**Experiment 01**

**AIM :**

To design and develop responsive and interactive user interfaces (UIs) using Tailwind CSS, ensuring scalability, consistency, and modern design practices.

**THEORY :**

User Interface (UI) design is a key aspect of modern web development as it directly impacts user experience and usability. Traditional CSS requires writing large amounts of custom code, which can become difficult to maintain. Tailwind CSS is a utility-first CSS framework that simplifies this process by providing pre-defined utility classes. With Tailwind, developers can quickly build responsive and interactive interfaces without writing custom CSS from scratch. Its mobile-first approach and ease of customization make it a popular choice for building modern, responsive UIs.

**Utility-First CSS Framework**

* Tailwind CSS is different from traditional frameworks like Bootstrap because it does not provide pre-styled components.
* Instead, it offers thousands of utility classes that can be combined to create any design.
* This approach increases flexibility and avoids overwriting default styles.

**Responsive Design**

* Tailwind follows a **mobile-first** design strategy, meaning styles are first applied to smaller devices and then extended to larger screens.
* It uses built-in breakpoints such as sm, md, lg, xl, and 2xl for adaptive layouts.
* This ensures that a single codebase works seamlessly across mobiles, tablets, and desktops.

**Interactivity through State Variants**

* Tailwind provides pseudo-class variants like hover:, focus:, active:, and disabled:.
* These variants allow developers to add interactive behaviors (e.g., hover color change, focus outline) without writing extra CSS.

**Customization and Theming**

* Tailwind includes a configuration file (tailwind.config.js) where developers can define custom colors, fonts, and spacing values.
* This ensures that large projects maintain a consistent design system.
* It also supports dark mode and theming, making it adaptable for modern UI trends.

**Performance and Optimization**

* Tailwind integrates with tools like **PurgeCSS** to remove unused styles, keeping the final CSS file lightweight.
* This improves website loading speed and overall performance.

**Component Reusability**

* Although utility classes are written in HTML, Tailwind supports the creation of reusable components using frameworks like React, Vue, or Angular.
* This allows a balance between customization and maintainability.

**Advantages Over Traditional CSS**

* Eliminates the need to switch between HTML and CSS files frequently.
* Prevents issues like naming conflicts in class selectors.
* Increases development speed by reducing boilerplate CSS code.

**SOURCE CODE :**

**Tailwind.config.ts**

import type { Config } from "tailwindcss";

export default {

darkMode: ["class"],

content: [

"./pages/\*\*/\*.{ts,tsx}",

"./components/\*\*/\*.{ts,tsx}",

"./app/\*\*/\*.{ts,tsx}",

"./src/\*\*/\*.{ts,tsx}",

],

prefix: "",

theme: {

container: {

center: true,

padding: '2rem',

screens: {

'2xl': '1400px'

}

},

extend: {

colors: {

border: 'hsl(var(--border))',

input: 'hsl(var(--input))',

ring: 'hsl(var(--ring))',

background: 'hsl(var(--background))',

foreground: 'hsl(var(--foreground))',

primary: {

DEFAULT: 'hsl(var(--primary))',

foreground: 'hsl(var(--primary-foreground))'

},

secondary: {

DEFAULT: 'hsl(var(--secondary))',

foreground: 'hsl(var(--secondary-foreground))'

},

destructive: {

DEFAULT: 'hsl(var(--destructive))',

foreground: 'hsl(var(--destructive-foreground))'

},

muted: {

DEFAULT: 'hsl(var(--muted))',

foreground: 'hsl(var(--muted-foreground))'

},

accent: {

DEFAULT: 'hsl(var(--accent))',

foreground: 'hsl(var(--accent-foreground))'

},

popover: {

DEFAULT: 'hsl(var(--popover))',

foreground: 'hsl(var(--popover-foreground))'

},

card: {

DEFAULT: 'hsl(var(--card))',

foreground: 'hsl(var(--card-foreground))'

},

sidebar: {

DEFAULT: 'hsl(var(--sidebar-background))',

foreground: 'hsl(var(--sidebar-foreground))',

primary: 'hsl(var(--sidebar-primary))',

'primary-foreground': 'hsl(var(--sidebar-primary-foreground))',

accent: 'hsl(var(--sidebar-accent))',

'accent-foreground': 'hsl(var(--sidebar-accent-foreground))',

border: 'hsl(var(--sidebar-border))',

ring: 'hsl(var(--sidebar-ring))'

