TRACTIVITY

Workshop #2

Sergio Andrés Díaz Cuervo - 20251020166

Johan Esmit Sichacá González - 20242020313

John Mario Jiménez Becerra – 20251020047

Professor: Carlos Andres Sierra Virguez

Subject: OOP

Conceptual design updates

Introduction

Within both academic and business contexts, it is always of vital importance to maintain a high level of organization regarding events and tasks. It is no secret that, due to the many daily responsibilities, people often completely forget about events or tasks that were planned long ago. Therefore, an application capable of solving this issue would represent a significant improvement in both cases.

For this reason, TRACKTIVITY was created, an application designed to be the personal assistant or planner for anyone, from high school or university students to business professionals who need to remember or plan many things at once. Within this app, the user, after registering, has access to two calendars where they can create both events and tasks, as well as share their active calendars with other users. In the app's ecosystem, the user will find dynamic and easy-to-use interfaces, making them feel in a useful and productive space.

Chapter I: Requirements

1. Functional Requirements

1.1 User Management

- New users can create an account and register.
- Users who are already registered can log in and be remembered by the application.
- The user will have full access to modify their profile information, which includes:
  a) Name
  b) Profile picture

1.2 Activity and Event Registration

- The user can create events at will (with a maximum of 15 per day).
- When creating an event, the user can assign the following details:
  a) Event date
  b) Event name
  c) Brief event description (optional)
  d) Set a specific time or a time range during which the event will take place
  e) Task duration (hours and minutes)
- The user will be able to view the event on the calendar after its creation.
- The user can create a task list with the following parameters:
  a) Task name
  b) Due or expiration date
  c) Category (subject or topic)
  d) Mark the task status (Completed or Incomplete)

- The user will be able to view their pending tasks as “Tasks” on their calendar.

1.3 Event Planning

- After completing their registration, the user will have access to the creation of up to **two calendars**:
  a) The first one, called **“Personal”**, where the events created for this calendar are displayed.
  b) The second one corresponds to tasks, called **“Tasks”**, where only the tasks created by the user will be shown.
  c) Finally, the user will have access to a **“Mixed”** view, in which they can simultaneously see both the personal calendar and the pending tasks.

1.4 Database Connectivity

- The user will have access to more than one task list (differentiated by name and category) and can choose which one to import into their calendar (only one task list can be viewed at a time).
- When the user creates an event in the calendar, it will also be visible outside the calendar in a section called **“Events”.**

1.1 Data Sharing

- The user will be able to share their calendar with other people; this shared calendar will be available for viewing only.

2. Functional Requirements

2.1 Performance

- It is estimated that the application can be used simultaneously by the entire class, so it is expected to support at least 18 users at the same time.
- Since the application mostly provides users with a more organized view of their events, it should not perform heavy computations. Therefore, the expected response time for both events and tasks should be a maximum of 2 to 3 seconds.
- When importing a database, the program undergoes several internal processes, such as reading the file to be imported, validating and transforming the data, processing it in memory, and finally rendering the calendar. Hence, an interval of 8 to 10 seconds at most is estimated for this process.

2.2 Security

- All passwords created by users must remain private, preventing access by any user other than their owner.
- Usernames will only be visible to users with whom the calendar has been shared.

2.3 Availability

- The software is expected to be available **99% of the time** for all users.

2.4 Usability

- The system interface must be intuitive and allow a new user to design their task list or calendar easily.
- The calendars must be presented in a simple format, without complicating the reading or understanding of dates or data.
- The lists should be as dynamic as possible, avoiding information overload and visual clutter.

2.5 Maintainability

- The code must be properly documented, following the proposed standards and maintaining appropriate organization.
- The program must be designed in a modular way to facilitate future updates.

User Stories

Title: User register	Priority: High	Estimate: 20/10/2025
User Story: As a new user I want to register in the application So that I can start making my activities		
Acceptance Criteria: Given that the user accesses the application for the first time When they complete the registration correctly and submit it Then the user sees a visual confirmation that their account has been successfully created and can proceed to the login screen.		

Title: Login	Priority: High	Estimate: 20/10/2025
User Story: As a user I want to log in with my credentials So that I can Access my calendars and task		
Acceptance Criteria: Given that the user already has a registered account When they user enter valid credentials Then the user gains access to their main dashboard and can view their personal calendar		

Title: Event creation	Priority: High	Estimate: 22/10/2025
User Story: As a user I want to create an event with date, name, and description So that I can organize my activities		
Acceptance Criteria: Given that the user is in their calendar view, When they select the “Create Event” option and enter the required information (name, date, description), Then the user can see the new event displayed on the selected date within the calendar		

Title: Task list within an event	Priority: High	Estimate: 25/10/2025
User Story: As a user I want to create a task list within an event So that I can track specific subtasks		
Acceptance Criteria: Given that the user already has a created event, When they add tasks and assign a status (pending or completed), Then the user can view the list of tasks within the event and observe the progress of each according to its status		

Title: Pending tasks view	Priority: High	Estimate: 27/10/2025
User Story: As a user I want to see all my pending tasks in a single list So that I know what I still need to do		
Acceptance Criteria: Given that the user has multiple events with tasks, When they open the “Pending” view, Then the user sees a list organized by date showing all tasks that are not yet completed, allowing them to prioritize their work		

Title: Event modification	Priority: High	Estimate: 29/10/2025
User Story: As a user I want to edit the name or date of an event So that I can fix it without creating a new one		
Acceptance Criteria: Given that the user has an existing event in their calendar, When they open the edit option, modify the data, and save the changes, Then the user can see the updated event in their calendar with the new information		

Title: Calendar Visualization	Priority: High	Estimate: 31/10/2025
User Story: As a user I want to view my events and task in a calendar interface So that I can easily visualize my schedule and deadlines		
Acceptance Criteria: Given that the user has already created events or tasks, When they access the calendar view, Then the user can see all scheduled events and pending tasks displayed on their corresponding dates.		

Title: Multiple Task List	Priority: High	Estimate: 03/11/2025
User Story: As a user I want to create and manage multiple task list So that I can organize my work by subjects or projects		
Acceptance Criteria: Given that the user is logged into the system, When they create new task lists and name them by topic or category, Then the user can select which task list to display on the calendar at a given time		

Title: Share calendar	Priority: Low	Estimate: 05/11/2025
User Story: As a user I want to share my calendar with other users So that they can view my activities		
Acceptance Criteria: Given that the user has created events in their calendar, When they select the “Share” option and enter the recipient’s email, Then the receiving user gets read-only access and can view the shared calendar but cannot modify it.		

