Welcome To Prompt Js

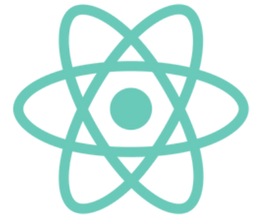
skillPrompt

DESIGN PATTERN

in React js

Presented by:
Susan Shakya
Jr.Frontend developer

What is Design Pattern?



- Design patterns are like **cooking recipes** for programmers. Each recipe in a cookbook provides a set of **instructions** that, when followed, help you create a delicious dish.
- Provides a set of **guidelines** or **templates.**
- Helps developers to solve specific type of problem **consistently** and **efficiently.**

.

Why to use Design Pattern?

Collaboration

Easier for team members to understand and work with each other’s code.

Code reusability

Create reusable components and structures, reducing code duplication.

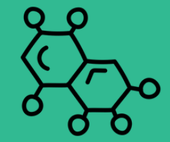
Maintainability

Well-implemented pattern helps to understand and maintain code.

Scalability

Manage complexity as your application grows.

Some React Design Patterns



Container & Presentation

Props combination

Custom Hooks

HOCs

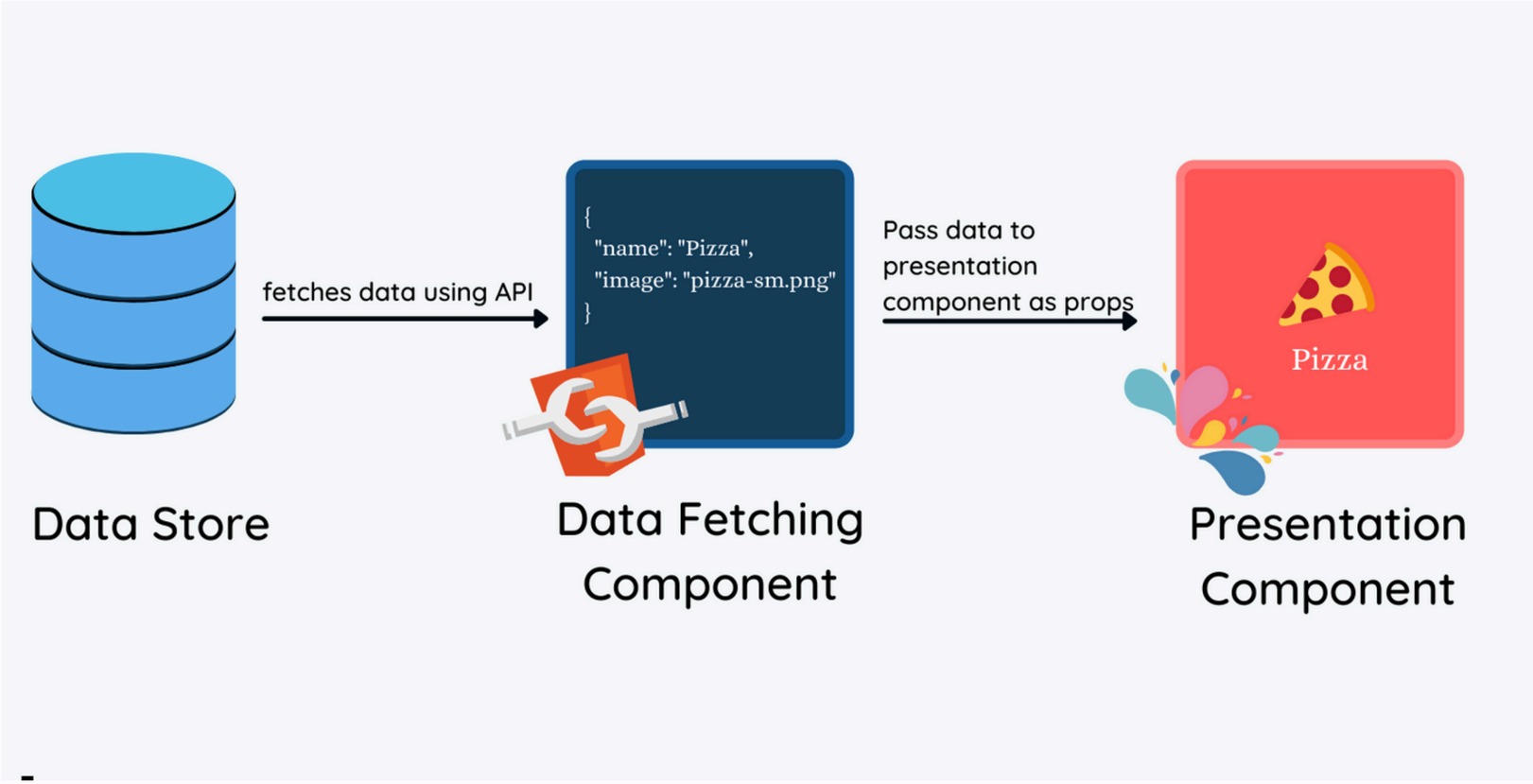
Container and Presentation Patterns

Container Components

Handles the states and logic, data fetching.

Presentation Components

Concern with the rendering UI elements based on Props.



-

```
const StarWarCharacterList = () => {
  const [character, setCharacter] = useState<Tstarswars[]>([]);

  const { isLoading, error } = useQuery({
    queryKey: ["starwars-api/api"],
    queryFn: async () => {
      try {
        const response = await fetch(
          "https://akabab.github.io/starwars-api/api/all.json"
        );
        const details = await response.json();
        if (!details) return;
        setCharacter(details);
        return details;
      } catch (error) {
        return <div>error</div>;
      }
    },
  });
  if (isLoading) {
    return <div>loading...</div>;