},

printz: {

50: '#f0f7ff',

100: '#e0effe',

200: '#bae0fd',

300: '#7ac7fc',

400: '#34aaf7',

500: '#0c8de3',

600: '#0270c0',

700: '#035a9c',

800: '#074d81',

900: '#0c426b',

950: '#082b48',

}

},

borderRadius: {

lg: 'var(--radius)',

md: 'calc(var(--radius) - 2px)',

sm: 'calc(var(--radius) - 4px)'

},

keyframes: {

'accordion-down': {

from: {

height: '0'

},

to: {

height: 'var(--radix-accordion-content-height)'

}

},

'accordion-up': {

from: {

height: 'var(--radix-accordion-content-height)'

},

to: {

height: '0'

}

}

},

animation: {

'accordion-down': 'accordion-down 0.2s ease-out',

'accordion-up': 'accordion-up 0.2s ease-out'

}

}

},

plugins: [require("tailwindcss-animate")],

} satisfies Config;

**OrderForm.tsx**

import React, { useState } from "react";

import { useNavigate } from "react-router-dom";

import { Button } from "@/components/ui/button";

import { Input } from "@/components/ui/input";

import { Textarea } from "@/components/ui/textarea";

import { Label } from "@/components/ui/label";

import { RadioGroup, RadioGroupItem } from "@/components/ui/radio-group";

import {

Select,

SelectContent,

SelectItem,

SelectTrigger,

SelectValue,

} from "@/components/ui/select";

import { useToast } from "@/components/ui/use-toast";

import { useOrders } from "@/context/OrderContext";

import { v4 as uuidv4 } from "uuid";

const OrderForm: React.FC = () => {

const navigate = useNavigate();

const { toast } = useToast();

const { currentDocument, addOrder, setCurrentOrder } = useOrders();

const [formData, setFormData] = useState({

customerName: "",

customerEmail: "",

customerPhone: "",

copies: 1,

colorMode: "color" as "color" | "bw",

paperSize: "a4" as "a4" | "a3" | "letter",

notes: "",

});

const [isSubmitting, setIsSubmitting] = useState(false);

const handleInputChange = (

e: React.ChangeEvent<HTMLInputElement | HTMLTextAreaElement>

) => {

const { name, value } = e.target;

setFormData({ ...formData, [name]: value });

};

const handleSelectChange = (name: string, value: string) => {

setFormData({ ...formData, [name]: value });

};

const handleColorModeChange = (value: string) => {

setFormData({ ...formData, colorMode: value as "color" | "bw" });

};

const handleCopiesChange = (e: React.ChangeEvent<HTMLInputElement>) => {

const value = parseInt(e.target.value);

if (!isNaN(value) && value >= 1) {

setFormData({ ...formData, copies: value });

}

};

const calculatePrice = () => {

const base = formData.colorMode === "color" ? 10.0 : 2.0;

const multiplier =

formData.paperSize === "a3"

? 2

: formData.paperSize === "letter"

? 1.1

: 1;

return parseFloat((base \* formData.copies \* multiplier).toFixed(2));

};

const handleSubmit = async (e: React.FormEvent) => {

e.preventDefault();

if (!currentDocument) {

toast({

title: "No document selected",

description: "Please upload a document first",

variant: "destructive",

});

navigate("/upload");

return;

}

setIsSubmitting(true);

try {

const price = calculatePrice();

const order = {

id: uuidv4(),

document: currentDocument,

...formData,

createdAt: new Date(),

status: "pending" as const,

price: price,

};

addOrder(order);

setCurrentOrder(order);

toast({

title: "Order submitted",

description: "Your order has been successfully submitted.",

});

navigate("/payment");

} catch (error) {

console.error("Error submitting order:", error);

toast({

title: "Submission failed",

description: "Failed to submit your order. Please try again.",

variant: "destructive",

});

} finally {

setIsSubmitting(false);

}

};

return (

<form

onSubmit={handleSubmit}

className="space-y-8 max-w-4xl mx-auto p-4 sm:p-6 lg:p-10"