CRC Cards

Class: User	
Responsabilities	**Collaborator**
▪ Log in, Sign in ▪ Assign tasks ▪ Store profile data	Task Notification Progress statistic

▪ Manage preferences	Check list Register

Class: Notification	
Responsabilities	**Collaborator**
▪ Notify the user ▪ Inform about the expiration of an event	User Task Calendar

Class Mom: Schedulable	
Responsabilities	**Collaborator**
▪ Has task data like name, description, priority, date, etc ▪ Cr	User Notifications Progress statistics

Child Class: Habit	
Responsabilities	**Collaborator**
▪ Has additional attributes like frecuency, streak of days	Calendar Progress statistic Notification Check List

Child Class: Task	
Responsabilities	**Collaborator**
▪ Has additional attributes like submission date, percentage progress, etc.	Calendar Progress statistic Notification Check List

Child Class: Event	
Responsabilities	**Collaborator**
▪ Has additional attributes like start hour and end hour	Calendar Progress statistic Notification Check List

Class: Progress Statistic	
Responsabilities	**Collaborator**
▪ Create daily, weekly and monthly statistics; Show tasks missing progress; Show task percentage completed	User Task Check list Calendar Notification

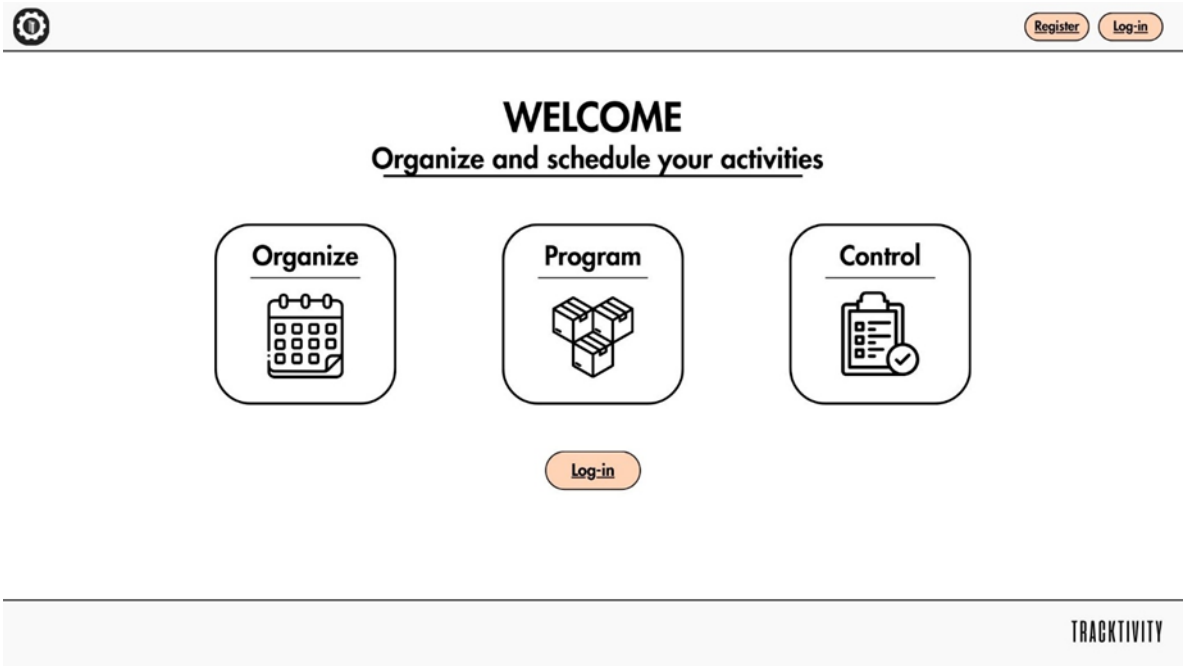
Class Mom: Calendar	
Responsabilities	**Collaborator**
▪ Controls general calendar management, the active date, and switching between views (daily, weekly, or monthly). ▪ Coordinates information between tasks, habits, and different views.	Day Week Month User Notification Check List

Child Class: Day	
Responsabilities	**Collaborator**
▪ It represents the calendar view focused on a single day, showing in detail the tasks, habits and scheduled events of the selected day.	Calendar Task Notification User Check List

Child Class: Week	
Responsabilities	**Collaborator**
▪ Shows tasks, habits and events distributed throughout the selected week	Calendar Task Notification User Check List

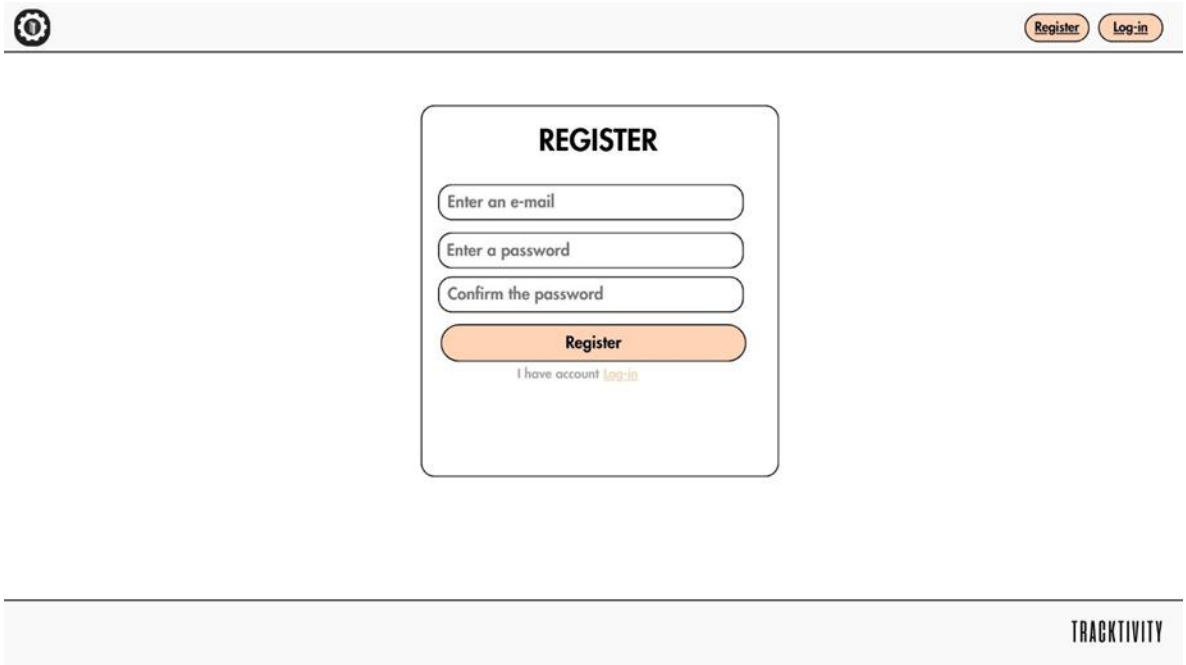
Child Class: Month	
Responsabilities	**Collaborator**
▪ Provides an overview of the month, showing dates with tasks and habits of the selected month	Calendar Task Notification User Check List

