HABITRA

TAKE CONTROL OF YOUR HABITS

Fabian Cabrera
Juan Avila

WHAT IS HABITRA?

Habitra is a project that aims to make a Habit Tracker to help accomplish goals, keep tabs on all of your good habits, and help getting rid of bad habits, all of this while making it fun, resembling a kind of videogame.The app is primarily intended for mobile devices.

OUR PURPOSE ?

Habitra aims to make an interactive Habit Tracker that

-Keep the habits in check

-Rewards completing said habits

-fulfill your tasks

The app keeps interactive by the use of this tools

DEADLINE : It allows to put a deadline to the tasks

D.LINE : DD/MM/YYYY

DIFFICULTY : It allows to put a difficulty to the habits and tasks

DIFF : EASY MEDIUM HARD

EXPERIENCE : It shows you the experience gained by complete tasks and habits.

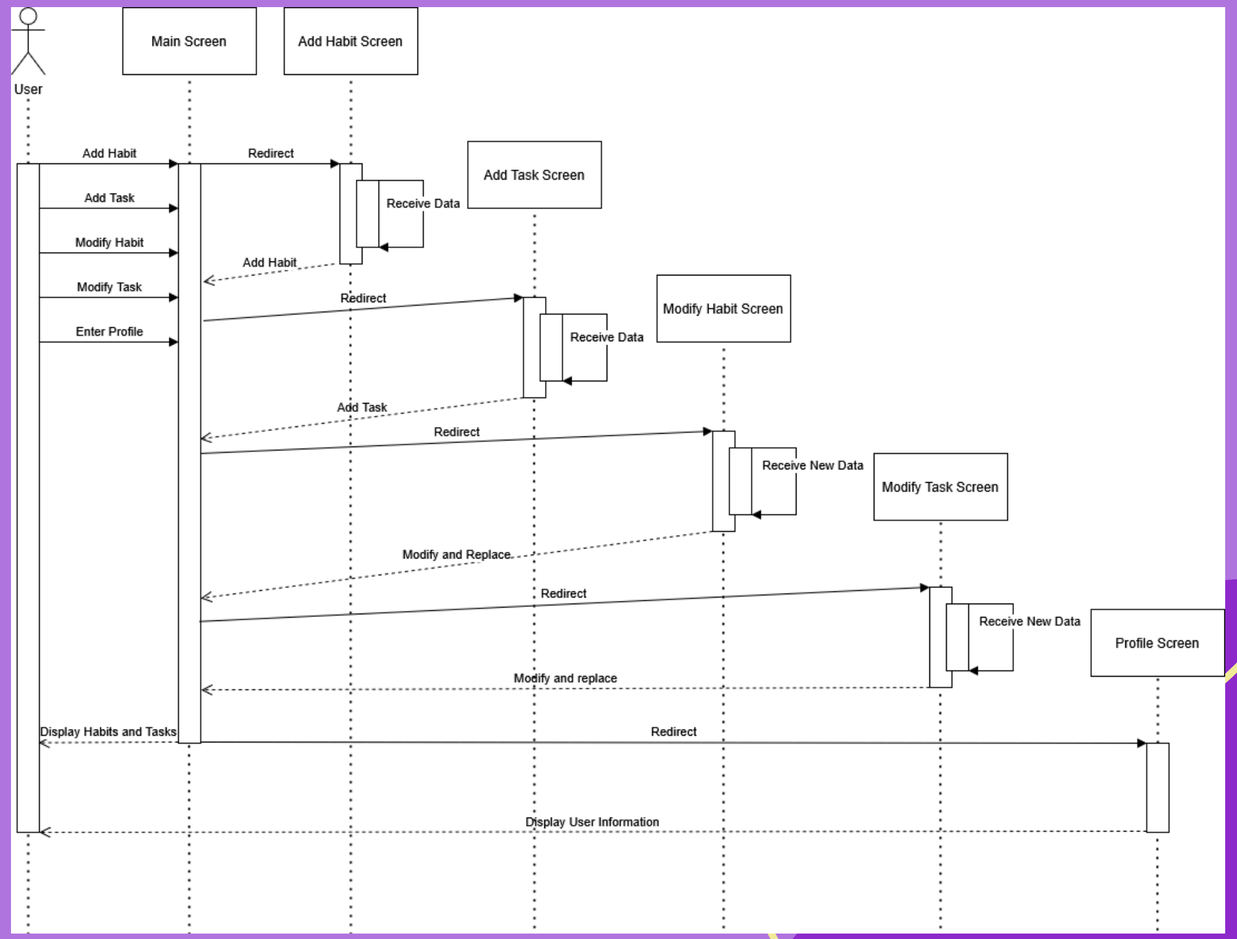
By the using of crc cards and the uml diagrams we could guide to create the classes and the mockups