>

{/\* Contact Info \*/}

<div className="grid grid-cols-1 md:grid-cols-2 gap-6">

<div className="flex flex-col gap-2">

<Label htmlFor="customerName">Full Name</Label>

<Input

id="customerName"

name="customerName"

value={formData.customerName}

onChange={handleInputChange}

required

className="w-full"

/>

</div>

<div className="flex flex-col gap-2">

<Label htmlFor="customerEmail">Email</Label>

<Input

id="customerEmail"

name="customerEmail"

type="email"

value={formData.customerEmail}

onChange={handleInputChange}

required

className="w-full"

/>

</div>

<div className="md:col-span-2 flex flex-col gap-2">

<Label htmlFor="customerPhone">Phone Number</Label>

<Input

id="customerPhone"

name="customerPhone"

type="tel"

value={formData.customerPhone}

onChange={handleInputChange}

required

className="w-full"

/>

</div>

</div>

{/\* Print Options \*/}

<div className="grid grid-cols-1 md:grid-cols-2 gap-6">

<div className="flex flex-col gap-2">

<Label htmlFor="copies">Number of Copies</Label>

<Input

id="copies"

name="copies"

type="number"

min="1"

value={formData.copies}

onChange={handleCopiesChange}

required

className="w-full"

/>

</div>

<div className="flex flex-col gap-2">

<Label>Color Mode</Label>

<RadioGroup

value={formData.colorMode}

onValueChange={handleColorModeChange}

className="flex flex-row sm:flex-col md:flex-row gap-4"

>

<div className="flex items-center space-x-2">

<RadioGroupItem value="color" id="color" />

<Label htmlFor="color">Color</Label>

</div>

<div className="flex items-center space-x-2">

<RadioGroupItem value="bw" id="bw" />

<Label htmlFor="bw">Black & White</Label>

</div>

</RadioGroup>

</div>

<div className="flex flex-col gap-2">

<Label htmlFor="paperSize">Paper Size</Label>

<Select

value={formData.paperSize}

onValueChange={(value) => handleSelectChange("paperSize", value)}

>

<SelectTrigger className="w-full">

<SelectValue placeholder="Select paper size" />

</SelectTrigger>

<SelectContent>

<SelectItem value="a4">A4</SelectItem>

<SelectItem value="a3">A3</SelectItem>

<SelectItem value="letter">US Letter</SelectItem>

</SelectContent>

</Select>

</div>

<div className="md:col-span-2 flex flex-col gap-2">

<Label htmlFor="notes">Additional Notes</Label>

<Textarea

id="notes"

name="notes"

value={formData.notes}

onChange={handleInputChange}

placeholder="Any special instructions..."

className="w-full"

/>

</div>

</div>

{/\* Summary \*/}

<div className="bg-muted p-4 sm:p-6 rounded-lg shadow-md text-sm sm:text-base">

<div className="flex justify-between">

<span>Document:</span>

<span>{currentDocument?.name || "None"}</span>

</div>

<div className="flex justify-between">

<span>Copies:</span>

<span>{formData.copies}</span>

</div>

<div className="flex justify-between">

<span>Color Mode:</span>

<span>{formData.colorMode === "color" ? "Color" : "B&W"}</span>

</div>

<div className="flex justify-between">

<span>Paper Size:</span>

<span>{formData.paperSize.toUpperCase()}</span>

</div>

<div className="flex justify-between font-semibold mt-3 text-lg">

<span>Total Price:</span>

<span>₹{calculatePrice()}</span>

</div>

</div>

{/\* Submit \*/}

<div className="text-right">

<Button

type="submit"

disabled={isSubmitting}

className="w-full sm:w-auto"