Mockups



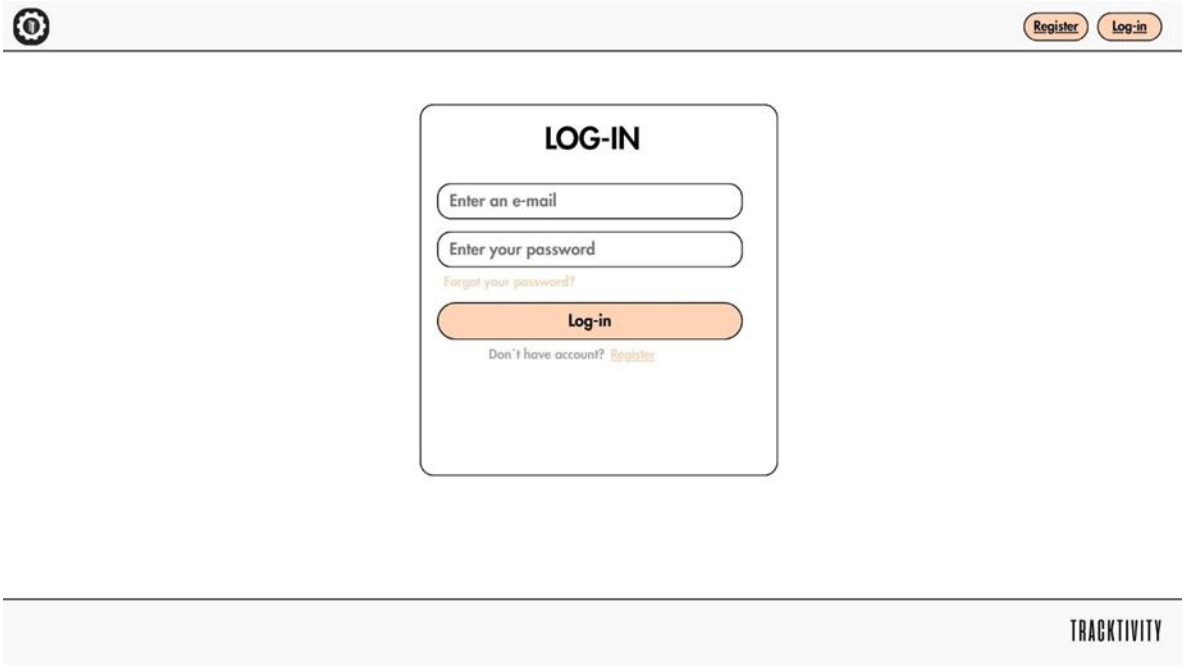
Mockup 1

Home: Displays the name of the app and provides access to log in or register, guiding the user to the main home page of the app



