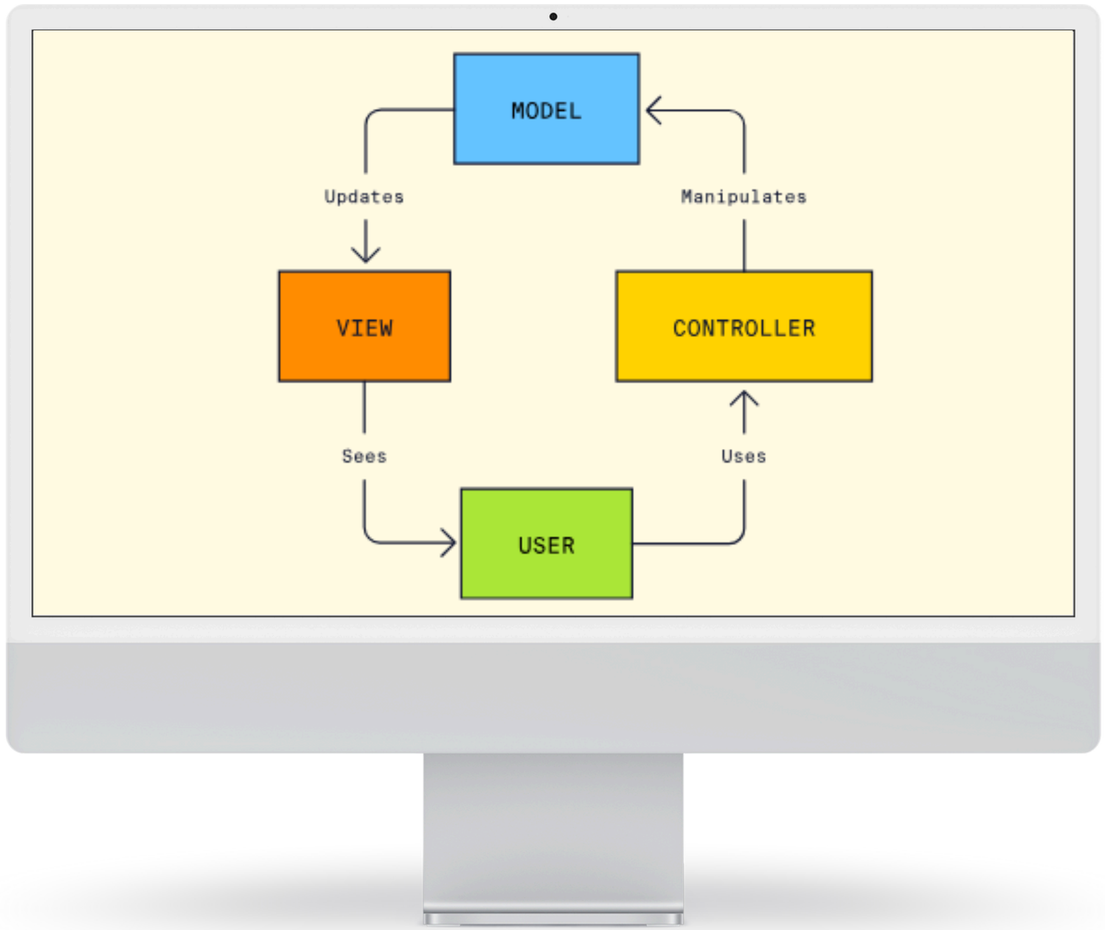
YEAR
2024-25

Model View Controller

ULL



PROGRAMMING INTERACTIVE APPLICATIONS

INTRODUCTION

Our team

Guillermo Silva González

guillermo.silva.26@ull.edu.es

Himar Edhey Hernández Alonso

edhey.alonso.34@ull.edu.es

Samuel Rodríguez Cuesta

samuel.rodriguez.44@ull.edu.es

WELCOME!

