WORKSHOP #4

Johan Esmit Sichacá González - 20242020313

Sergio Andres Diaz Cuervo - 20251020166

John Mario Jimenez Becerra - 20251020047

Professor: Carlos Andres Sierra Virguez

Universidad Distrital Francisco Jose de Caldas

Subject: OOP

WORKSHOP #4

Revisiting Layers and Design

Upon reviewing our class diagrams and previous design documentation, we confirmed that they aligned well with the final layered approach. Only minor adjustments were required to clarify the responsibilities and slightly reduce the coupling, while the overall strength of the original design remained intact.

Java FX based GUI Prototype

Based on the main login and registration mockups, we created the corresponding graphical interface prototypes using the JavaFX framework. These prototypes include the buttons that, in the final version, will redirect to the login and registration interfaces. In the final application, all interfaces will be interconnected.

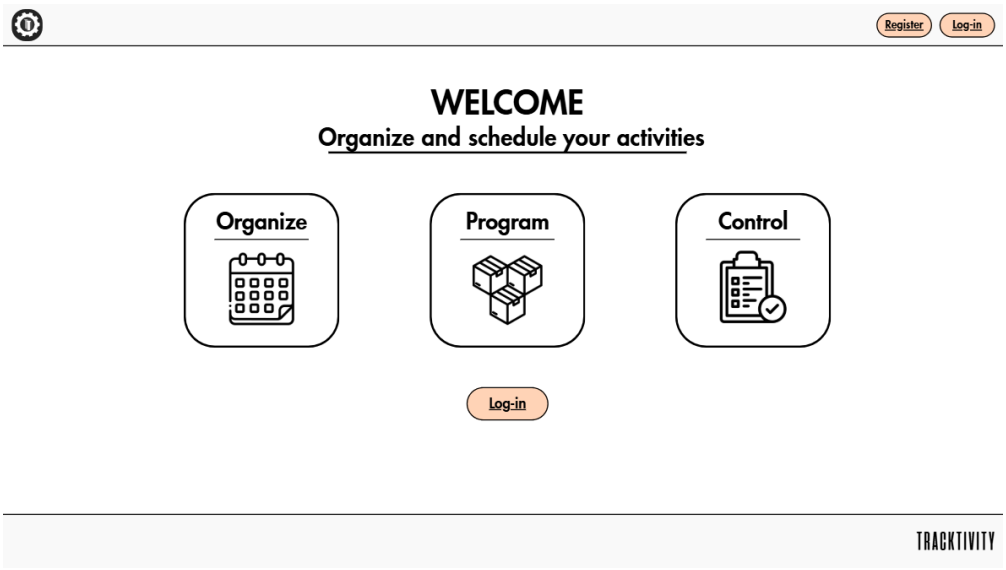