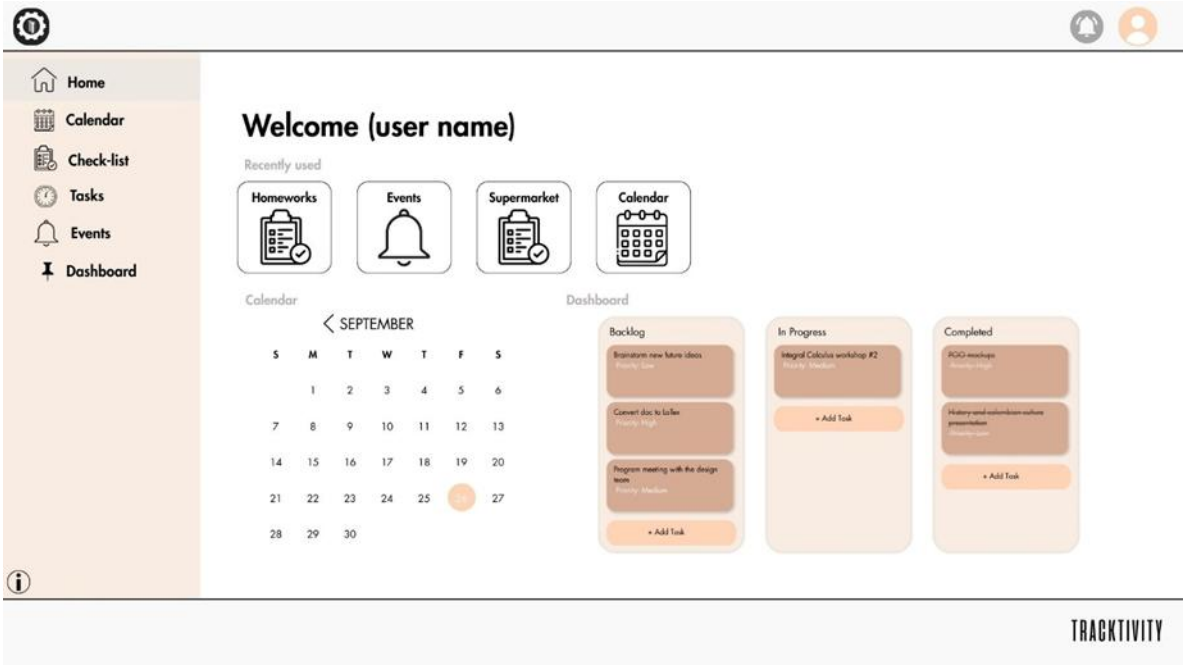
Mockup 2

Registration: Presents a form to create a new account



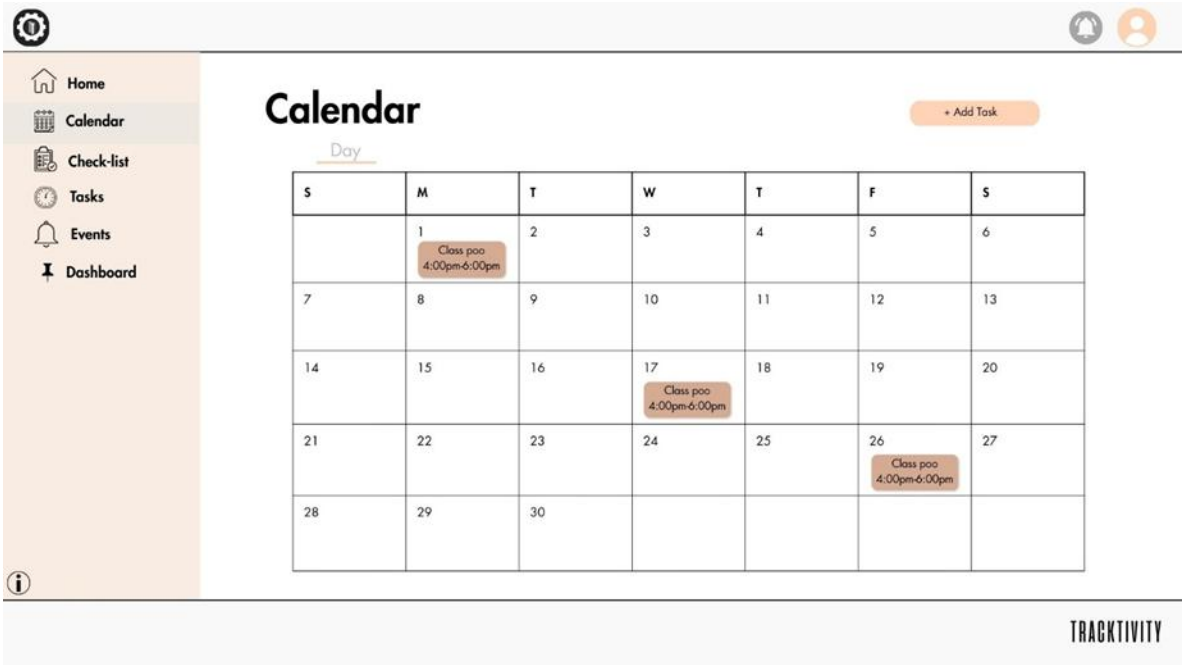
Mockup 3

Login: Allows you to enter your email and password to access the System



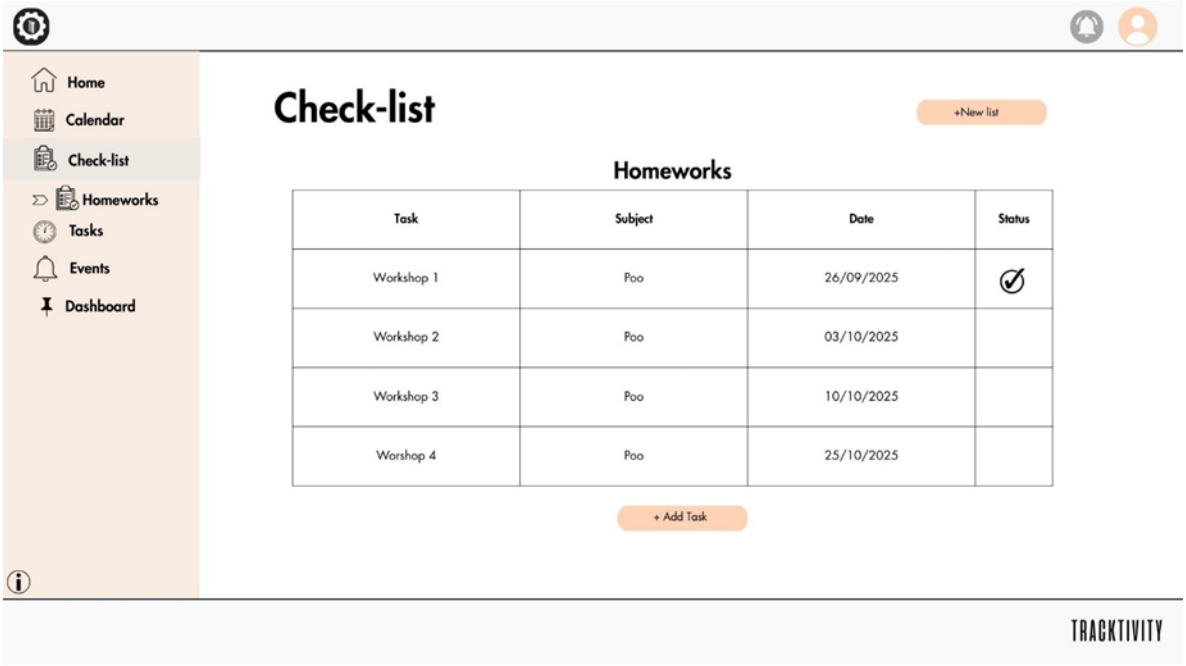
Mockup 4

Main screen: Displays a greeting, shortcuts to recently used sections, calendar view and task board