MAKING USE

OF PRINCIPLES LIKE:

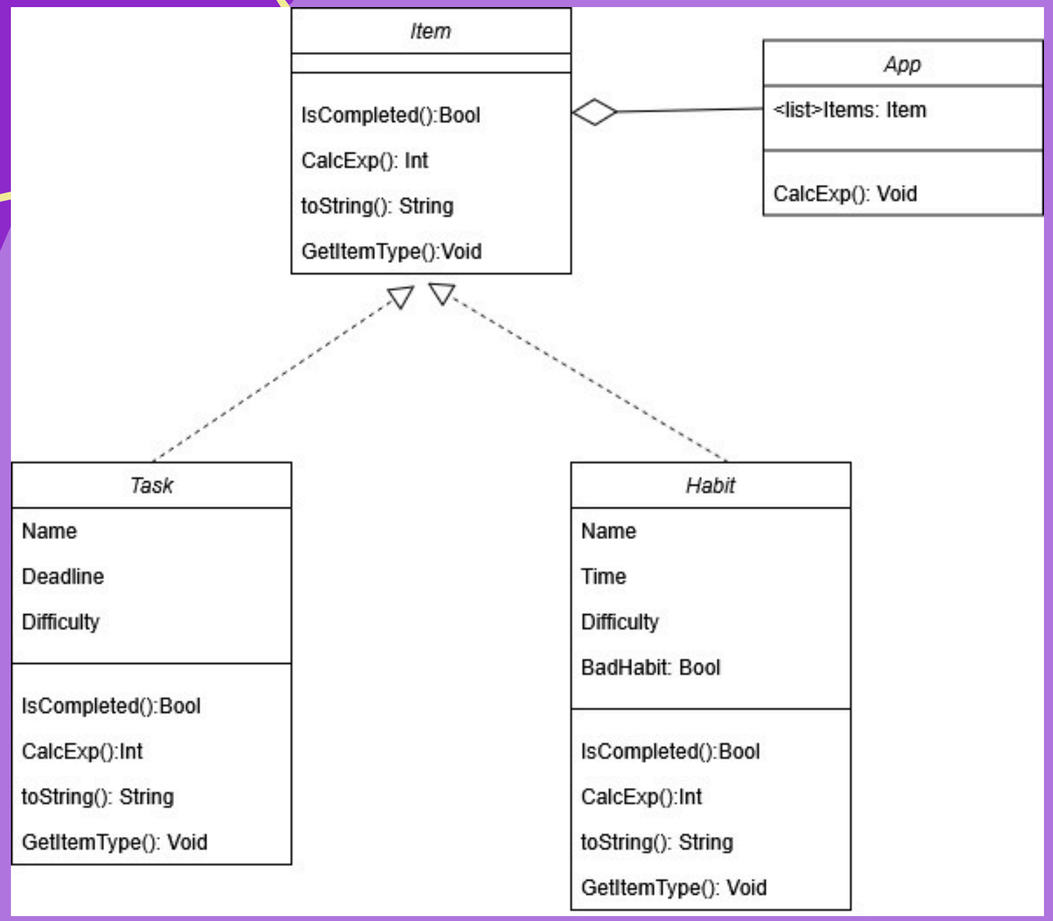
-INHERITANCE

-ABSTRACTION



Sequence diagram showing all the operation of the app

CLASS DIAGRAM



CRC CARDS

Task

Responsibilities:

- Be completed
- Give experience when is completed
- Be modified
- Be expired

Colaborators:

- The user
- Habit Class

Habit

Responsibilities:

- Be completed
- Give experience when is completed
- Be modified
- Be updated

Colaborators:

- The user
- Task Class

OOP PRINCIPLES

In this project was useful the implementation of some of the OOP principles by help to structure all the code splitting it in classes , each class represents an object and thsi objects interact correctly thanks to some principles.

MOCKUPS

MAIN SCREEN OF HABITRA

NEW HABIT SCREEN

INHERITANCE

It was utilized to promote code reuse and logical class hierarchy.