Figure 1

Main Mockup

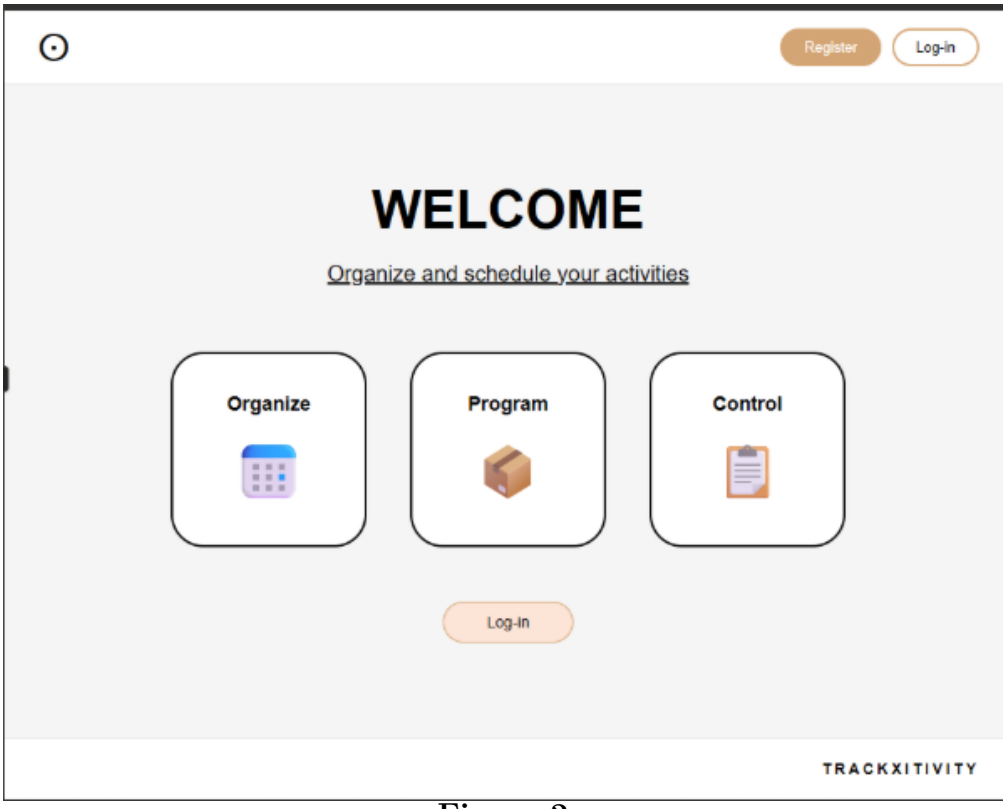


Figure 2

Main GUI

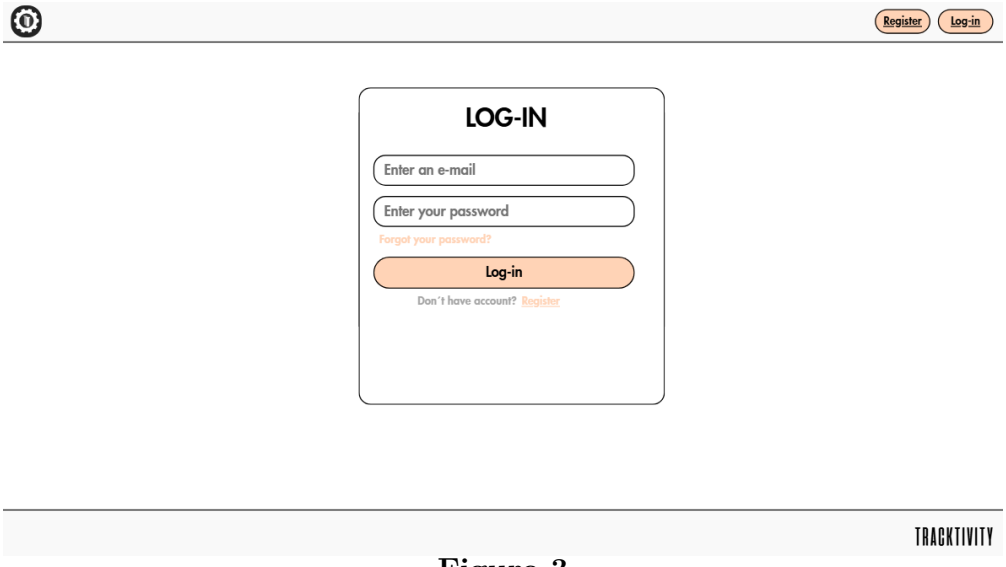


Figure 3

Login Mockup

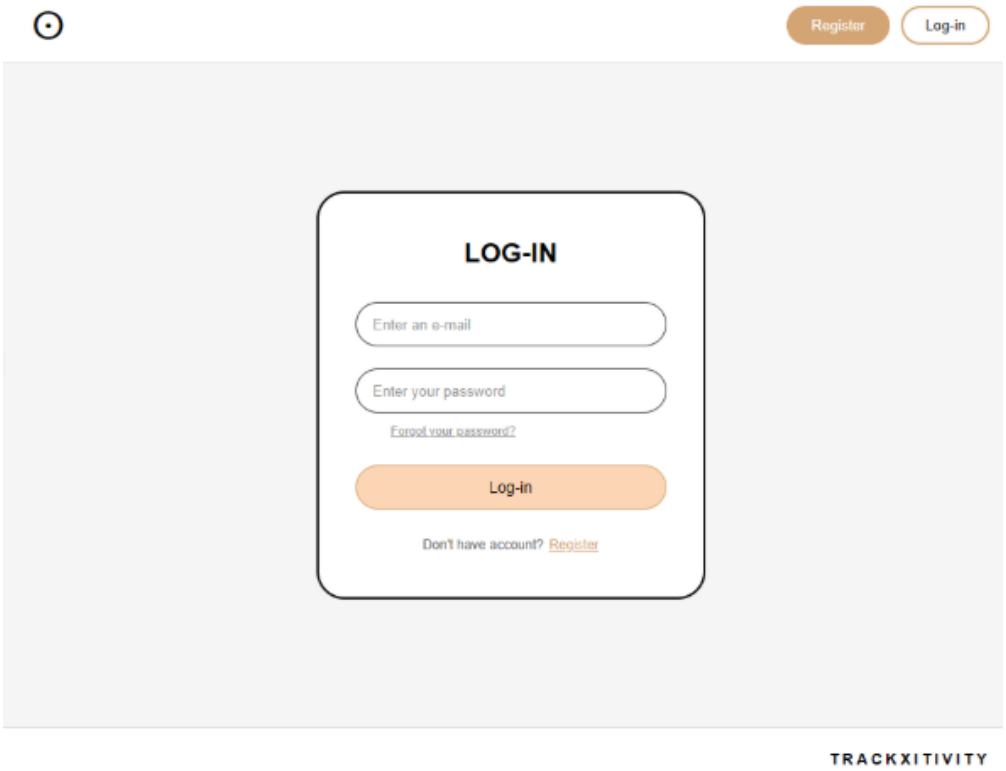


Figure 4

Login GUI

Figure 5

Register Mockup

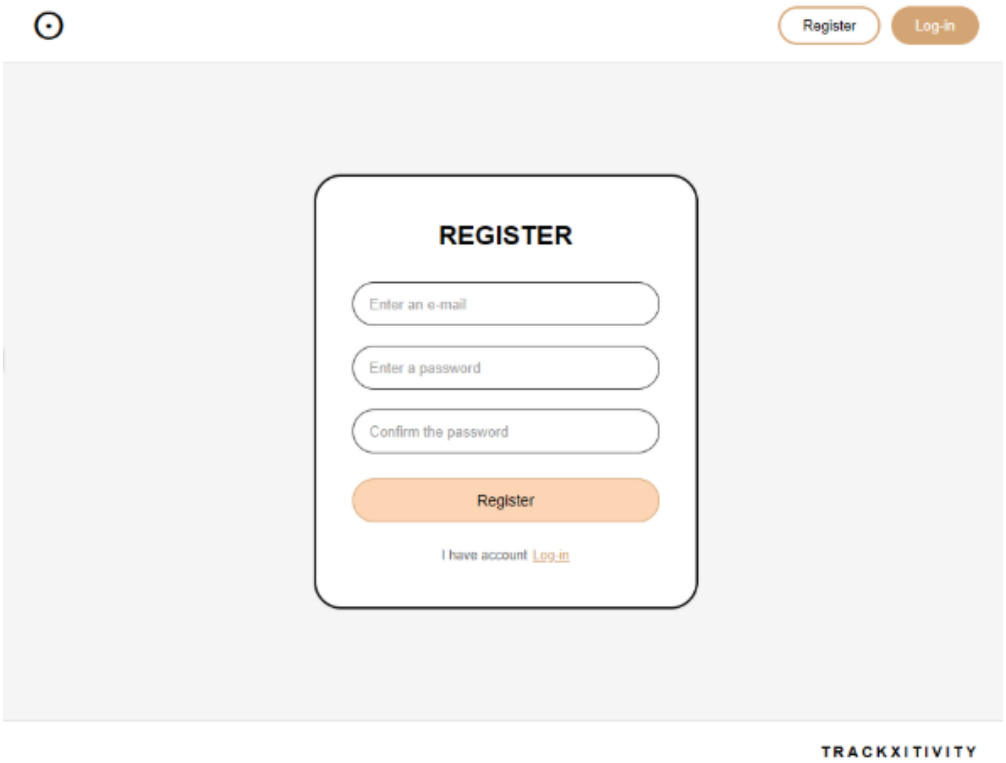


Figure 6

Register GUI

File Storage

To implement data saving and retrieval(File storage) in the User class, a text file was created to store each user's information (name, password, and email). This allows the application to detect whether a user already exists based solely on their name. Additionally, it ensures that only the Profile class can update this information.

```java
public class User {

    /**
     * Registers a new user and saves it to the file.
     */
    void register(Scanner sc, List<User> users) throws Exception {
        System.out.print(s: "Name: ");
        String n = sc.nextLine();

        for (User u : users)