Index

01

Introduction

Explanation about de MVC pattern

02

Shopping List

Implementing a Shopping List with MVC

03

Weather App

Implementing a Weather App with MVC, following SOLID Principles

04

Pai Cocktails

Implementinga Cocktail App using reusable components

INTRODUCTION

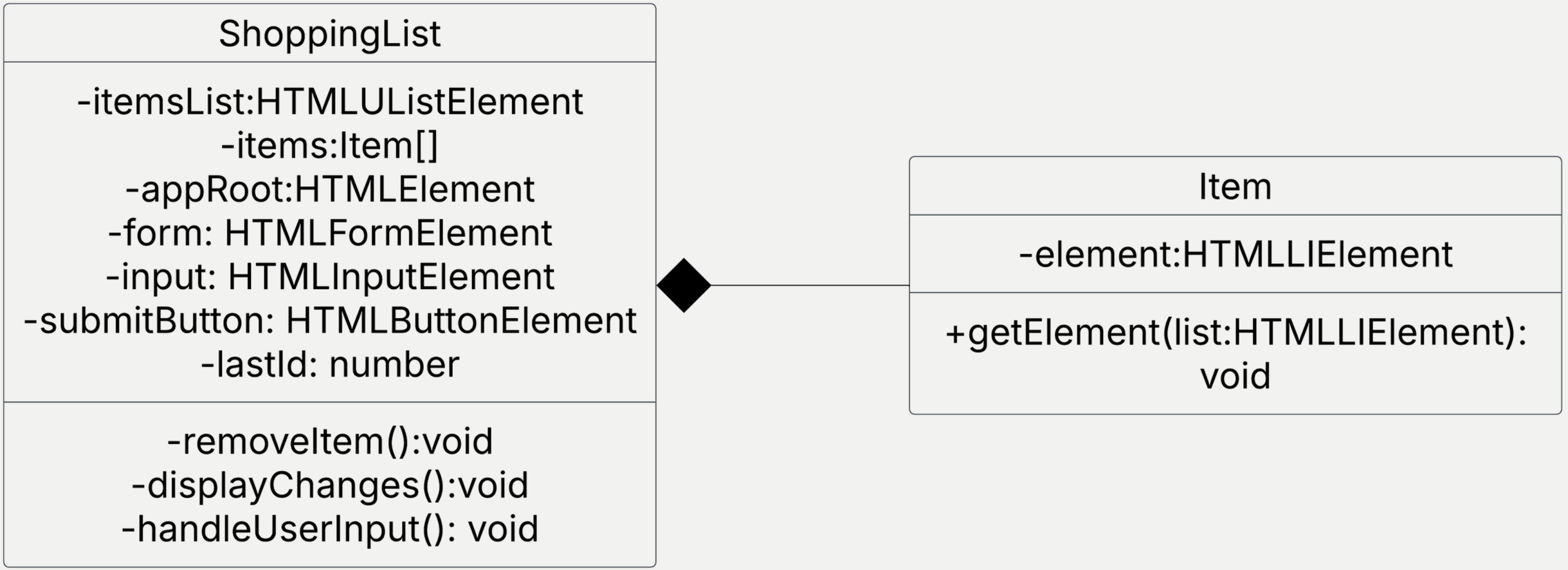
Shopping List

Delete

BUY NOW

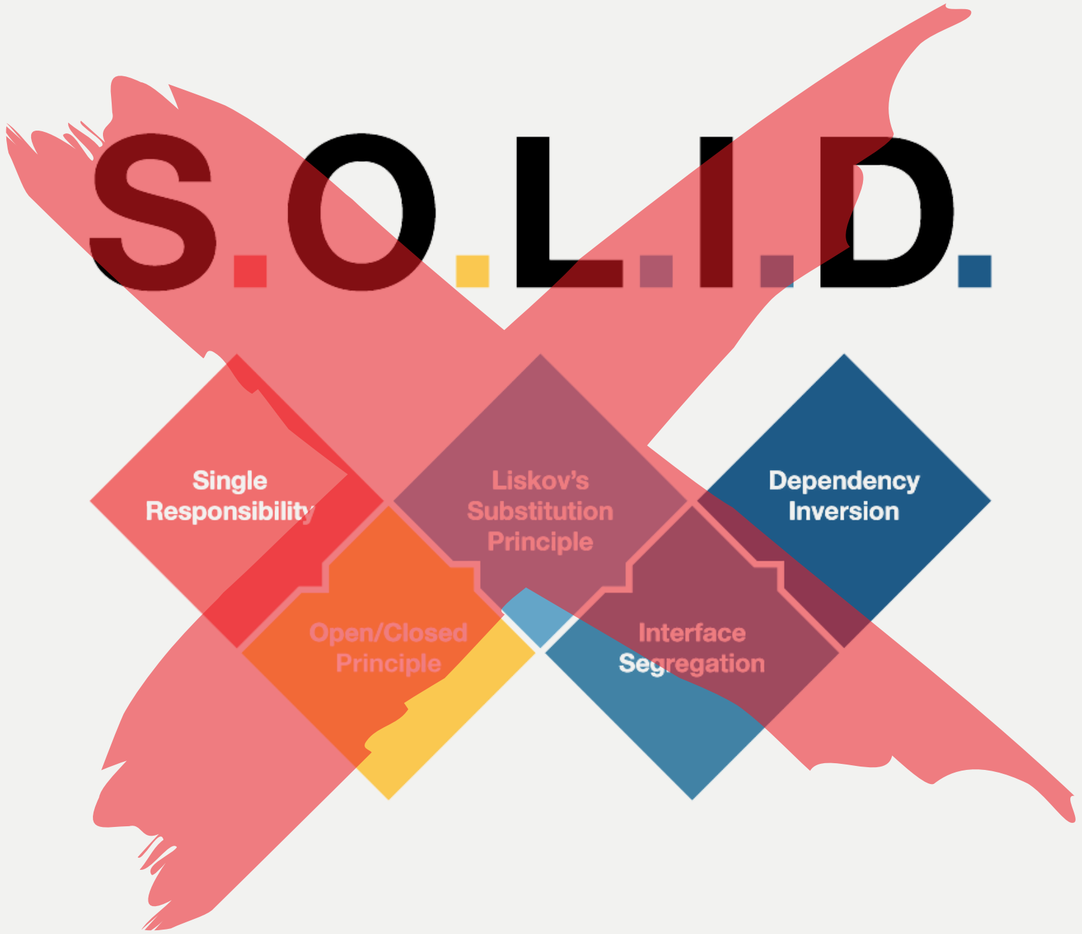
INTRODUCTION

Shopping List



INTRODUCTION

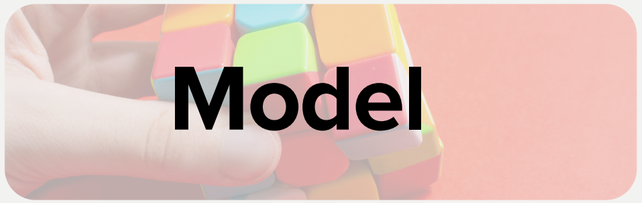
Shopping List



INTRODUCTION

¿What is the Model View Controller?

The Model-View-Controller (MVC) is a software design pattern that separates an application into three interconnected components.



INTRODUCTION

History

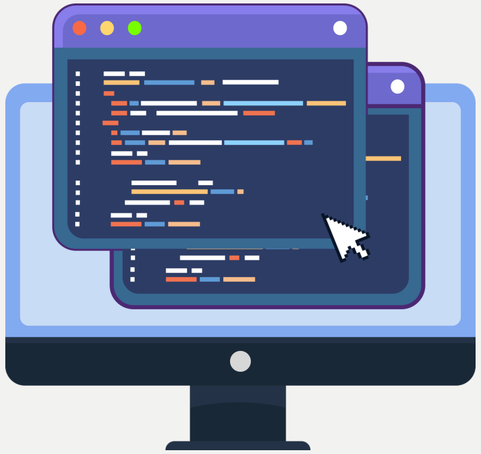
1970 - Smalltalk

Trygve Reenskaug

INTRODUCTION

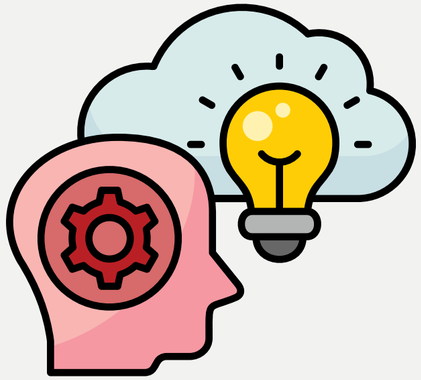
Model

Defines data, logic, and rules.



Internal Behavior

Data Structure



Internal Logic

Business Rules

View

INTRODUCTION

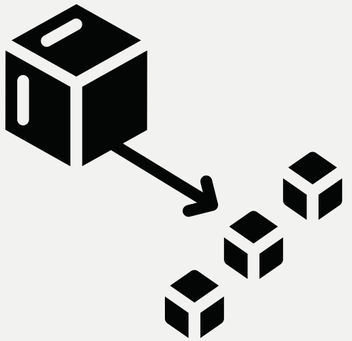
Presents data to the user.

Presentation

Displays the data provided by the Model in a user-friendly format.