-Item Interface

```java
/**This is an interface for Tasks and habits */

public interface Item{
    public abstract Boolean IsCompleted();
    public abstract void CalcExp();
    public abstract String toString(); //Override of toString Java method
    public abstract String getName();
    public abstract Integer getExp();
}
```

-Habit class that implements from item

```java
/**This is a class for Habits */
public class Habit implements Item{
    private String habName;
    private int time;
    private int exp;
    private int diff;
    private Boolean status;
    private Boolean isBad;

    /**Constructor for habits
     * @param name: The name of the habit
     * @param time: Periodicy of the habit
     * @param difficulty: The difficulty of the habit
     * @param status: If the habit is completed or not
     * @param bad: Wheter the habit is flagged as bad or not
     */
    public Habit(String name, int time, int difficulty, boolean status, Boolean bad){
        this.habName= name;
        this.time= time;
        this.diff= difficulty;
        this.status= status;
        this.isBad= bad;
    }
```

POLYMORPHISM

Polymorphism will be mainly achieved by overriding, since child classes
and classes that implements from an interface will implement some methods and attributes differently

-Implemented methods

```
/**Calculates experience */
    public void CalcExp(){
        this.exp= 100*(diff);
```

OVERRIDE!

```
/**Calculates experience <p>
     * Checks if the habit is bad or not in order to add or substract experience
     */
    public void CalcExp(){
        if(getBad().equals(false))
            this.exp= 10*(diff*time);
        else
            this.exp= -(10*(diff*time));
    }
```

-Abstract method

```
public abstract void CalcExp();
```

ABSTRACTION

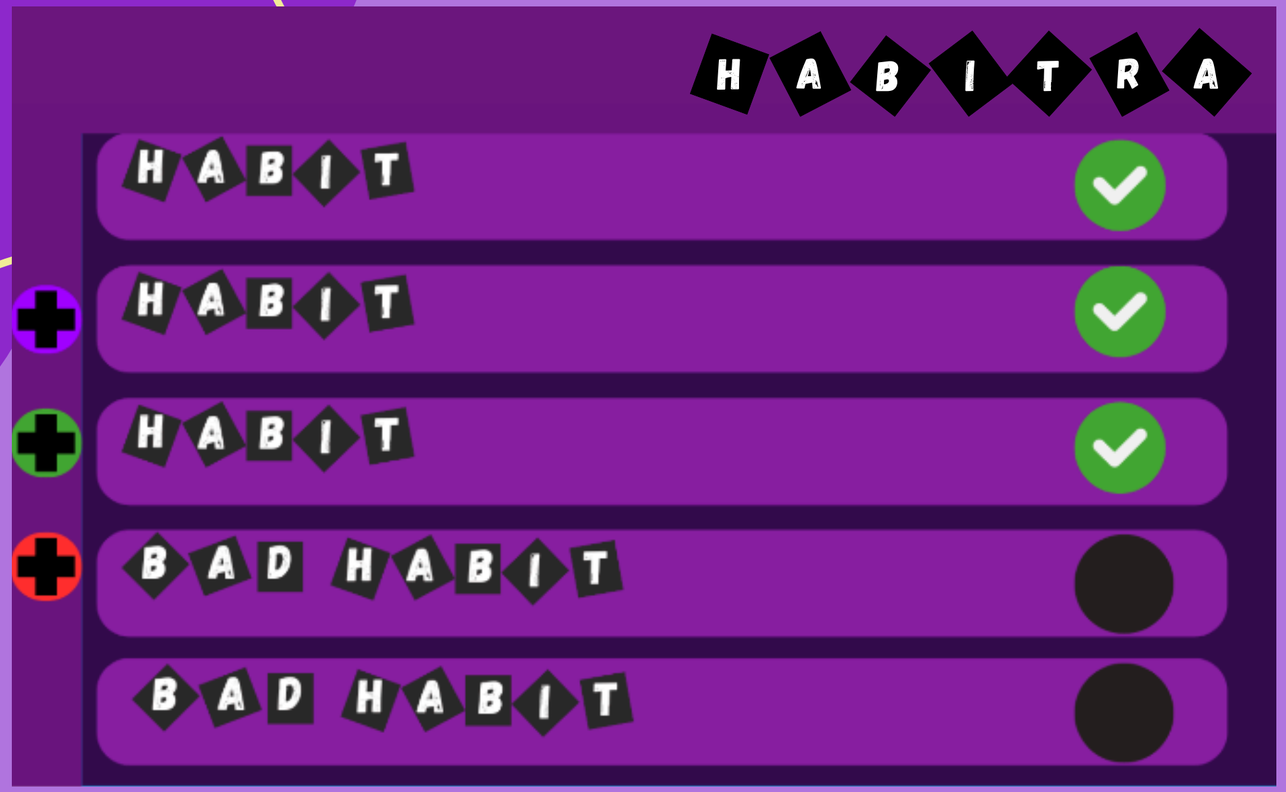
It was enforced through the use of abstract classes and interfaces, defining general behaviors while leaving specific details to be implemented by subclasses