  }
  if (error) {
    return <div>something is wrong while fetching a data</div>;
  }
  console.log("character", character);
  if (!character) {
    return <div>No data found</div>;
  }

  return (
    <>
      <div className="text-white grid grid-cols-4 bg-[#332f2f]">
      {character.map((data) => (
        <ul key={data.id} className="border m-4 p-6 ">
          <li>
            <span className="font-bold">Id:</span> {data.id}
          </li>
          <li>
            <span className="font-bold">Name:</span> {data.name}
          </li>
          <li>
            <span className="font-bold">Gender:</span>
            {data.gender}
          </li>
        </ul>
      ))}
    </div>
    </>
  );
};
```

container

```
const StarWarCharacterList = () => {
  const { data, isLoading, error } = useQuery({
    queryKey: ["starwars-api/api"],
    queryFn: async () => {
      try {
        const response = await fetch(
          "https://akabab.github.io/starwars-api/api/all.json"
        );
        const details = await response.json();
        if (!details) return;
        // setCharacter(data);
        return details;
      } catch (error) {
        return <div>error</div>;
      }
    },
  });

  return (
    <>
      <CharacterListUI character={data} loading={isLoading} erro
    </>
  );
};
```

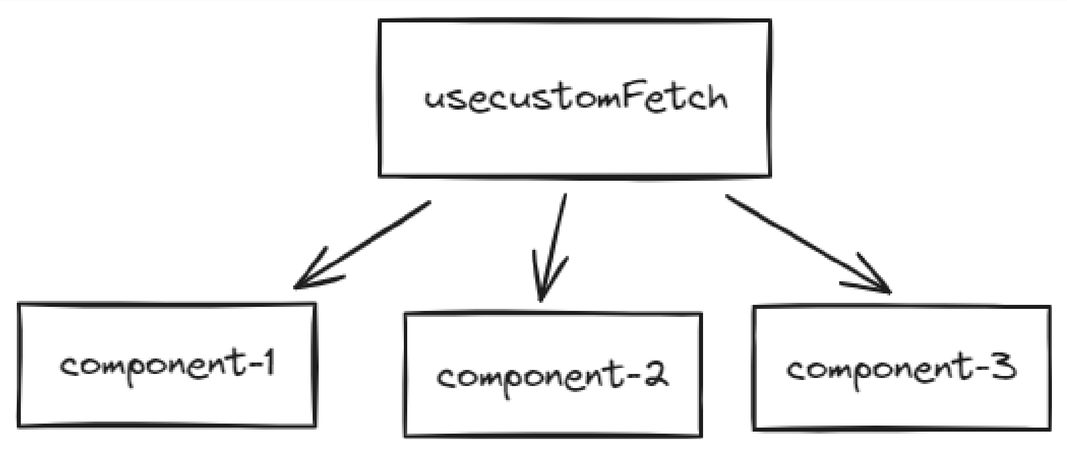
Presentation

```
const CharacterListUI = ({
  character,
}: {
  character: Tstarswars[];
}) => {
  return (
    <div className="text-white grid grid-cols-4 bg-[#332
      {character.map((data) => (
        <ul key={data.id} className="border m-4 p-6 ">
          <li>
            <span>Id:</span> {data.id} </li>
          <li><span>Name:</span> {data.name} </li>
          <li>  <span>Gender:</span>{data.gender}</li>
        </ul>
      ))}
    </div>
```