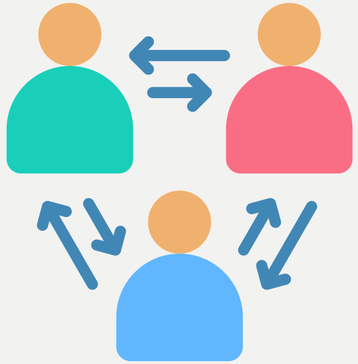
User Interaction

Captures user inputs like clicks, text entries, or selections.



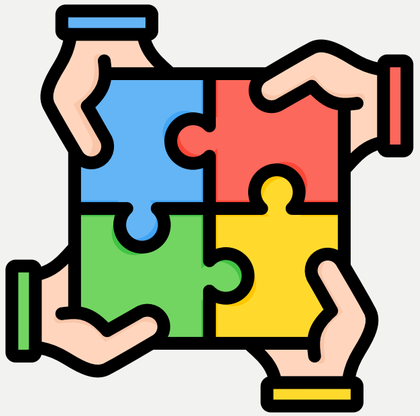
Decoupling

Focuses on presentation without implementing logic or managing data.

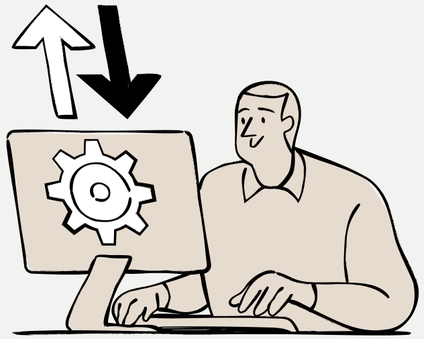


User Interaction Handler

Mediator



Logic Executor



View Updater

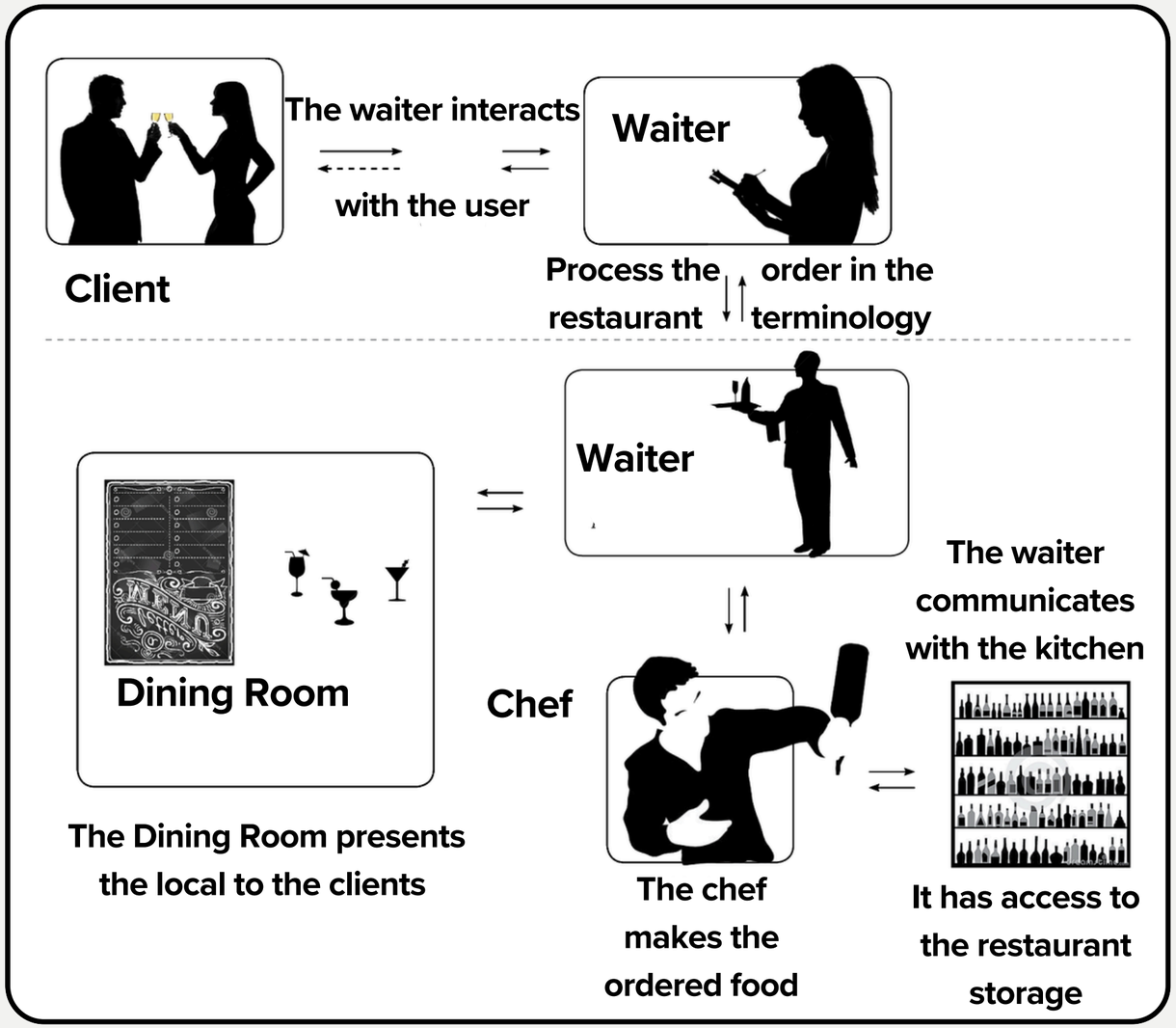
INTRODUCTION

Controller

Handles user input dynamically linking model and view.

INTRODUCTION

Analogy



01

Client (User)

Represents the user interacting with the application.

02

Dining Room (View)

It is the environment that displays information clearly.

03

Waiter (Controller)

Takes customer orders and communicates with the kitchen.