>

{isSubmitting ? "Processing..." : "Continue to Payment"}

</Button>

</div>

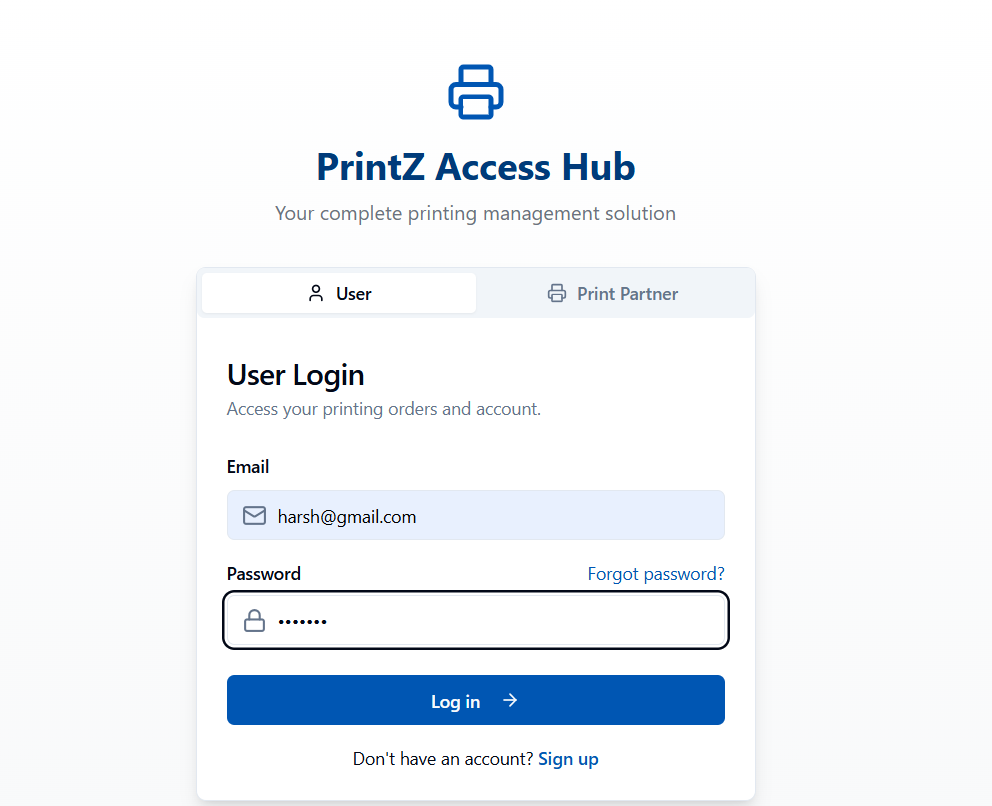
</form>

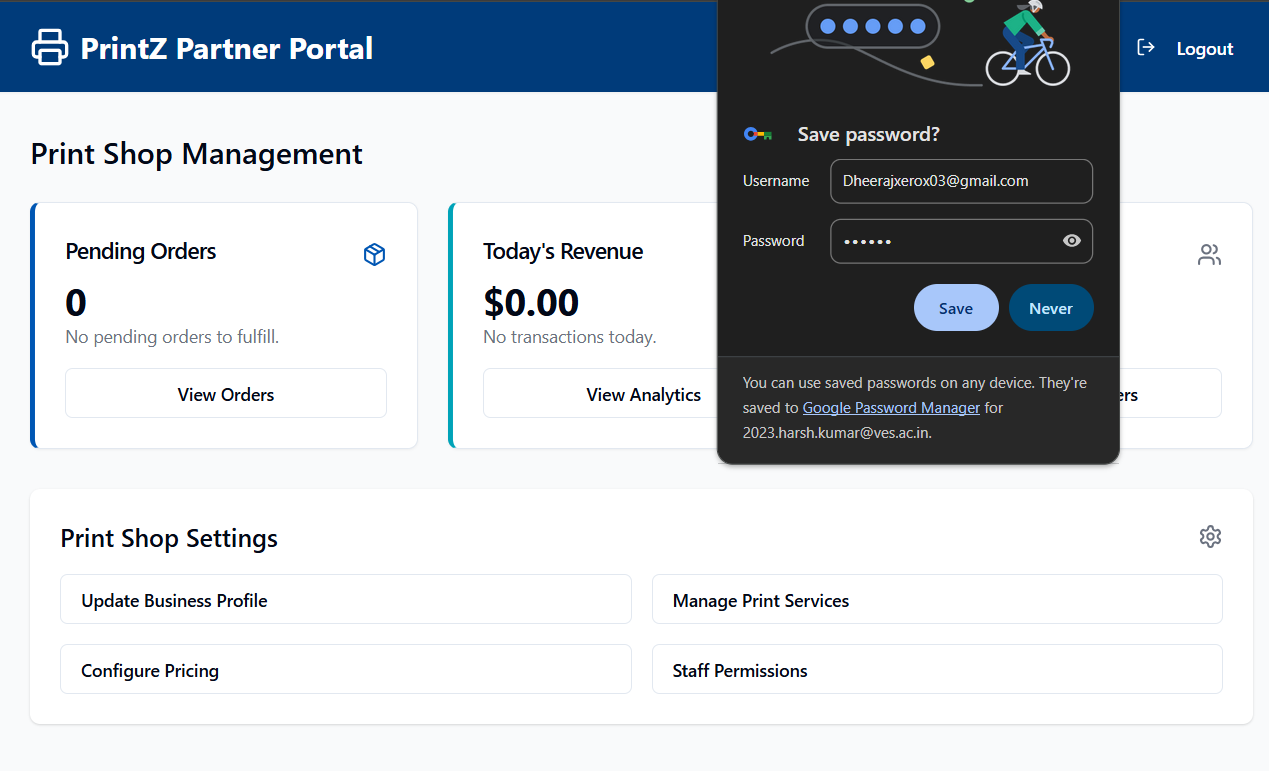
);