Let see output

Components Composition With Custom Hooks

- Extract component logic into **reusable functions**, make it easier to **find** and **update** specific functionality between components..
- Follow the naming convention of starting with **use**.
- Allow to use “useState” and other build in hooks



Custom Hooks

```
mport { toast } from 'sonner';

export const useToster = () => {
  const notify = (message: string, type: 'success' | 'error') => {
    if (type === 'error') {
      toast.error(message);
    } else {
      toast.success(message);
    }
  };

  return { notify };
};
```

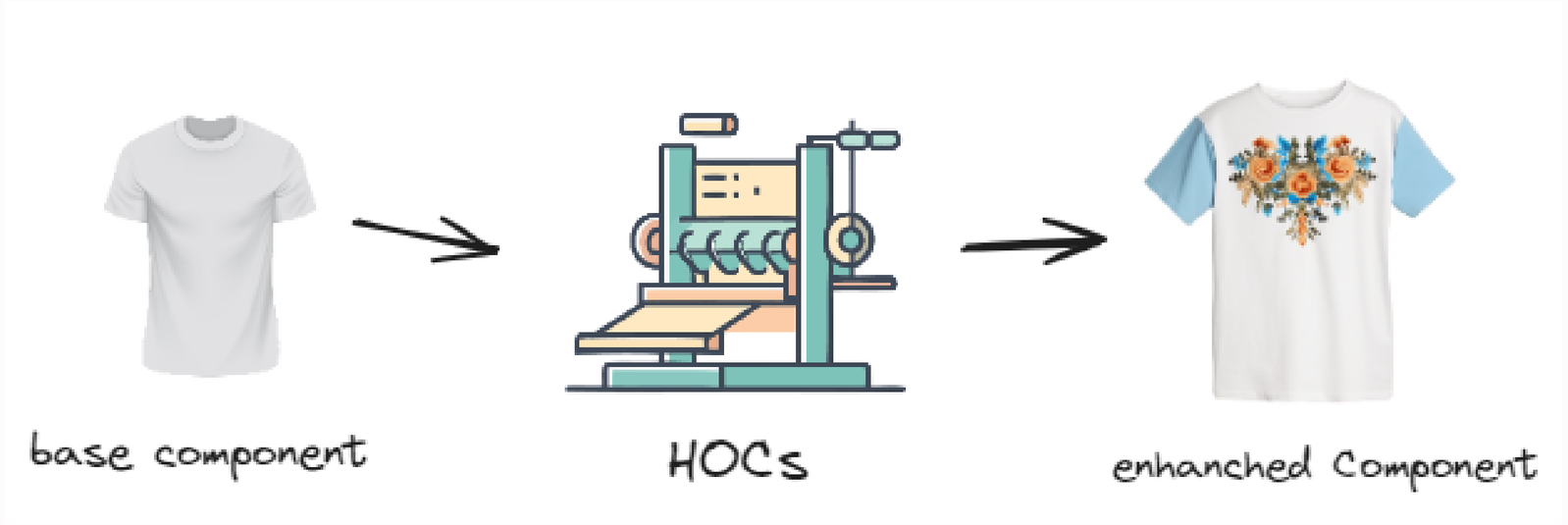
Components

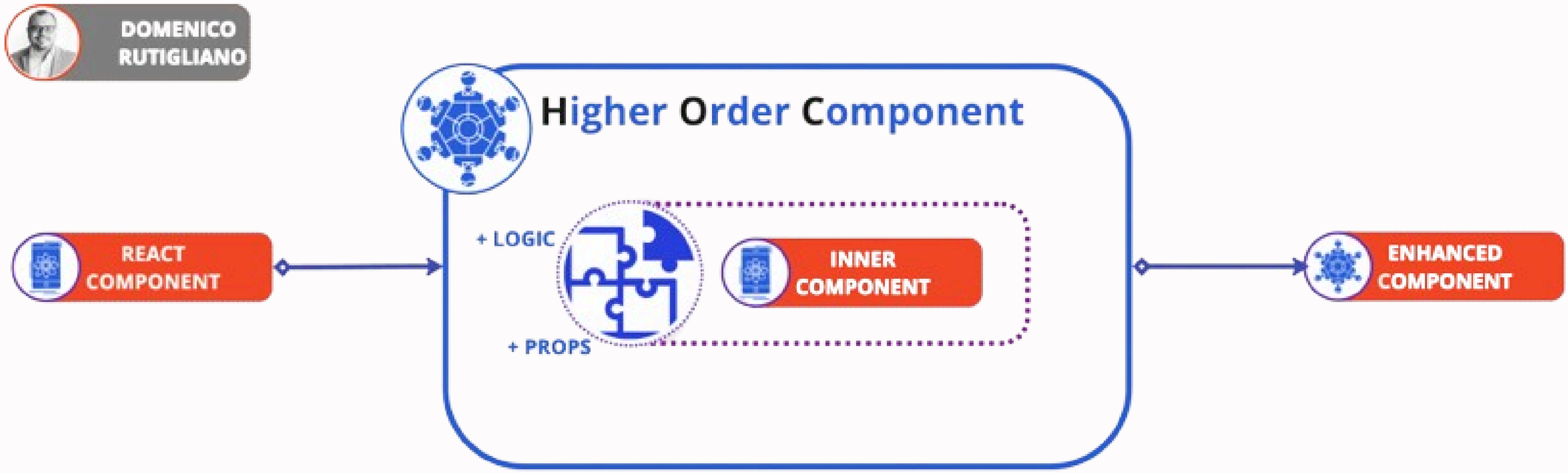
```
export const Logout = () => {
  const { notify } = useToster();

  const handleLogin = () => {
    notify("This is success toaster", "success");
  };

  const handleLoginError = () => {
    notify("This is error toaster", "error");
  };

  return (
    <div className="w-full h-screen flex gap-10 justify-center
     <button onClick={handleLogin}>Success</button>
     <button onClick={handleLoginError}>Error</button>
    </div>
  );
```

Let see output

Higher Order Components

- A function that **takes** a **component as argument** and returns a **new component** with enhanced behavior.
- Enhance the original component by **wrapping** it with additional logic.
- Follow the naming convention of starting with “**with”.**


base component
HOCs
enhanced Component


DOMENICO RUTIGLIANO
Higher Order Component
REACT COMPONENT
+ LOGIC
+ PROPS
INNER COMPONENT
ENHANCED COMPONENT

Base Components

```
const Counter= () => {
  return (
    <div className="w-full  flex flex-col items-center ;
      <div className="text-2xl ">counter: {count}</div>
      <div className="flex gap-4">
        <button>Addtion</button>
        <button>subtraction</button>
      </div>
    </div>
  );
};
```

HOCs

```
export const withHOC = (baseComponent) => {
  return (props) => {
    const [count, setCount] = useState(0);
    const increase = () => {
      setCount(count + 1);
    };

    const decrease = () => {
      if (count !== 0) {
        setCount(count - 1);
      }
    };

    return (
      <baseComponent
        count={count}
        increase={increase}
        decrease={decrease}
        {...props}
      />
    );
  };
};
```