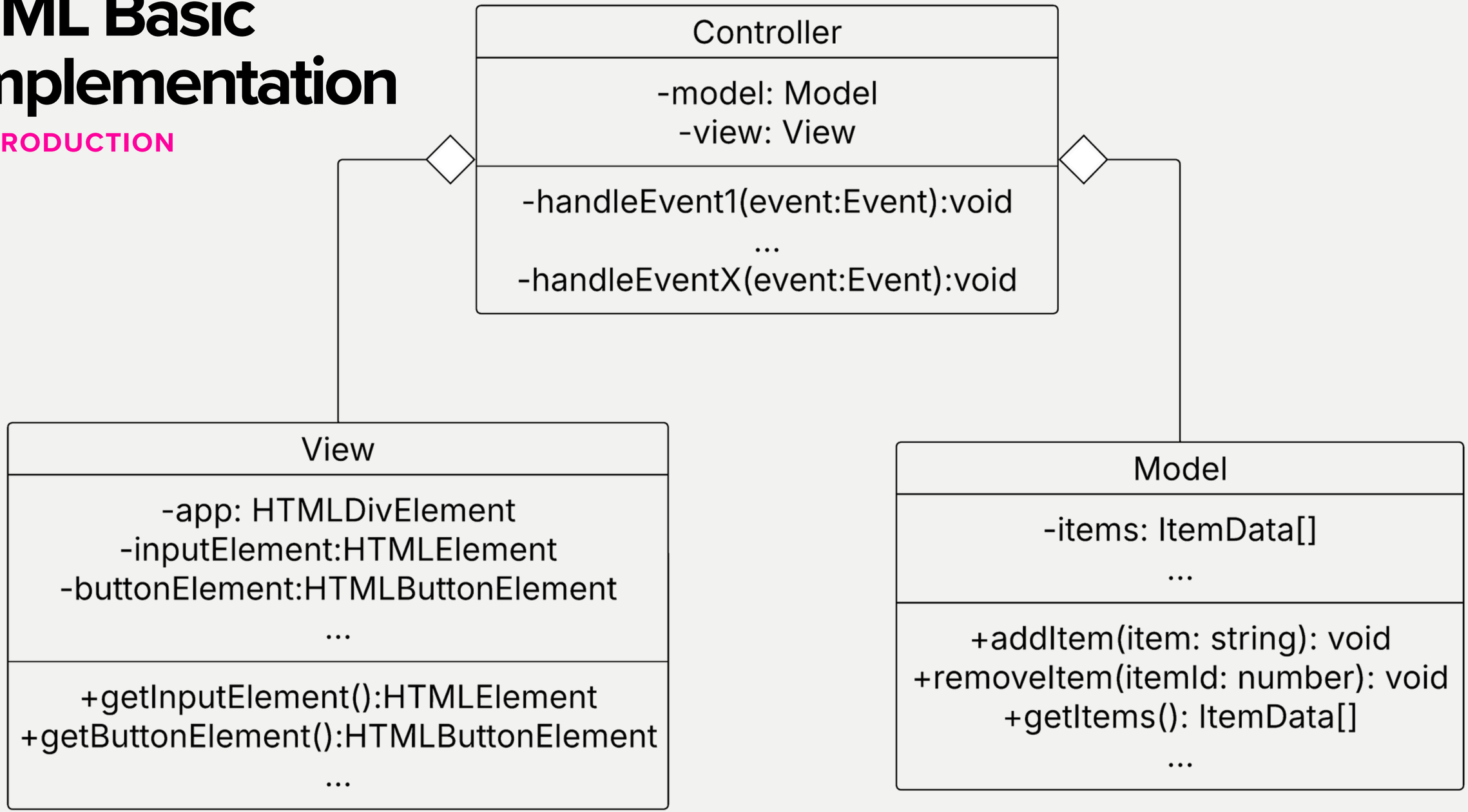
04

Kitchen (Model)

Manages all the processes necessary to prepare the food (data and business logic).