-Interface with abstract methods

```java
/**This is an interface for Tasks and habits */
public interface Item{
    public abstract Boolean IsCompleted();
    public abstract void CalcExp();
    public abstract String toString(); //Override of toString Java method
    public abstract String getName();
    public abstract Integer getExp();
}
```

MAIN SCREEN

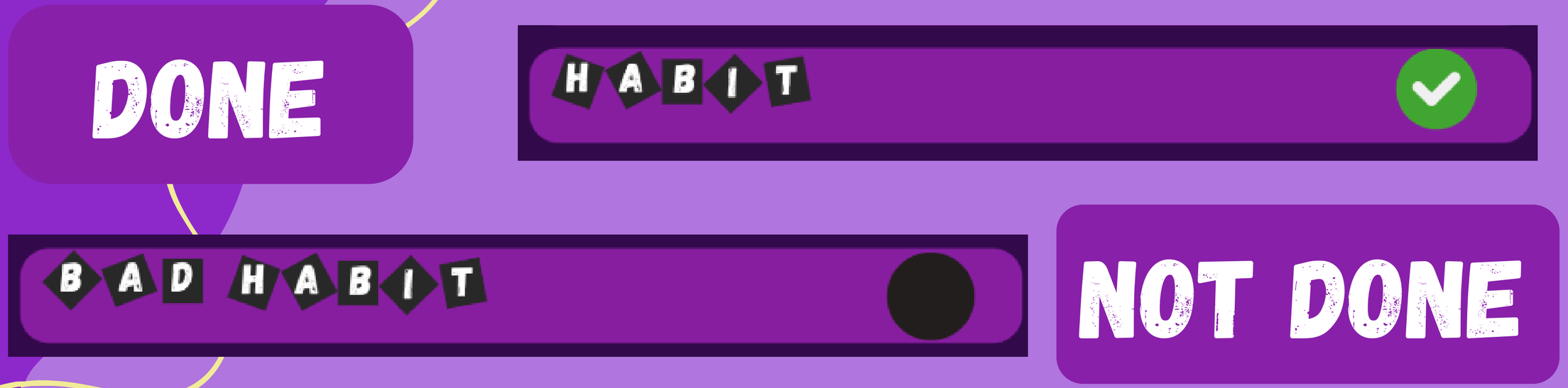
MAIN SCREEN



This button send you to another module in wich you can add a new habit.

This button send you to another module in which you can add a new bad habit

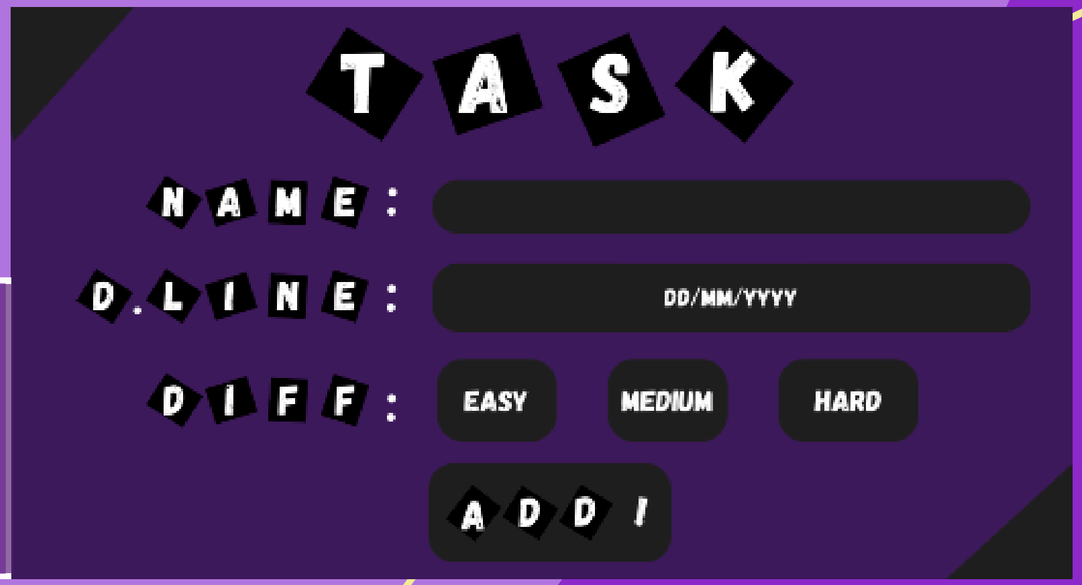
This button send you to another module in which you can add a task .