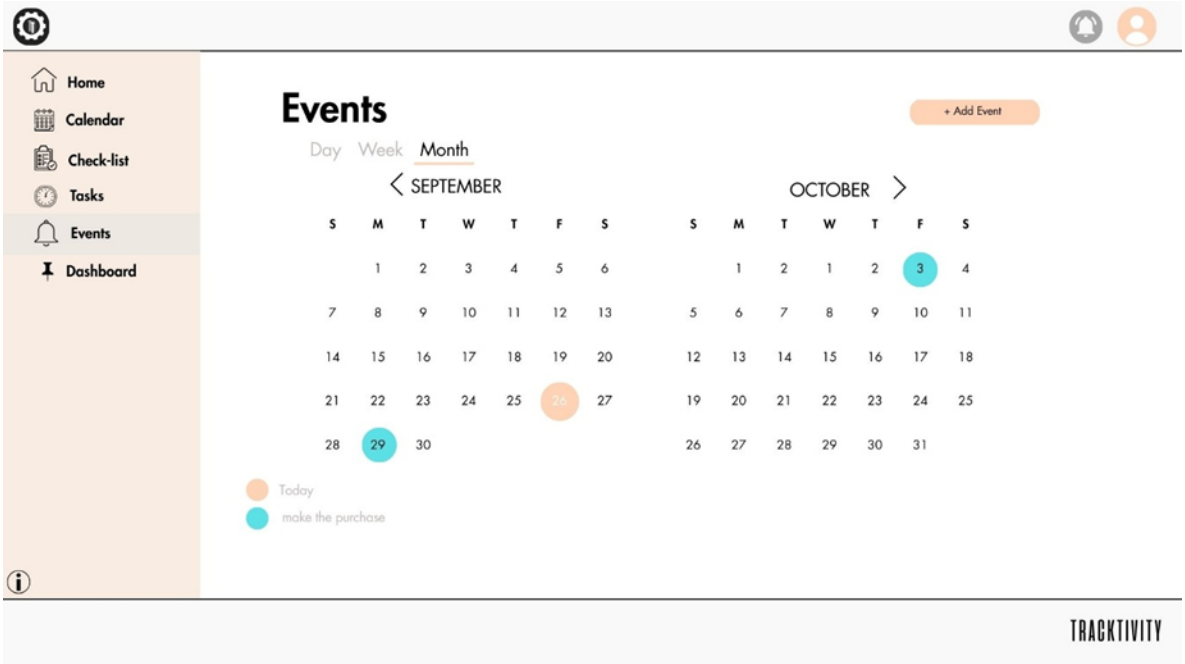
Mockup 3

Calendar: Displays the user's events and tasks organized by date



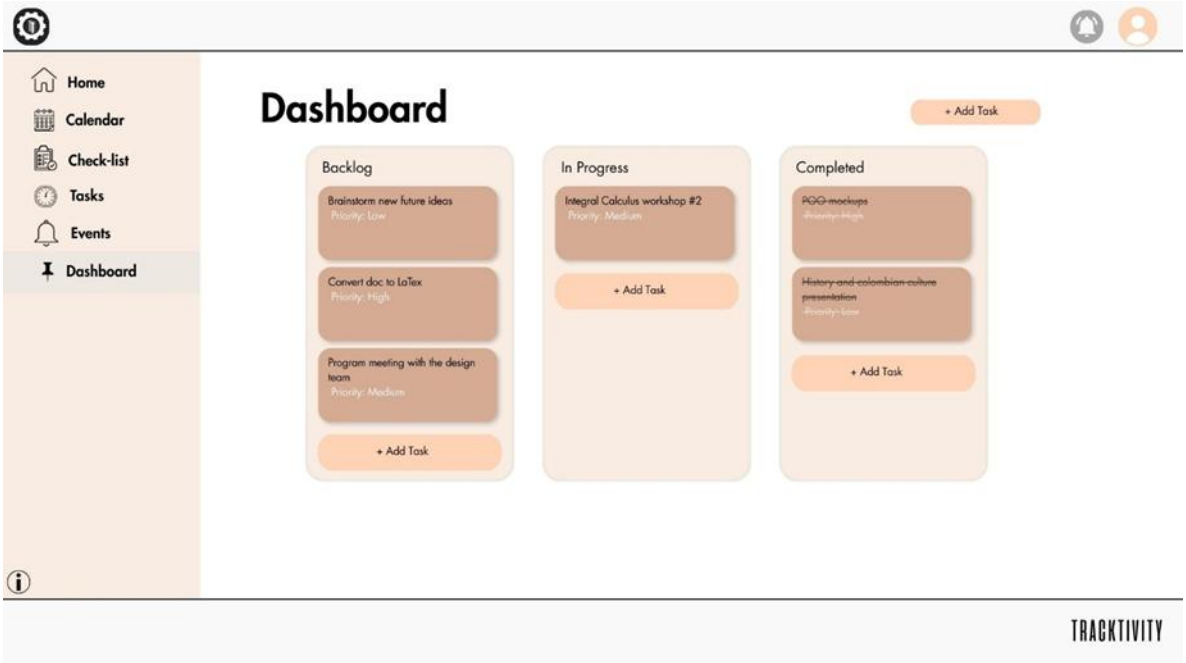
Mockup 4

Check Lists: Displays created tasks with their status and date



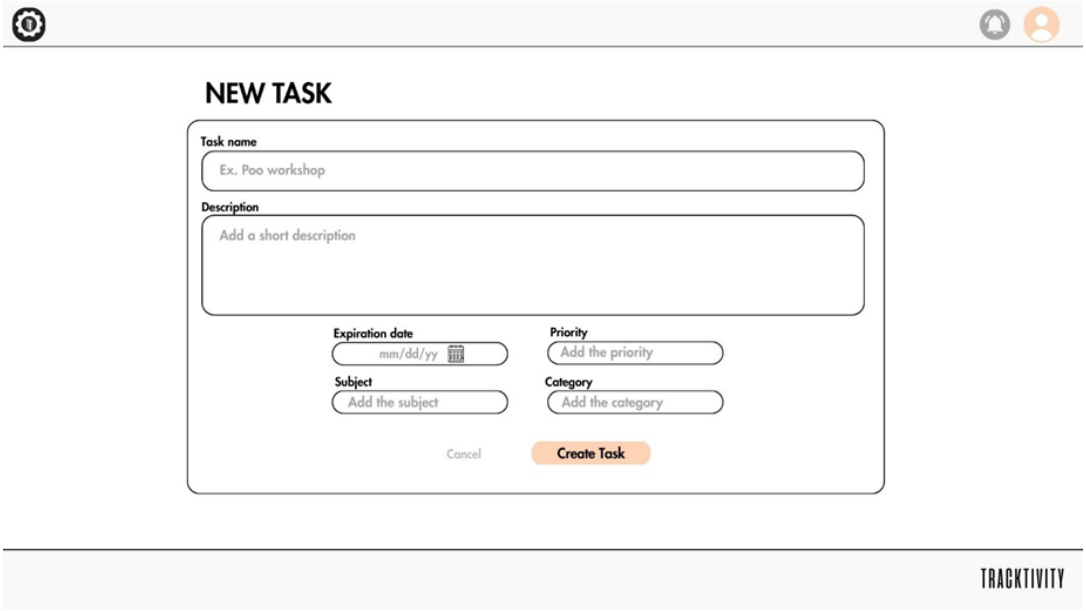
Mockup 5

Events: Display events from the calendar view



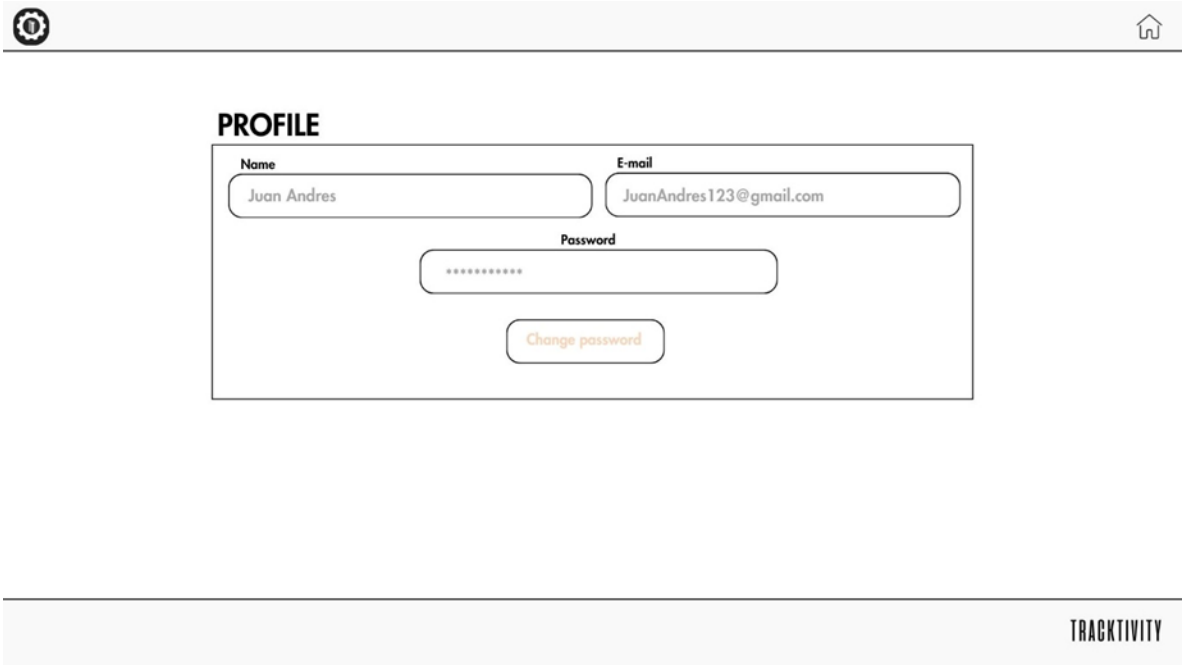
Mockup 6

DashBoard: Allows you to view the status of events from a board view



Mockup 7

Task creation: Allows you to record an event with name, date, description, priority, category and subject



Mockup 8

Profile: allows the user to view their profile information and update it

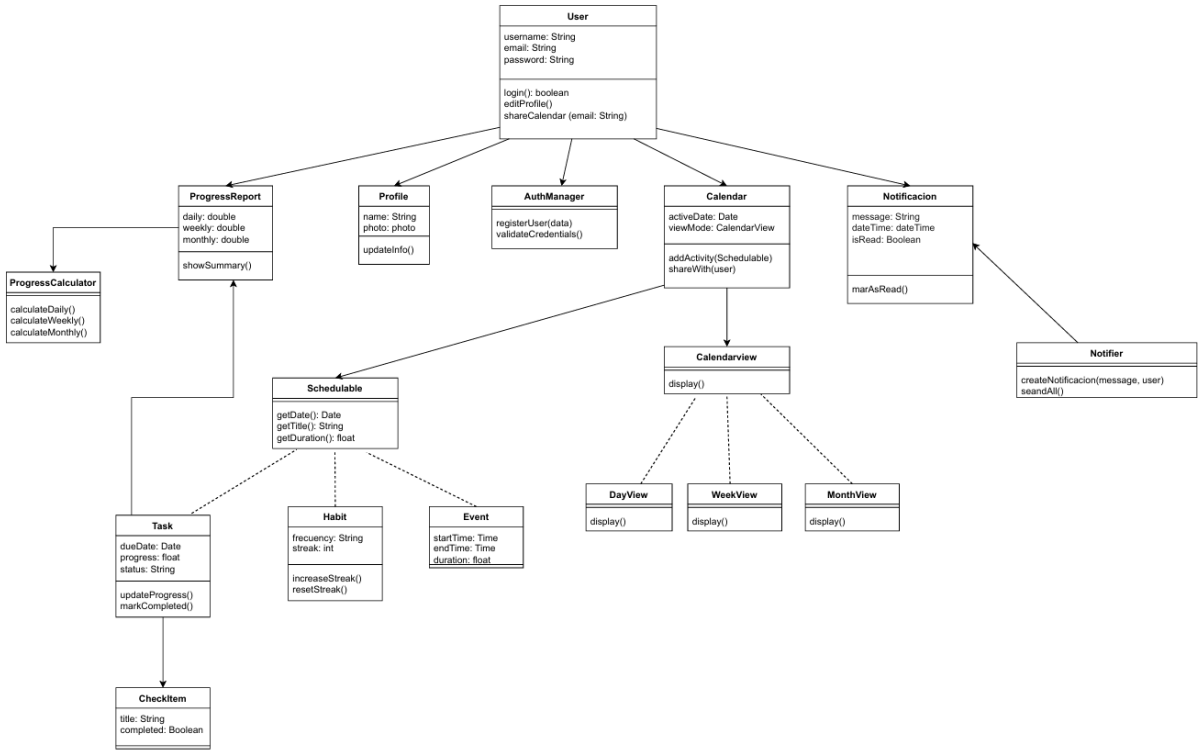
Reflections

During the development of Workshop 1, we applied the fundamental principles of Object-Oriented Programming to design a functional system for managing users, events, and tasks. One of the main challenges was properly structuring the classes and their responsibilities while maintaining consistency between functional and non-functional requirements. We also faced difficulties defining the connection logic between the calendar, tasks, and notifications without losing clarity or modularity in the code. Through this process, we learned the importance of good prior planning, the usefulness of user stories to guide development, and the relevance of documenting code to facilitate future improvements. Overall, this workshop strengthened our skills in design, collaboration, and logical thinking within the context of object-oriented development.

Feedback

The feedback we received helped us identify several key areas for improvement in our project. First, we need to include an introductory context that clearly explains the purpose and scope of the application, both in the document and in the Git repository. Regarding the functional requirements, we were advised to avoid using phrases like “the program should” and instead rewrite them from the user’s perspective (e.g., “A new user can register in the application”). For the non-functional requirements, we must add a brief analysis justifying the numerical values chosen, connecting them to realistic expectations and project goals. In the user stories, each should be presented in separate tables, include an estimated completion date, and the acceptance criteria must be written in third person to reflect the user’s point of view, ensuring they meet the project’s MVP. The CRC Cards need to be reformatted as separate tables and revised to distribute responsibilities more evenly among classes, avoiding overdependence on just two. Lastly, for the mockups, we must add short descriptions for each screen and explain how they relate to the overall project. Finally, we were reminded to include a short reflection that summarizes our focus, the challenges we encountered, and the lessons we learned during the project’s development.