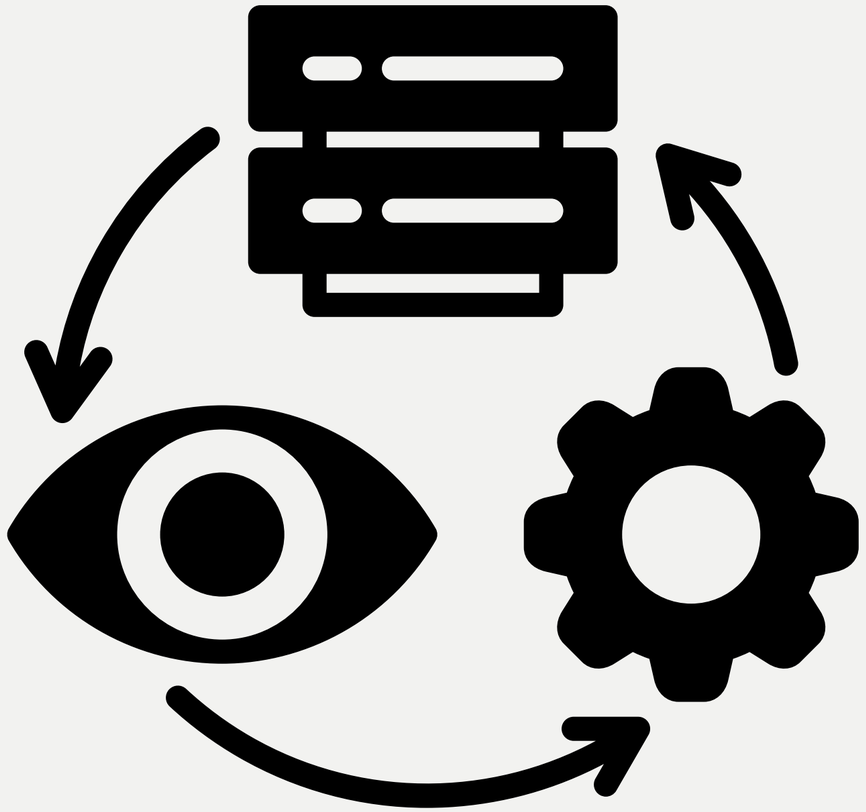
UML Basic Implementation

INTRODUCTION



MVC

Shopping List



HTML EVENTS

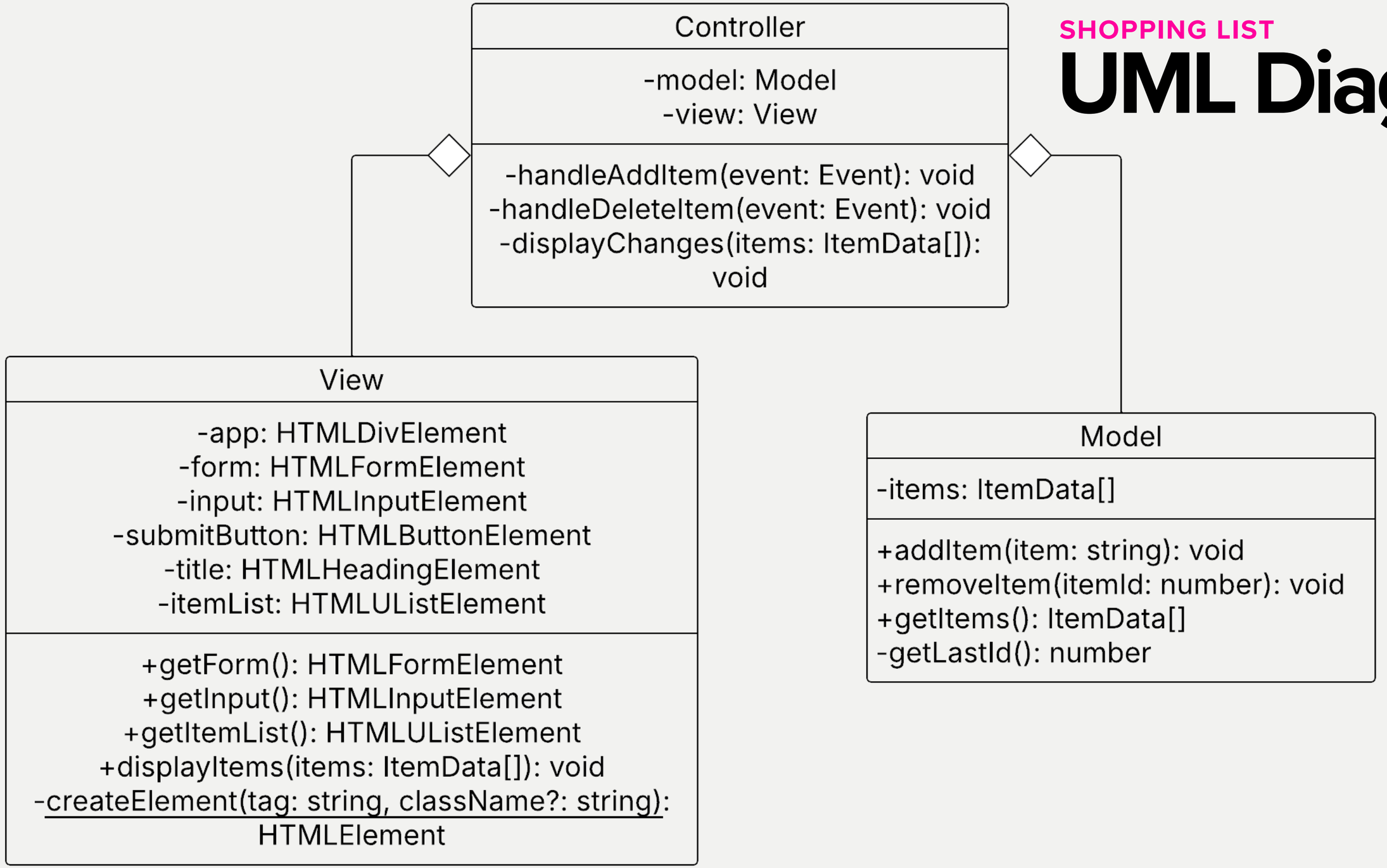
Prevent default

Disables the original behavior of the event.

```
private handleAddItem(event: Event): void {
  event.preventDefault();
  const input: HTMLInputElement = this.view.getInput();
  if (input.value) {
    this.model.addItem(input.value);
    this.displayChanges(this.model.getItems());
    // Resets the input field
    input.value = '';
  }
}
```

SHOPPING LIST

UML Diagram



SOLID

SOLID Application

01. SRP (Single Responsibility Principle)

02. OCP (Open/Closed Principle)

03. LSP (Liskov Substitution Principle)

04. ISP (Interface Segregation Principle)

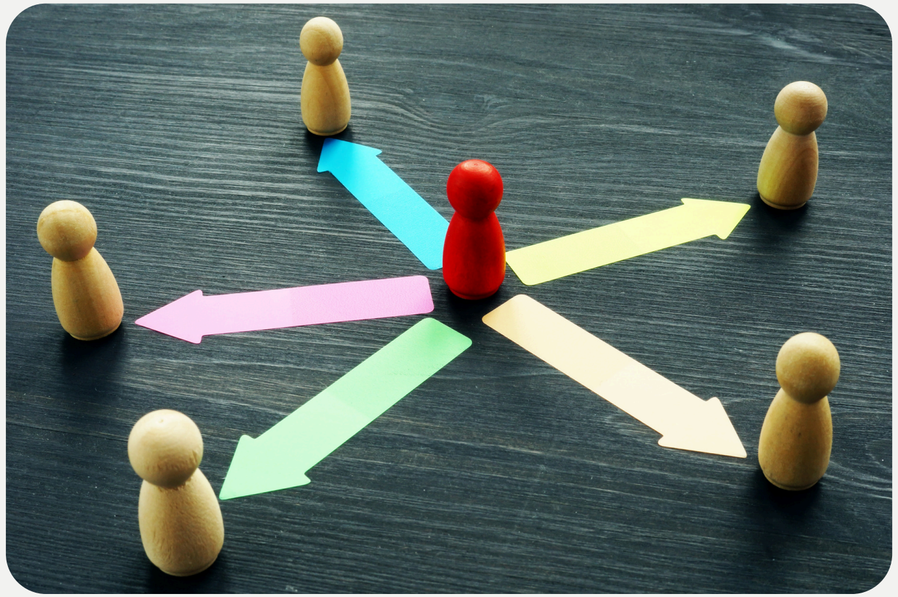
05. DIP (Dependency Inversion Principle)