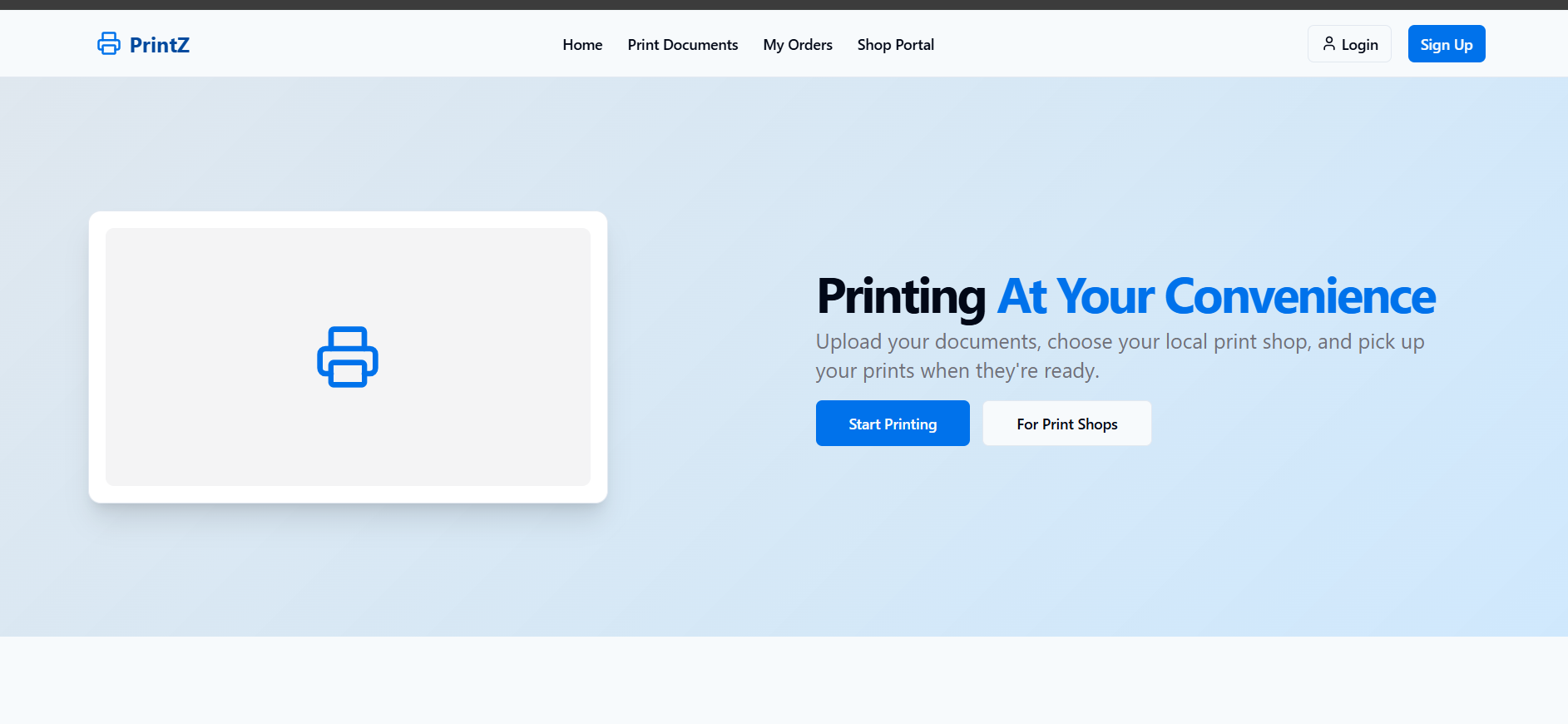
};

