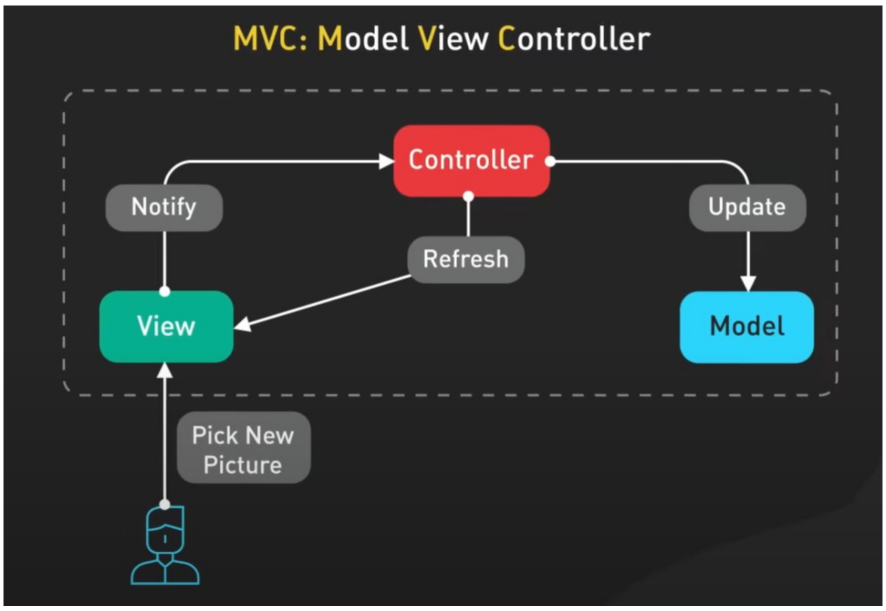
Understanding MVC and MVVM model



This diagram beautifully explains the **MVC (Model–View–Controller)** design pattern

Let's break it down step by step 👇

🧩 MVC Overview

MVC separates an application into three main logical components:

- **Model** → Manages the data, logic, and rules of the application.
- **View** → Represents the UI; what the user sees and interacts with.
- **Controller** → Handles user input and interacts with the model to update data or the view.

🔄 Step-by-Step Flow Explanation (Based on the Diagram)

1. 👤 User Interaction

- The user **interacts with the View** — in this example, they **pick a new picture**.
- This action (event) is captured by the **View**.

2. 📣 View → Controller (Notify)

- When the user performs an action, the **View notifies the Controller**.
- The Controller is responsible for interpreting this input — e.g., understanding that the user wants to change the displayed picture.

3. ⚙️ Controller → Model (Update)

- The **Controller processes the request** and decides what needs to change.
- It then **updates the Model** — for example, setting the new picture data in the application's state.

4. 💾 **Model → View (Notify / Data Change)**
   - After the Model's data is updated, it may **notify the View** (directly or indirectly) that its data has changed.
   - This ensures the UI reflects the latest data.
5. 🔄 **Controller → View (Refresh)**
   - The **Controller may also trigger a refresh** of the View so that the new data is displayed.
   - The **View then re-renders** the updated information to the user — showing the new picture in this case.

🧠 Conceptually

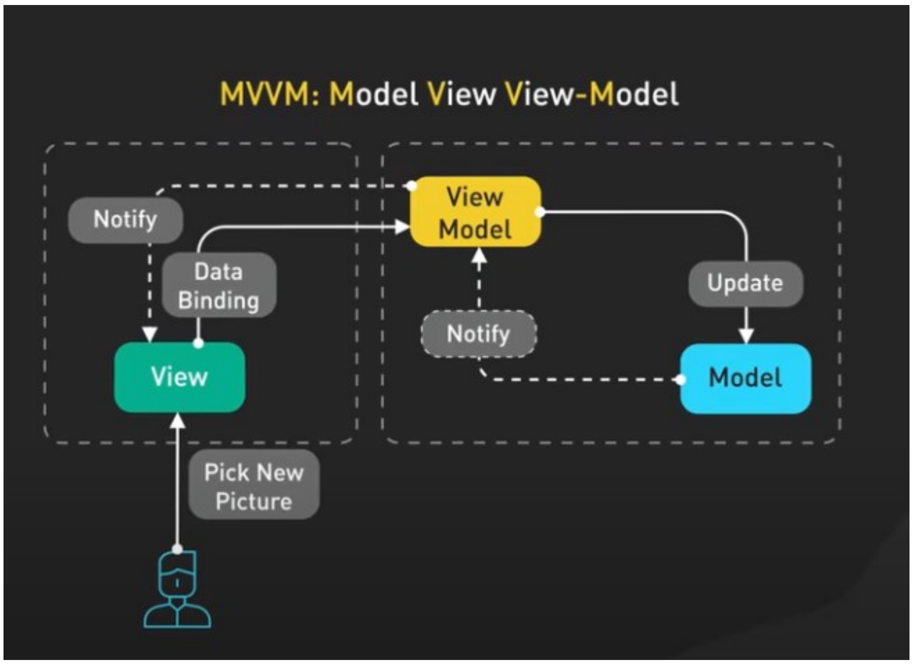
- **Model** = "What the app knows" (data and rules)
- **View** = "What the app shows" (UI)
- **Controller** = "What the app does" (logic and decision-making)