SOLID

Common Problems

01

Concentration of responsibilities

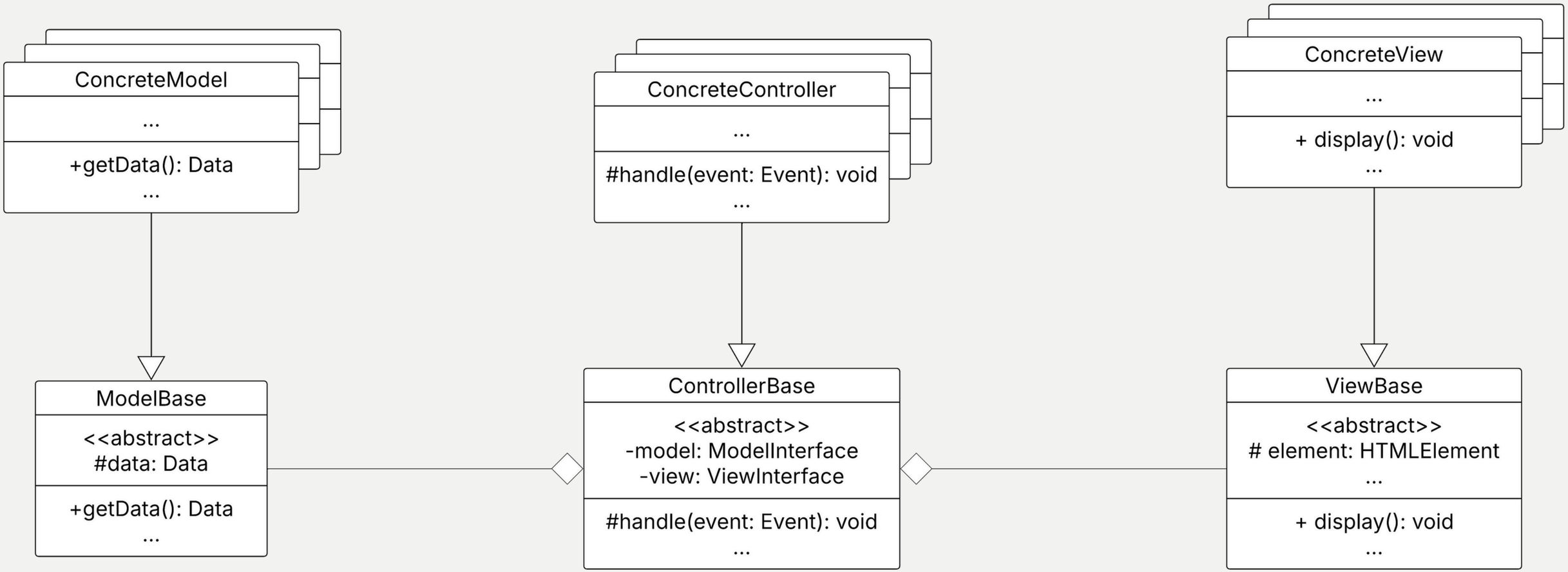


02

Desing Complexity

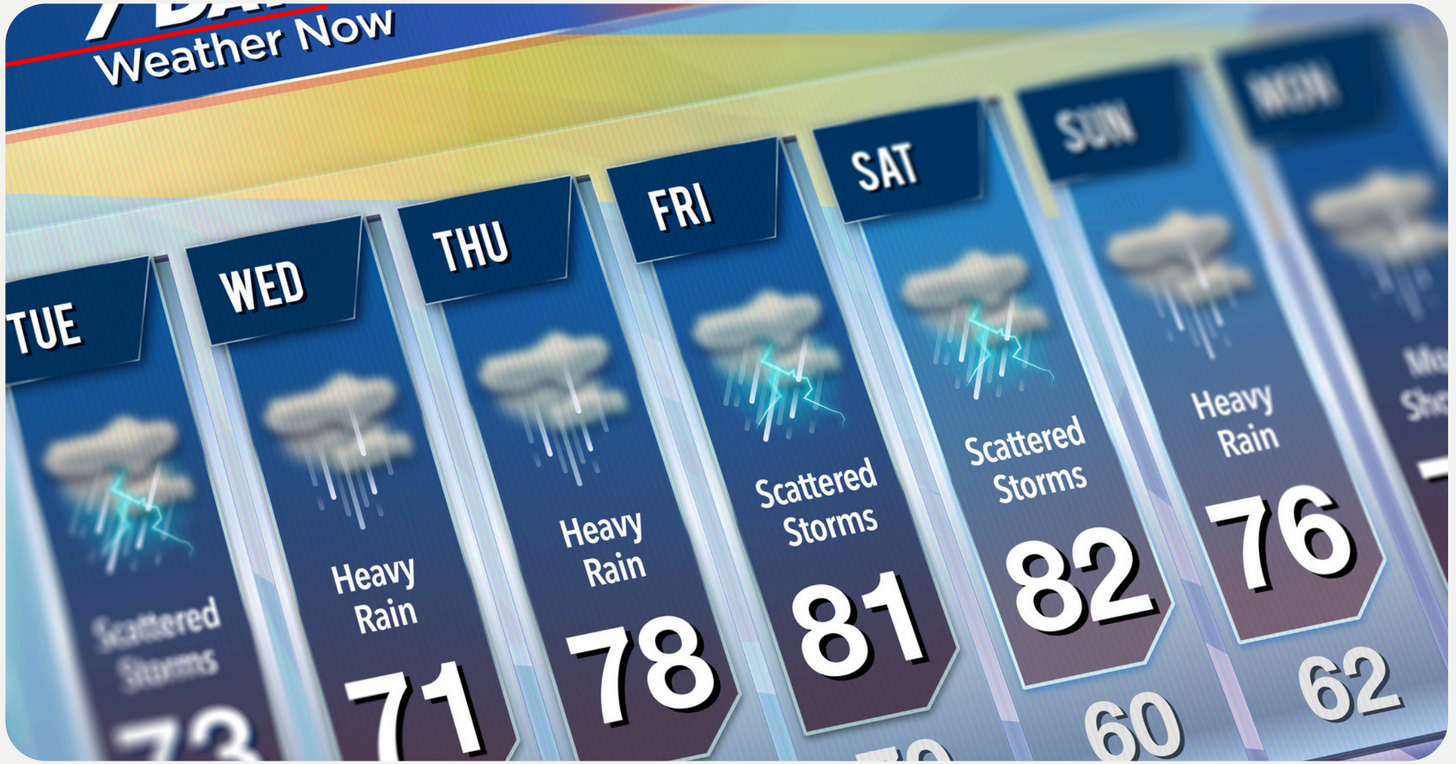
WEATHER APP

UML with abstractions



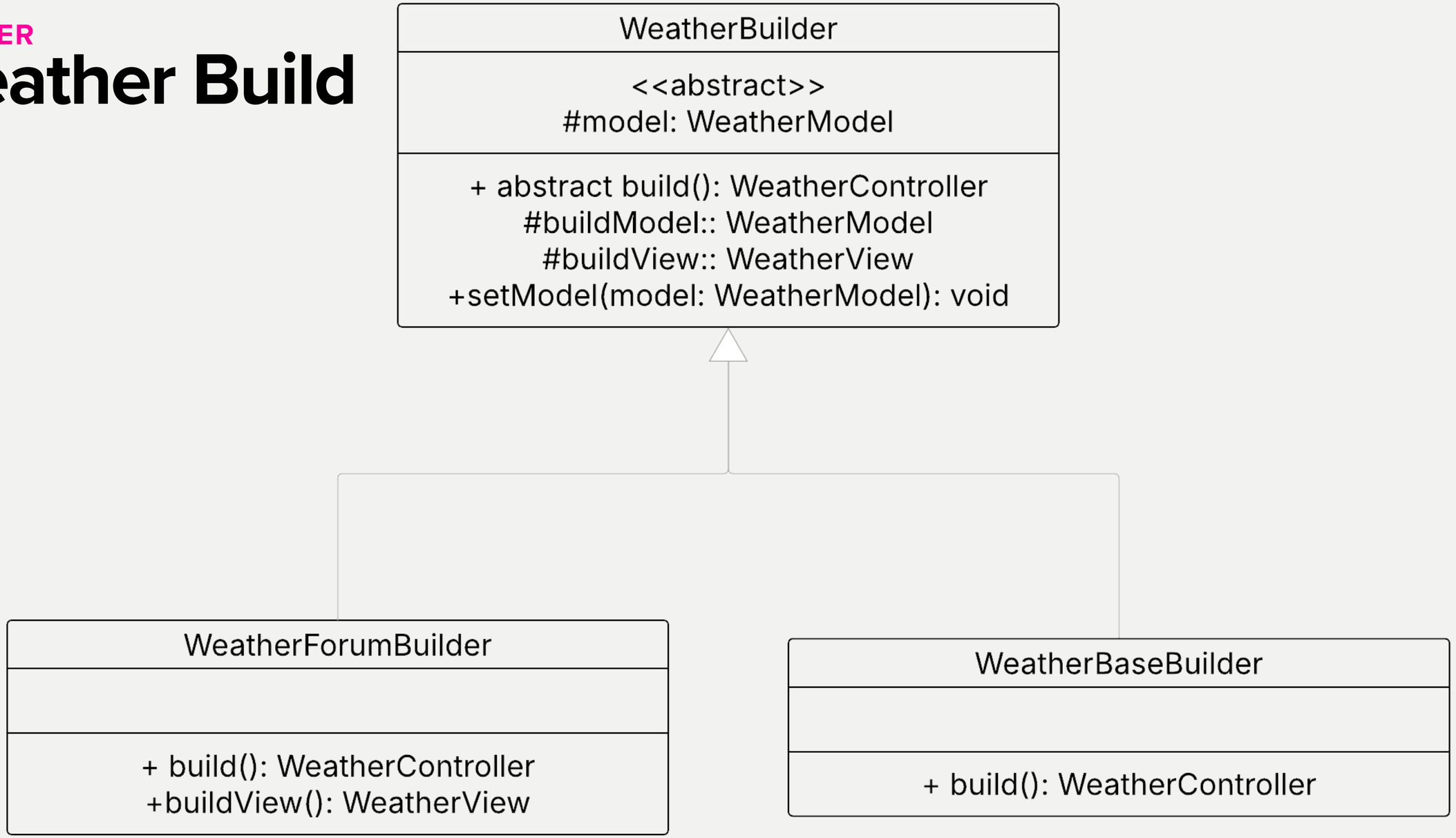
BUILDER

Weather APP



BUILDER

Weather Build



API

Weather API


```
private readonly API_KEY: string = '307b0af477e64520b4a103306252703';
private readonly PAGE_URL: string = 'http://api.weatherapi.com/v1';
private readonly ENDPOINT: string = '/forecast.json';
private readonly BASE_URL: string =
  `${this.PAGE_URL}${this.ENDPOINT}?key=${this.API_KEY}`;
```

API

