.**
13. ***Close application:* This use case allows the user to close the EMS application. This use case is initiated when the user clicks on the "Quit" button under the "Left" section.**

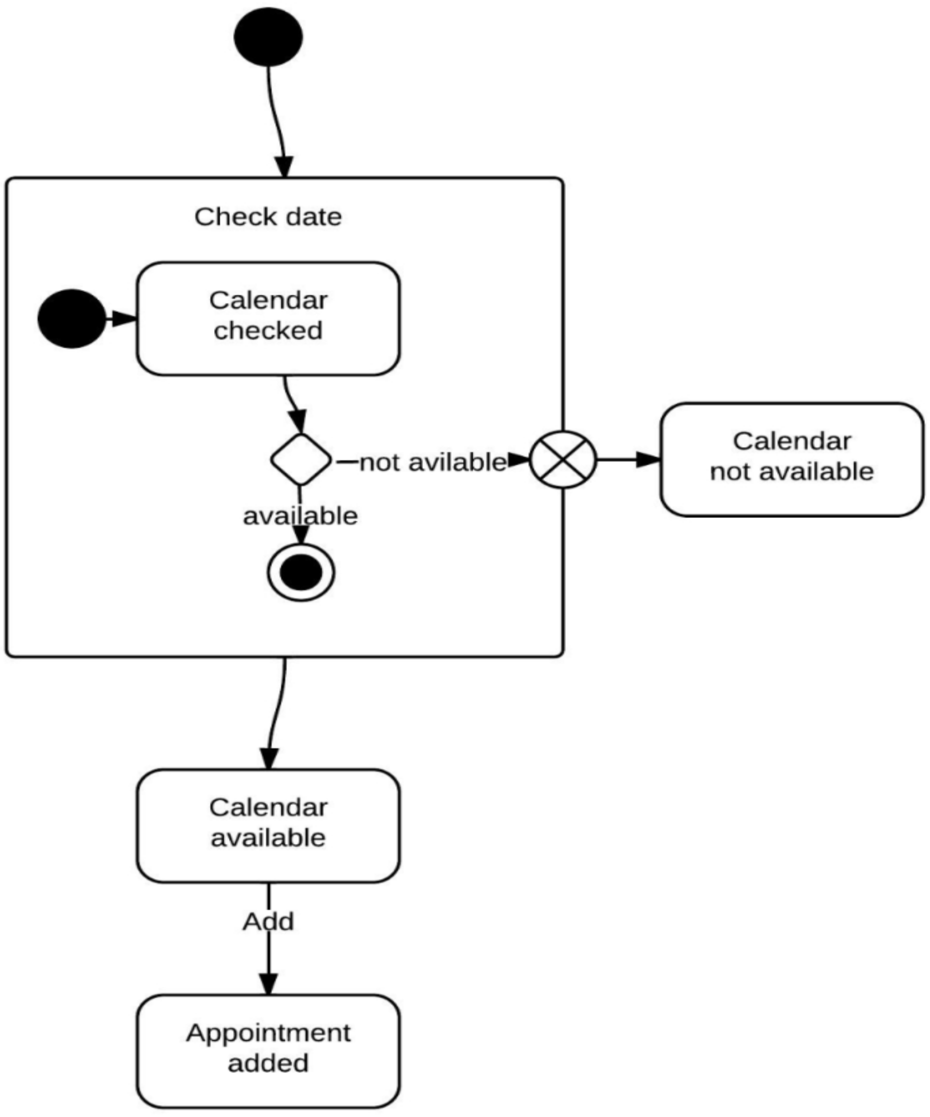
USE CASE DIAGRAM:



CLASS DIAGRAM:



State Diagram:



Design Principles Used:

*1. Single Responsibility Principle (SRP): **Each class or method in the code has a single responsibility or reason to change. For example, the ShowController class is responsible for handling user interactions and displaying the appropriate data on the view, while the Calendar, Timeplot, and Meeting classes are responsible for modelling the data.***

*2. Separation of Concerns: **The code separates the concerns of the model, view, and controller into different packages, making it easier to understand and maintain.***

*3. Encapsulation: **The Calendar, Timeplot, and Meeting classes encapsulate their data and behaviour, preventing direct access to their internal state and ensuring that their state is consistent.***

*4. Abstraction: **The Calendar, Timeplot, and Meeting classes are abstractions of the data they model, providing a simplified and structured representation of the data.***

*5. Dependency Inversion Principle: **The higher-level modules in the system, such as the CalendarManager and CalendarController, depend on the abstract interfaces rather than concrete implementations. This makes the system more flexible and easier to maintain.***

*6. Liskov Substitution Principle: **The subclasses of the Calendar abstract class seem to be substitutable for the base class, as they all adhere to the same interface and functionality.***

Design Patterns Used:

1. Model-View-Controller (MVC) pattern: This pattern separates the application into three main components: the Model (data and business logic), the View (user interface), and the Controller (handles user input and updates the Model and View).

2. Singleton pattern: This pattern is used to ensure that only one instance of certain classes (such as the Calendar and TimePlot classes) is created throughout the application.

3. Observer pattern: The Observer pattern is used to maintain a list of dependents and to notify them automatically of any changes to the subject. This pattern is used in the Meeting class, where the Observer interface is implemented to update the list of attendees when a meeting is added or removed from the system.

4. Builder Pattern:** In this code, the Builder pattern allows the* *Meeting* ***class to be instantiated with optional parameters using a fluent API rather than having to specify all parameters in the constructor