In this screen is shown the habits , bad habits and tasks that the user have saved .And they can be checked if the user has already made the task or habit.

ADD HABITS AND TASKS

ADD HABITS AND BAD HABITS



ADD TASKS

ADD HABITS AND TASKS

After using one of the three plus buttons on the main screen they will lead you to one of these screens

ADD HABITS AND BAD HABITS

ADD TASKS

ADD HABITS AND TASKS

These screens are similar , they have the space where put the name of the habit/task and the difficulty. The difference between these two is the time that will be confirmed (daily,weekly,monthly) in the habits and bad habits and the deadline in the tasks

HABITS

TASKS

TIME: DAILY WEEKLY MONTHLY

D.LINE:

CHANGE HABITS AND TASKS

CHANGE

NAME : HABIT NAME*

TIME : DAILY | WEEKLY | MONTHLY

DIFF : EASY | MEDIUM | HARD

MODI

CHANGE HABITS AND BAD HABITS

CHANGE

NAME : TASK NAME*

D.LINE : DD/MM/YYYY

DIFF : EASY | MEDIUM | HARD

MODI

CHANGE TASKS

CHANGE HABITS AND TASKS

This two screens are practically the same that add habits and add tasks .The difference is that this modifies an habit already created and instead of add button there is mod button.

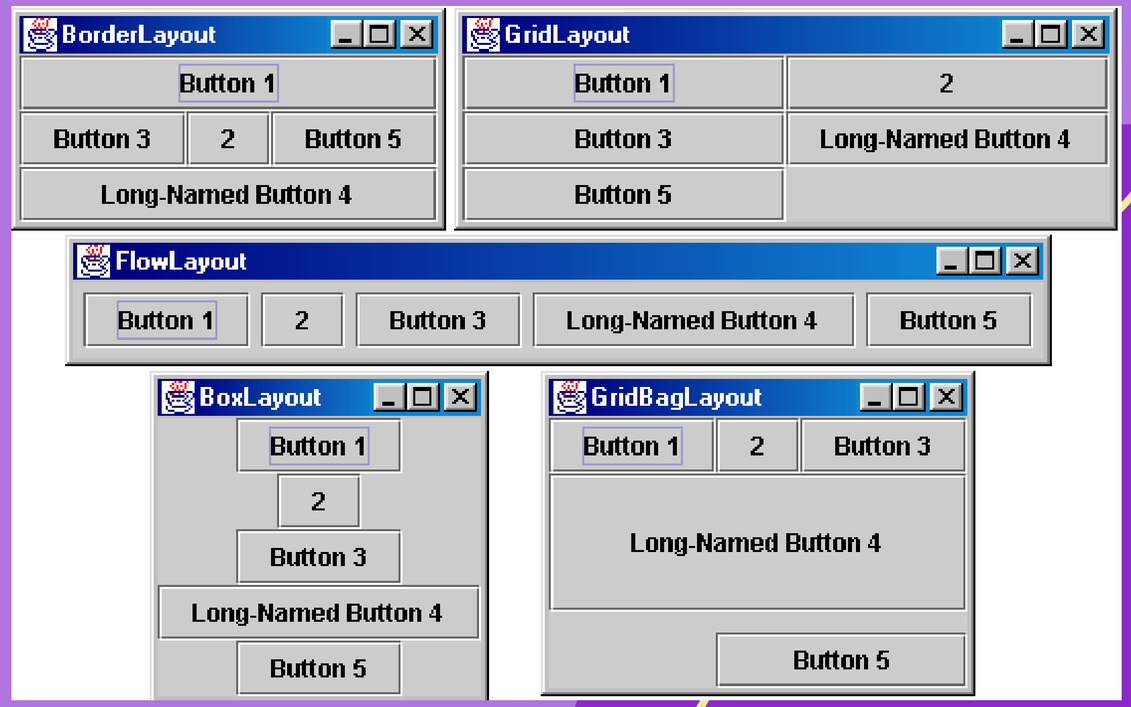
GUI

The GUI (graphical user interface) is one of the most important parts because it is what the user sees and interacts with.