Weather API

We fetch the data in the model using the API and the query stored

```
public override async getData(): Promise<WeatherData> {
  try {
    const URL: string = this.BASE_URL +
      `&q=${this.location}&days=${this.days}`;
    const response = await fetch(URL);
    const json: WeatherData = await response.json();
    this.calculateRainyDays(json);
    return json;
  } catch (error) {…
  }
}
```

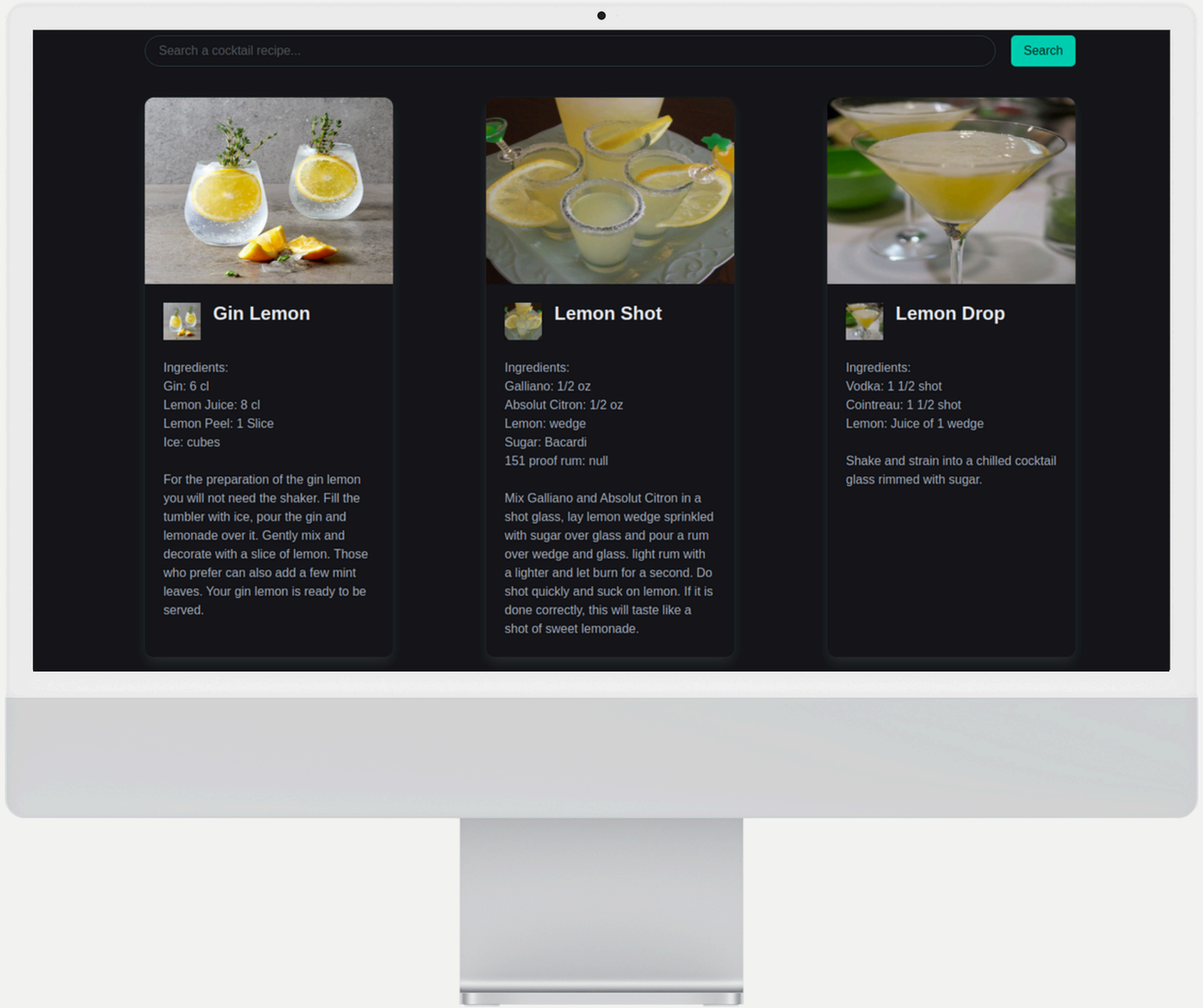
MVC

Cocktails APP

COCKTAILS APP

More complexity

Requires multiple reusable components and a higher level of abstraction to ensure modularity and maintainability.