            if (u.name.equals(n)) { System.out.println(x: "User already exists."); return; }

        System.out.print(s: "Password: ");
        String p = sc.nextLine();

        System.out.print(s: "Email: ");
        String e = sc.nextLine();

        User newUser = new User();
        newUser.name = n;
        newUser.password = p;
        newUser.email = e;

        users.add(newUser);
        saveUsers(users);
        System.out.println(x: "Registered successfully.");
    }

    /**
     * Attempts to log the user in by checking credentials.
     */
    void login(Scanner sc, List<User> users) {
        System.out.print(s: "Name: ");
        String n = sc.nextLine();

        System.out.print(s: "Password: ");
        String p = sc.nextLine();

        for (User u : users)
            if (u.name.equals(n) && u.password.equals(p)) {
                System.out.println(x: "Login successful.");
                return;
            }

        System.out.println(x: "Incorrect name or password.");
    }
```

Figure 7

Method for saving information in the file

Documentation and Artifact Submission

As can be seen in the images, there is a significant reduction in the total number of classes in the diagram, while maintaining what had been learned about SOLID concepts. In addition, the App class is added, which is the main class and represents the TrackTivity application. This led us to change certain relationships between other classes, such as the

User class, which had the most relationships with the various classes. Its responsibilities now become part of the main class. Responsibilities are added to certain classes, such as the Notification class, to which a function is added so that the user can activate or deactivate these notifications. You can also see how the arrows that mark the relationships between classes are well marked, and the organization of the classes goes from being hierarchical to decentralized, resulting in a cleaner and better organized technical design.