Technical Design



UML Diagram

This is the UML diagram of the project, which represents the relationships between the main classes, as well as the inheritance, composition, and dependency links that structure the system. In this version, new classes such as ProgressCalculator, Habit, and Profile have been added to expand functionality and improve code organization. Additionally, it shows how the classes interact with each other to manage users, tasks, events, notifications, and progress tracking within the application.

Implementation Plan for OOP Concepts

Encapsulation

The application applies the concept of encapsulation by restricting direct access to data through the use of modifier methods. The attributes of all classes will be private to protect the state of each object, and only public methods will be exposed when it is necessary to access or modify information from other classes. Additionally, getters and setters will be used to perform validations before changing the value of an attribute.

Inheritance

The application implements inheritance to avoid overloading some classes and to prevent code duplication.

Base class: Schedulable

This class defines all the common characteristics (Date, Title, Duration) for all schedulable elements (Task, Event, Habit), through abstract methods that handle these values (getDate(), getTitle(), getDuration()).

Derived classes: Event, Task, Habit

These classes inherit the attributes and behaviors from the base class and implement them according to their specific type:

Event implements the date value through a startTime method that defines the start time.

Task implements the date value through a dueDate method that defines the deadline.

Habit adds two attributes — frequency and streak — to the existing ones and includes methods that manage the duration and progress of habit streaks.

Base class: CalendarView

This class defines a display() method that all its derived classes inherit and override based on the required visual implementation.

Derived classes: DayView, WeekView, MonthView

Each class uses this display() method, adapting it to show the calendar by day, week, or month view.

Polymorphism

The design applies polymorphism through the overriding of inherited methods. Each subclass redefines the behavior of certain methods to adapt them to a different purpose.
For example, the display() method inherited by DayView, WeekView, and MonthView from CalendarView keeps the same name but is adapted in each class to show a different type of view.
Other overridden methods include getDate() and getDuration() in Task and Event, which implement the abstract methods of Schedulable with behaviors that differ depending on the type of object.

At this stage of the design, method overloading has not been implemented. The use of polymorphism focuses exclusively on overriding, allowing inherited behaviors to be redefined when necessary.

Directory Structure

The project structure is organized within the main src folder, grouping the classes according to their function and the inheritance relationships established in the design. This

organization improves code readability, enforces separation of responsibilities, and facilitates system scalability.

The auth package contains the classes related to user authentication and management. In this module, the AuthManager class handles user registration and login, while User stores profile information.

The profiles package separates user profile configuration (Profile) from account management (User), applying the principle of separation of responsibilities.

The calendar package includes the classes related to scheduling and activity organization. The abstract class Schedulable serves as the base class for all schedulable elements, while Event, Task, and Habit are derived classes that implement or extend its behavior. The CheckItem class remains in the same package due to its direct relationship with Task.

The views package groups the classes responsible for the calendar's visual presentation. CalendarView functions as an abstract base class, while its derived classes DayView, WeekView, and MonthView override the display() method to present different calendar views depending on the context.

The notifications package manages everything related to alerts and reminders. In this module, the Notification class defines the structure of the message, while Notifier centralizes its creation and delivery to users.

Finally, the progress package gathers the classes that calculate and present performance and progress statistics within the system, ensuring continuous evaluation of user tasks and habits.

Work in progress code

```
App.java
src > app > App.java
package app;

public class App {
    /**
     * Main method that launches the application.
     * In future versions, it will initialize the GUI and controllers.
     */
    public static void main(String[] args) {
        System.out.println("Welcome to TrackTivity!");
    }
}
```

Class App

AuthManager.java

src > auth > AuthManager.java > AuthManager

```java
package auth;

/**
 * Manages user registration and authentication processes.
 */
public class AuthManager {

    /** Constructor of class AuthManager */
    public AuthManager() {}

    /**
     * Will register a new user and store data in the database.
     * @return a new User object.
     */
    public User registerUser(String username, String email, String password) {
        return new User(username, email, password);
    }

    /**
     * Will validate credentials with the authentication system.
     * @return true if the credentials are valid.
     */
    public boolean validateCredentials(User user, String email, String password) {
        return true;
    }
}
```

Class AuthManager

User.java

src > auth > User.java > {} auth

```java
package auth;

/**
 * Represents a user in the system.
 * Manages authentication and personal information.
 */
public class User {
    private String username;
    private String email;
    private String password;

    /** Constructor of class User */
    public User(String username, String email, String password) {
        this.username = username;
        this.email = email;
        this.password = password;
    }

    /**
     * Will verify user credentials in future versions.
     * @return true if credentials are valid.
     */
    public boolean login(String enteredEmail, String enteredPassword) {
        return true;
    }

    /**
     * Will return the user's email stored in the database.
     * @return user email.
     */
    public String getEmail() {
        return email;
    }
}
```

Class User

Checkitem.java

src > calendar > Checkitem.java > Checkitem > markAsCompleted()

```java
package calendar;

/**
 * Represents an item in a checklist used within a task.
 */
public class Checkitem {
    private String title;
    private boolean completed;

    /** Constructor of class CheckItem */
    public Checkitem(String title) {
        this.title = title;
        this.completed = false;
    }

    /**
     * Will mark this checklist item as completed.
     */
    public void markAsCompleted() {}
}
```

Class Checkitem

Event.java

src > calendar > Event.java > {} calendar

```java
package calendar;

// import java.util.Date → Used for handling event date.
import java.util.Date;

/**
 * Represents a calendar event.
 * Inherits from Schedulable and adds start and end times.
 */
public class Event extends Schedulable {
    private String startTime;
    private String endTime;

    /** Constructor of class Event */
    public Event(String title, Date date, String startTime, String endTime) {
        super(title, date);
        this.startTime = startTime;
        this.endTime = endTime;
    }

    /**
     * Will display event details such as title, date, and duration.
     */
    public void showDetails() {}
}
```

Class Event

Habit.java

src > calendar > Habit.java > {} calendar

```java
package calendar;

// import java.util.Date → Used to manage start or repetition date of a habit.
import java.util.Date;

/**
 * Represents a recurring habit maintained by the user.
 * Extends Task to reuse common attributes.
 */
public class Habit extends Task {
    private String frequency;
    private int streak;

    /** Constructor of class Habit */
    public Habit(String title, Date date, String frequency, int streak) {
        super(title, date, status:"Active", progress:0);
        this.frequency = frequency;
        this.streak = streak;
    }

    /**
     * Will reset the habit streak when the routine is broken.
     */
    public void resetStreak() {}

    /**
     * Will display habit information such as frequency and streak count.
     */
    public void showDetails() {}
}
```