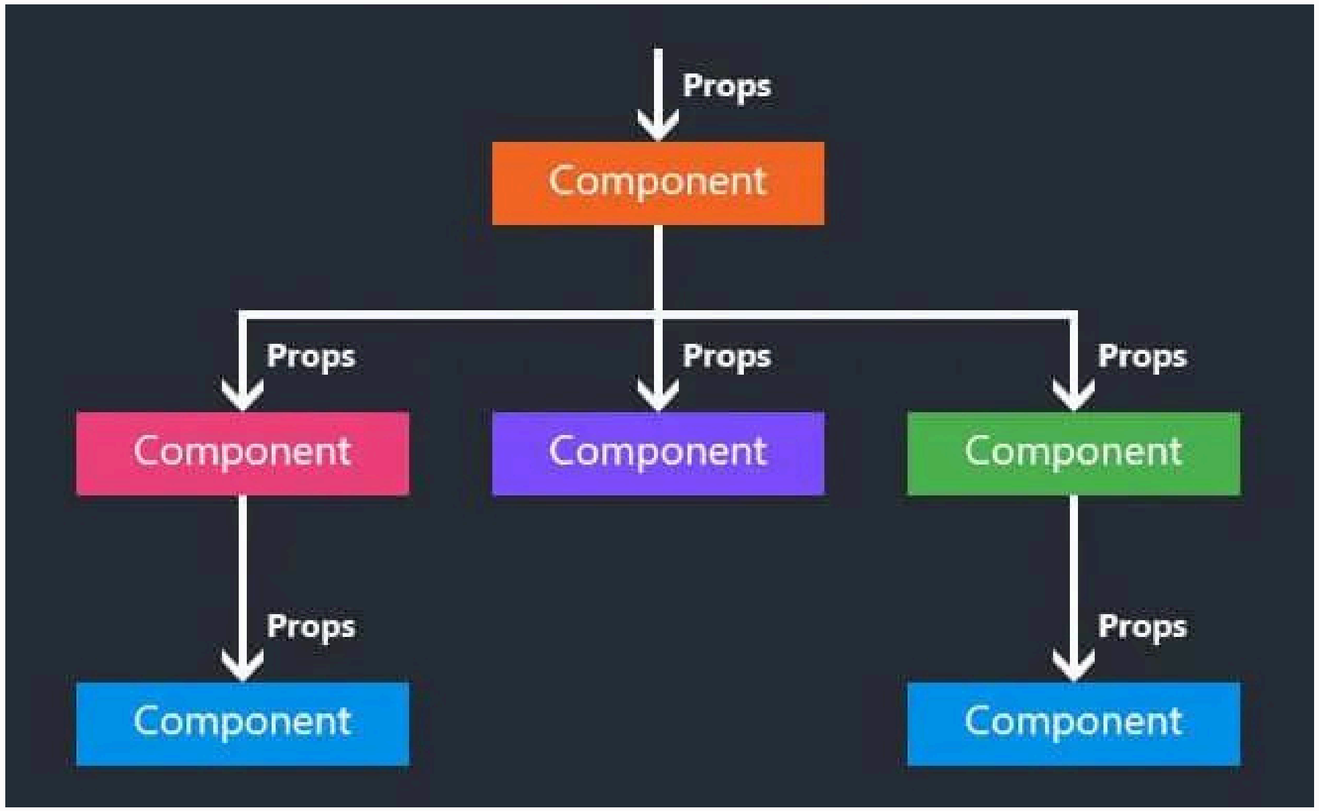
Enchanced Components

```tsx
const Counter: React.FC<TCounter> = (props) => {
  const { count, decrease, increase } = props;
  return (
    <div className="w-full h-64 flex flex-col items-center justify-center gap-4">
      <div className="text-2xl ">counter: {count}</div>
      <div className="flex gap-4">
        <button onClick={increase}>Additon</button>
        <button onClick={decrease}>Subtraction</button>
      </div>
    </div>
  );
};

export const EnhancedComponent = withHOC(Counter);
```

Let see output

Props (Properties) Combination

- Used to **pass** a **data** from a **parent** components to a **child** components
- Props are **read** only, i.e they cannot be modified by the receiving components.



Parent

```
export const ParentApp = () => {
  const paragraphProps = {
    name: "David Warner",
    profession: "cricketer",
    country: "Australia",
  };
  const Paragraph =
    "David Andrew Warner is a former Australian international cricketer and a for
  return <Props {...paragraphProps} Paragraph={Paragraph}></Props>;
};
```