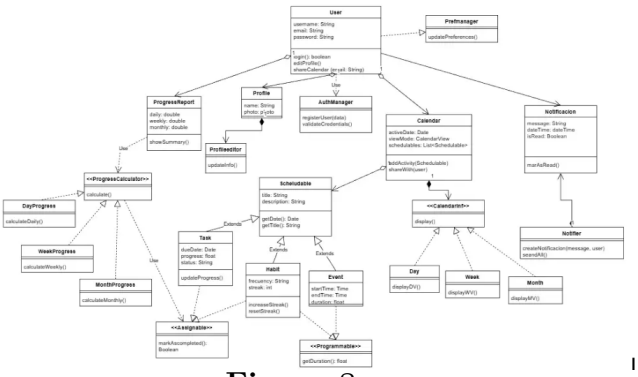


Figure 8

UML Workshop #3

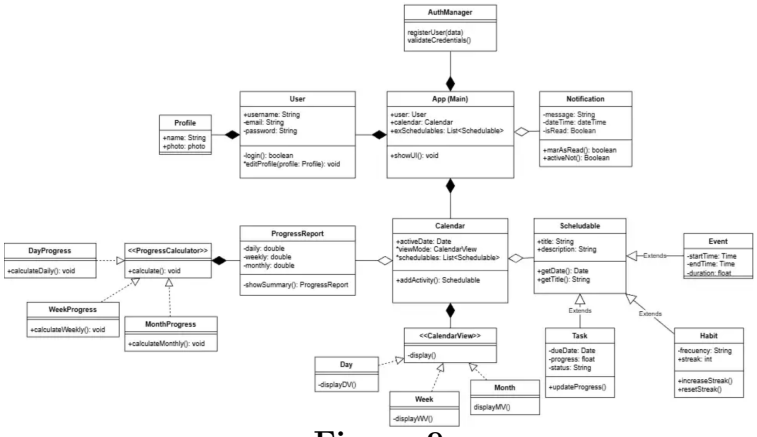
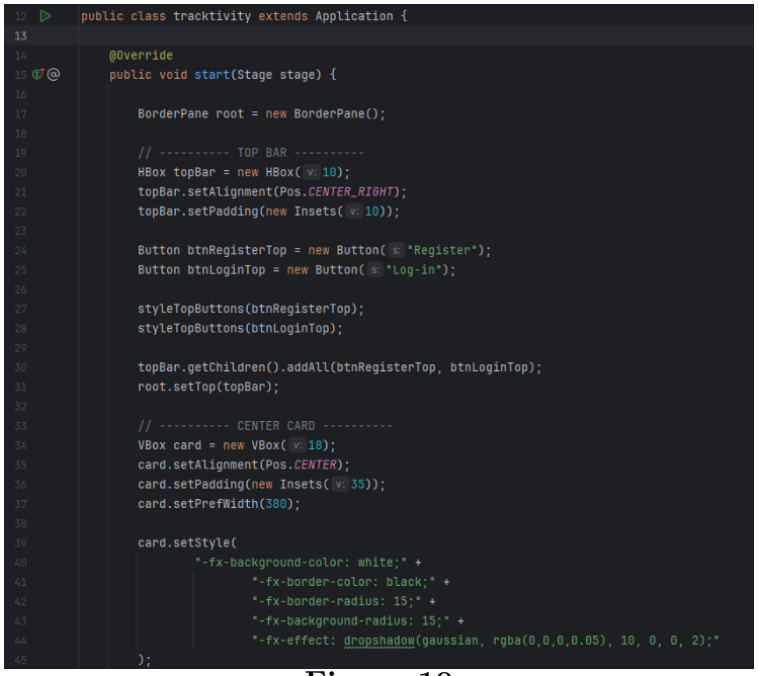


Figure 9

UML Update

Presented below are excerpts of each graphical interface, as the complete code is extensive:



```
public class tracktivity extends Application {

    @Override
    public void start(Stage stage) {

        BorderPane root = new BorderPane();

        // ---------- TOP BAR ----------
        HBox topBar = new HBox( v: 10);
        topBar.setAlignment(Pos.CENTER_RIGHT);
        topBar.setPadding(new Insets( v: 10));

        Button btnRegisterTop = new Button( s: "Register");
        Button btnLoginTop = new Button( s: "Log-in");

        styleTopButtons(btnRegisterTop);
        styleTopButtons(btnLoginTop);

        topBar.getChildren().addAll(btnRegisterTop, btnLoginTop);
        root.setTop(topBar);

        // ---------- CENTER CARD ----------
        VBox card = new VBox( v: 18);
        card.setAlignment(Pos.CENTER);
        card.setPadding(new Insets( v: 35));
        card.setPrefWidth(380);

        card.setStyle(
                "-fx-background-color: white;" +
                        "-fx-border-color: black;" +
                        "-fx-border-radius: 15;" +
                        "-fx-background-radius: 15;" +
                        "-fx-effect: dropshadow(gaussian, rgba(0,0,0,0.05), 10, 0, 0, 2);"
        );
```

Figure 10

Main GUI Code

Figure 11

Login GUI Code

Brief Reflection

During Workshop #4, we primarily focused on the graphical user interface (UI) for the application, as well as making corrections to the previous submission, mostly reflected in the new class diagram. Each section retained previously learned concepts. During the project's development, using the "JavaFX" extension presented a significant challenge. Having no prior experience with this topic, it was a completely new and challenging task.Despite this, we learned how to manage this extension using tools such as Intellij and various tutorials, resulting in the main interfaces that the application will have. It also prompted us to reorganize the UML diagram, taking into account the Pokemon example provided by the professor. We learned how to mark the relationships between classes and decide the function that each one performs within the app, as well as whether it was

marked as an interface, abstract class, parent class, child class, etc. This greatly simplified what had been done in the previous workshop and, of course, applied the SOLID principles that had been proposed previously. In general, we learned the basics of creating a graphical interface for an application using JavaFX, strengthening our problem-solving skills and use of new tools. We also reinforced what we had learned previously, giving us the opportunity to better organize our ideas for the application.