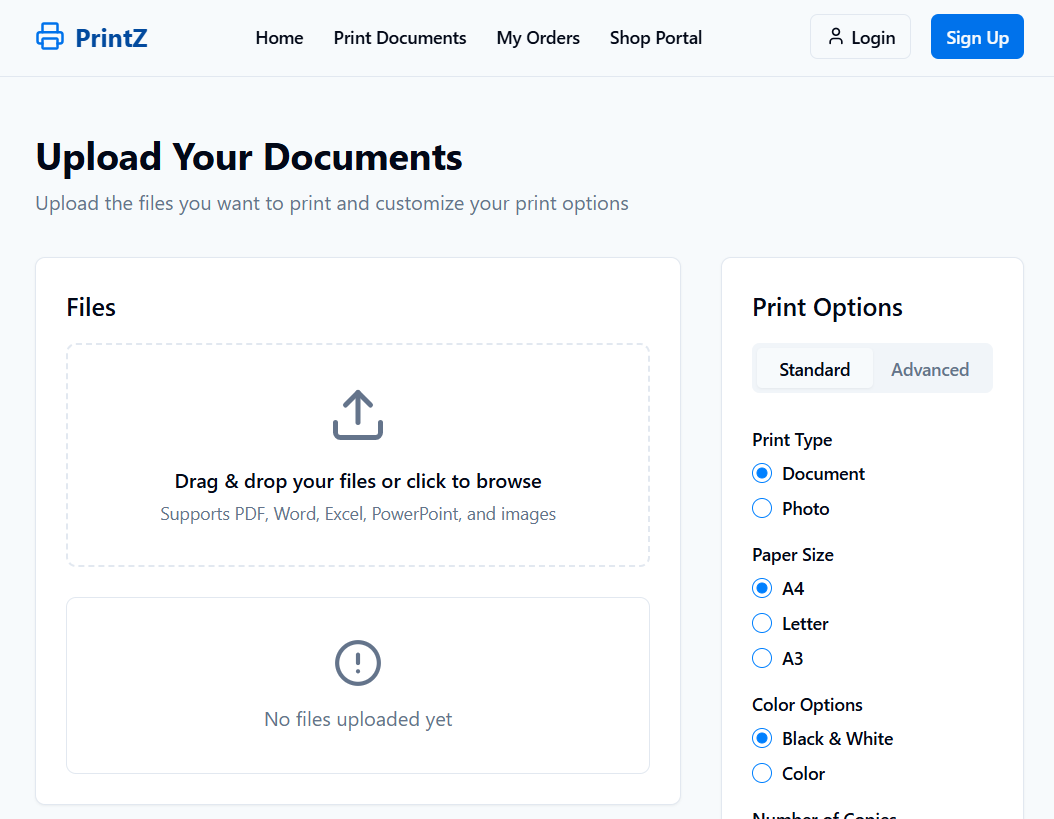
export default OrderForm;

**OUTPUT :**









**ADDITIONAL FEATURES:**

AUTHENTICATION SYSTEM

UPLOAD FUNCTIONALITY

REACT CONTEXT API

BREAKPOINTS FOR DIFFERENT SCREEN SIZES

INTEGRATION FOR RAZORPAY(FOR ONLINE PAYMENT) IN FUTURE

**CONCLUSION:**

By using Tailwind CSS, developers can design responsive, scalable, and interactive UIs more efficiently compared to traditional CSS approaches. Its utility-first philosophy eliminates the need for extensive custom styling, while its responsive utilities and state variants ensure adaptability across devices and interactivity for enhanced user experience.