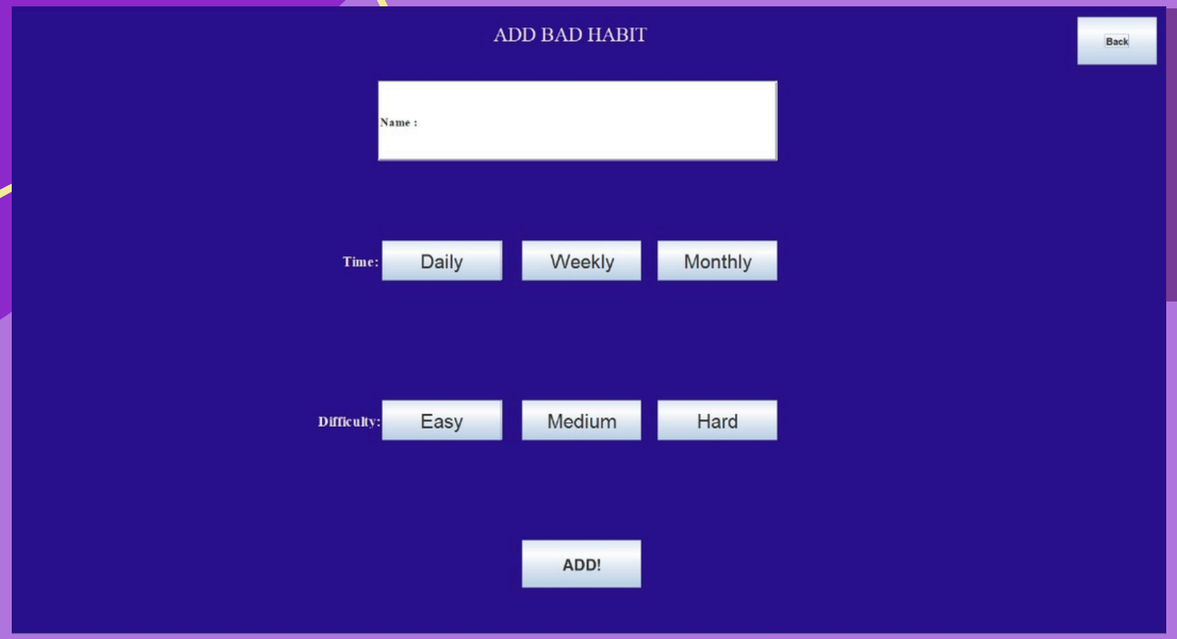
In our case the tool we use to make this was Java swing supporting the MVC by separate the different functions when it comes to making an app

GUI

Java swing helps to create a simple but functional interface by the using of its own tools like JButtons JFrames JPanels and some other elements that ease the creation of what we expected with mockups.



GUI



ADD BAD HABITS

ADD HABITS

RESULTS

- Fully implemented core logic in Java, using OOP and SOLID principles.
- Built a functional and user-friendly graphical interface based on initial mockups.
- All user stories were fulfilled, ensuring alignment between design and implementation.
- Achieved a modular and extensible architecture, ready for future features like data persistence and multi-user support.

THANKS!

Clear, J. (2018), Atomic Habits: An Easy and Proven Way to Build Good Habits and Break Bad Ones, Avery.

Habitica (2024), ‘Habitica: Gamify your life’, https://habitica.com. Available at: https://habitica.com.

Renfree, I., H. D. M. P. S. K. and Cox, A. (2016), Don’t kick the habit: The role of dependency in habit formation apps, in ‘Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems’, ACM, San Jose, CA, USA, pp. 3222–3233.

https://users.dcc.uchile.cl/~lmateu/CC60H/Trabajos/edavis/swing.html