COCKTAILS APP

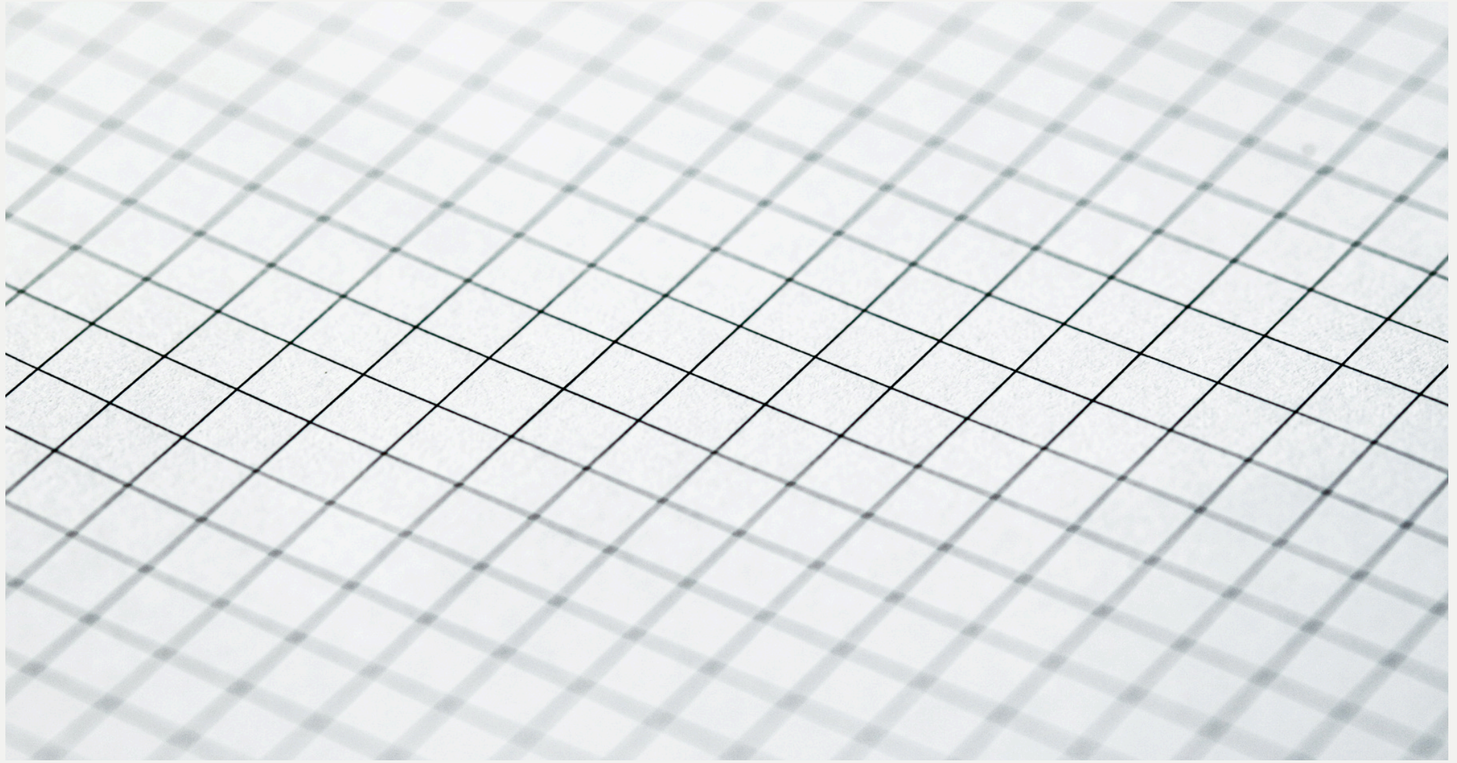
New abstractions

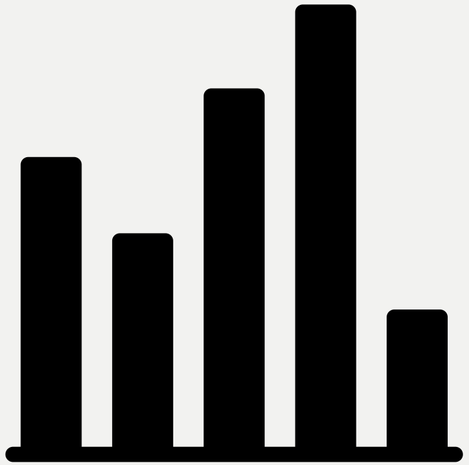
To manage this complexity, we will use generic interfaces to ensure the scalability and maintainability of the code.

```ts
export interface Model<T, EventInfo = {}> {
  handle(event: string, eventInfo: EventInfo): void;
  fetchData(): Promise<T>;
}

export interface View<Data, EventInfo = {}> {
  onEvent(
    event: string,
    callback: (eventInfo: EventInfo) => void
  ): void;
  render(data: Data): HTMLElement;
}

export interface Controller {
  onUpdate(
    callback: (element: HTMLElement) => void
  ): void;
  render(): void;
}
```

PAI LAB-8

PAI Lab-8





Alternatives to MVC

Model-View-ViewModel

Model-View-Presenter

Model-View-Intent

Libraries that use MVC

ASP.NET Core

Laravel

References

Inspiration for shopping list

Weather API

MVC Definition

MVC Explanation

Sliders (MDN)

Prevent default (MDN)

Select tag

Thank You

Contact info

guillermo.silva.26@ull.edu.es **Guillermo Silva González**

edhey.alonso.34@ull.edu.es **Himar Edhey Hernández Alonso**

samuel.rodriguez.44@ull.edu.es **Samuel Rodríguez Cuesta**