Class Habit

Schedulable.java

src > calendar > Schedulable.java > {} calendar

```java
package calendar;

// import java.util.Date → Used for managing basic date values in schedulable elements.
import java.util.Date;

/**
 * Abstract base class for all schedulable calendar elements.
 * Defines shared attributes and methods for tasks, events, and habits.
 */
public abstract class Schedulable {
    protected String title;
    protected Date date;

    /** Constructor of class Schedulable */
    public Schedulable(String title, Date date) {
        this.title = title;
        this.date = date;
    }

    /**
     * Will display detailed information of the schedulable item.
     */
    public abstract void showDetails();
}

```

Class Schedulable

Task.java

src > calendar > Task.java > {} calendar

```java
package calendar;

// import java.util.Date → Used to represent the due date of a task.
import java.util.Date;

/**
 * Represents a task with a deadline and progress.
 */
public class Task extends Schedulable {
    private String status;
    private float progress;

    /** Constructor of class Task */
    public Task(String title, Date date, String status, float progress) {
        super(title, date);
        this.status = status;
        this.progress = progress;
    }

    /**
     * Will update the progress percentage and completion state.
     */
    public void updateProgress(float newProgress) {}

    /**
     * Will display detailed task information.
     */
    public void showDetails() {}
}
```

Class Task

Notification.java

src > notifications > Notification.java > Notification > Notification(String)

```java
package notifications;

// import java.time.LocalDateTime → Used to register the exact date and time a notification is created.
import java.time.LocalDateTime;

/**
 * Represents a system notification sent to the user.
 */
public class Notification {
    private String message;
    private LocalDateTime dateTime;
    private boolean isRead;

    /** Constructor of class Notification */
    public Notification(String message) {
        this.message = message;
        this.dateTime = LocalDateTime.now();
        this.isRead = false;
    }

    /**
     * Will mark the notification as read in future versions.
     */
    public void markAsRead() {}
}
```

Class Notification

Notifier.java

src > notifications > Notifier.java > ...

```java
package notifications;
import auth.User;

/**
 * Handles creation and sending of notifications.
 * Works together with the Notification class.
 */
public class Notifier {

    /** Constructor of class Notifier */
    public Notifier() {}

    /**
     * Will create and send a notification to the specified user.
     */
    public void createNotification(String message, User user) {}
}
```

Class Notifier

Profile.java

src > profiles > Profile.java > {} profiles

```java
package profiles;

/**
 * Represents the user's profile information.
 * Stores personal details and avatar settings.
 */
public class Profile {
    private String name;
    private String photo;

    /** Constructor of class Profile */
    public Profile(String name, String photo) {
        this.name = name;
        this.photo = photo;
    }

    /**
     * Will update profile information such as name and photo.
     */
    public void updateInfo(String newName, String newPhoto) {}
}
```

Class Profile

ProgressCalculator.java

src > progress > ProgressCalculator.java > {} progress

```java
package progress;

/**
 * Calculates user progress based on completed tasks and habits.
 */
public class ProgressCalculator {

    /** Constructor of class ProgressCalculator */
    public ProgressCalculator() {}

    /**
     * Will calculate the user's daily progress percentage.
     * @return daily progress value.
     */
    public double calculateDaily() { return 0.0; }

    /**
     * Will calculate weekly progress from completed items.
     * @return weekly progress value.
     */
    public double calculateWeekly() { return 0.0; }

    /**
     * Will calculate overall monthly progress.
     * @return monthly progress value.
     */
    public double calculateMonthly() { return 0.0; }
}
```

Class ProgressCalculator

ProgressReport.java

src > progress > ProgressReport.java > {} progress

```java
package progress;

/**
 * Displays a summary report of the user's progress.
 */
public class ProgressReport {
    private double daily;
    private double weekly;
    private double monthly;

    /** Constructor of class ProgressReport */
    public ProgressReport(double daily, double weekly, double monthly) {
        this.daily = daily;
        this.weekly = weekly;
        this.monthly = monthly;
    }

    /**
     * Will generate and display a visual progress summary.
     */
    public void showSummary() {}
}

```

Class ProgressReport

CalendarView.java

src > views > CalendarView.java > {} views

```java
package views;

/**
 * Abstract base class for all calendar views (day, week, month).
 * Allows polymorphic display behavior across subclasses.
 */
public abstract class CalendarView {
    protected String viewName;

    /** Constructor of class CalendarView */
    public CalendarView(String viewName) {
        this.viewName = viewName;
    }

    /**
     * Will render the corresponding calendar view on screen.
     */
    public abstract void display();
}

```

Class CalendarView

```
DayView.java ×
src > views > DayView.java > {} views
1   package views;
2
3   /**
4    * Displays the daily view of the calendar.
5    */
6   public class DayView extends CalendarView {
7
8       /** Constructor of class DayView */
9       public DayView() {
10          super(viewName:"Daily View");
11      }
12
13      /**
14       * Will show all scheduled elements for a single day.
15       */
16      public void display() {}
17  }
```

Class DayView

```
MonthView.java ×
src > views > MonthView.java > {} views
1   package views;
2
3   /**
4    * Displays the monthly view of the calendar.
5    */
6   public class MonthView extends CalendarView {
7
8       /** Constructor of class MonthView */
9       public MonthView() {
10          super(viewName:"Monthly View");
11      }
12
13      /**
14       * Will show a monthly overview including all activities.
15       */
16      public void display() {}
17  }
18
```

Class MonthView

```
WeekView.java ×
src > views > WeekView.java > {} views
package views;

/**
 * Displays the weekly view of the calendar.
 */
public class WeekView extends CalendarView {

    /** Constructor of class WeekView */
    public WeekView() {
        super(viewName:"Weekly View");
    }

    /**
     * Will show the current week's events, tasks, and habits.
     */
    public void display() {}
}
```

Class WeekView

Reflections

During Workshop 2, we applied Object-Oriented Programming concepts and UML modeling to design a structured system for managing users, events, and tasks. A key challenge was creating accurate class diagrams and translating them into functional code without losing clarity or modularity. This experience taught us the importance of planning with UML, visualizing system architecture, and keeping clear documentation. Overall, it strengthened our skills in modeling, object-oriented thinking, and problem-solving.