Child component

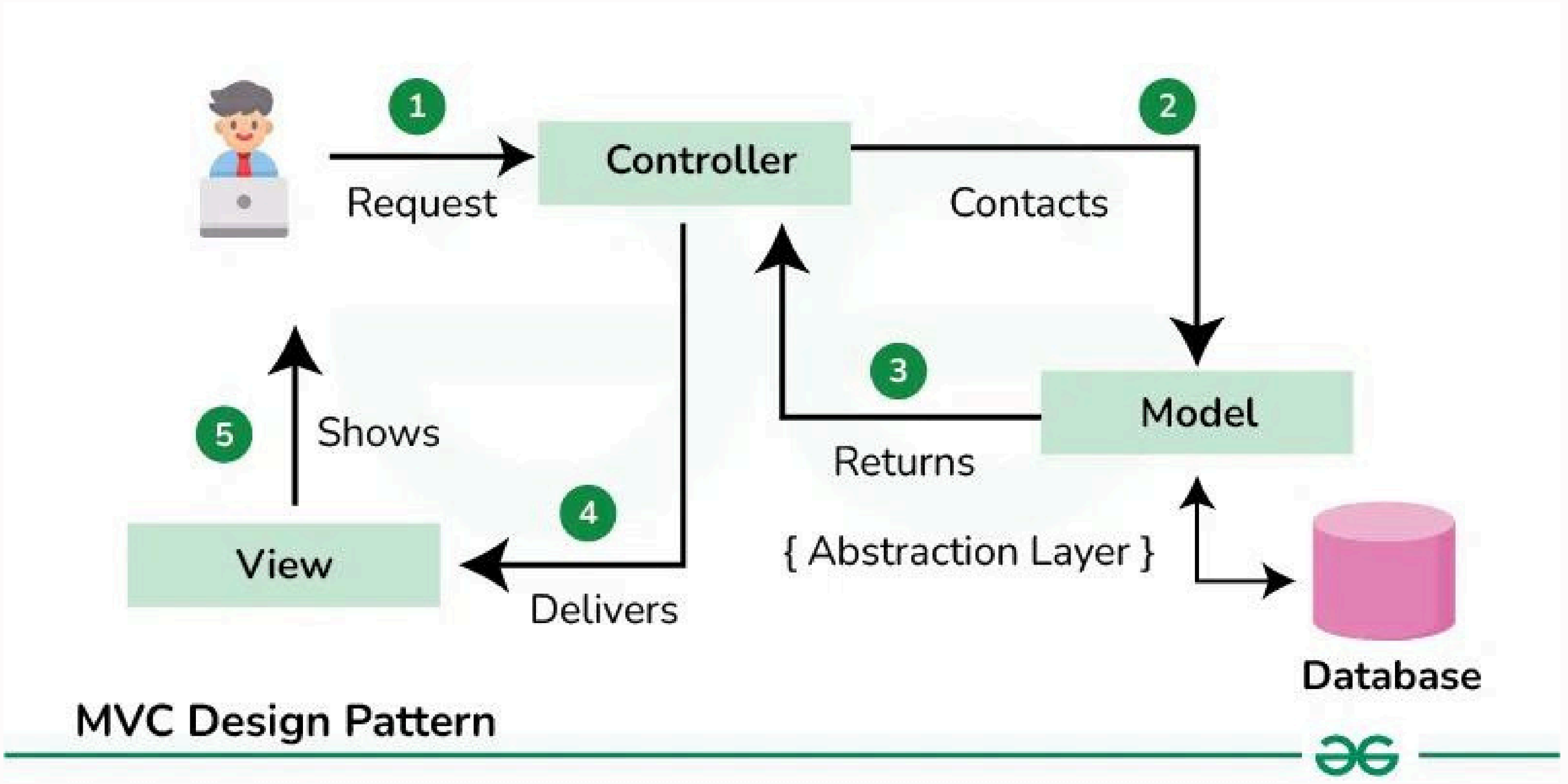
```
export function Props(props: any) {
  const { Paragraph } = props;
  return (
    <div className="w-full h-screen flex justify-center items-center">
      <div className="text-xl w-[60rem] border-2 mt-20 p-14">
        <h2>
          <span className="font-bold">Name:</span> {props.name}
        </h2>
        <h2>
          <span className="font-bold">Professional: </span> {props.profession}
        </h2>
        <h2>
          <span className="font-bold">Country: </span> {props.country}
        </h2>
        <p> {Paragraph}</p>
      </div>
    </div>
```

Let see output

Model view controller (MVC)

It's a way to separate code, making it easier to manage and develop.

- **Model**
  - It is structure of database which **interacts** and **manage** database.
- **View**
  - React components serve as the view.
  - It’s the web pages, buttons, forms, and everything visible in the browser.
- **Controller**
  - Handles incoming requests, processes them, and sends responses.



MVC Design Pattern

Any Queries