🎯 Key Idea

The **main goal of MVC** is **separation of concerns**:

- You can change the UI (View) without touching the logic (Controller or Model).
- You can modify business logic (Model) without affecting the presentation layer.

Explains MVVM (Model–View–ViewModel), a modern UI architectural pattern used in frameworks like Angular, Knockout.js, Vue, and Blazor.



Let's decode this flow step by step 👇

🧩 MVVM: Model–View–ViewModel

MVVM evolved from MVC to make **UI updates automatic** through **data binding** — eliminating manual refresh calls from the controller.

🔄 Step-by-Step Flow (from the diagram)

1. 👤 User Interaction

- The user performs an action — for example, **picks a new picture** in the UI.
- This action happens in the **View** (the HTML or UI component).

2. 📣 View → ViewModel

- The **View notifies the ViewModel** when something changes (like a user input).
- This happens automatically through **data binding**.
    - Example: If a textbox is bound to `viewModel.pictureName`, typing updates the property directly.

3. ⚙️ ViewModel → Model (Update)

- The **ViewModel** contains the presentation logic and acts as a **bridge** between the UI (View) and data (Model).
- It **updates the Model** with new data — e.g., setting the selected picture in the backend or state.

4. 💾 **Model → ViewModel (Notify)**

- The **Model** notifies the **ViewModel** when its data changes (e.g., from an API response or background update).

5. 🔄 **ViewModel → View (Automatic Data Binding)**

- Thanks to **data binding**, the **View automatically refreshes** when data in the **ViewModel** changes.
- There's **no explicit "refresh" command** needed (unlike in MVC).

🧠 Conceptual Roles

Component	Role	Example
Model	Manages data and business logic	REST API / Entity / Database
ViewModel	Holds UI-related data and logic	TypeScript class with observables
View	Displays UI and binds to ViewModel	HTML/Template (Angular, Knockout)

⚡ 2-Way Data Binding in Modern Frameworks

🔸 Angular Example

Angular uses the `[(ngModel)]` directive for two-way data binding:

```
<input [(ngModel)]="user.name" placeholder="Enter your name">
<p>Hello, {{ user.name }}</p>
```

- When user types → View updates `user.name` in ViewModel.
- When `user.name` changes in ViewModel → View updates automatically.

This is done internally using **property binding + event binding**:

```
<input [value]="user.name" (input)="user.name = $event.target.value">
```

🔸 Knockout.js Example

Knockout uses **observables** and the `data-bind` attribute:

```
<input data-bind="value: userName, valueUpdate: 'afterkeydown'" />
<p data-bind="text: userName"></p>

function AppViewModel() {
    this.userName = ko.observable("Rajeev");
}
ko.applyBindings(new AppViewModel());
```

- `ko.observable()` automatically tracks changes.
- UI updates when data changes, and vice versa.

🔸 How MVVM Differs from MVC

Aspect	MVC	MVVM
Intermediate Layer	Controller	ViewModel
Data Update	Manual (Controller refreshes View)	Automatic (data binding)
Suitable For	Server-side rendering (Spring MVC, Django)	Client-side frameworks (Angular, Knockout, Vue)
Data Flow	Mostly one-way	Two-way (View ↔ ViewModel)

🧠 Summary

MVC

- User → View → Controller → Model → View
- Manual update of UI

MVVM

- User → View ↔ ViewModel ↔ Model
- Automatic sync using data binding