Using Zod for Form Validation

What is Zod?

Zod is a **schema validation library** that allows you to define and validate data structures declaratively. It's lightweight, easy to use, and integrates seamlessly with React.

Why Use Zod?

- **Declarative Syntax**: Define validation rules in a clean and readable way.
- **Customizable**: Add custom error messages and validation logic.
- **Composable**: Combine schemas to create complex validation rules.

How to Use Zod

1. Install Zod

```
npm install zod
```

2. Define a Zod schema

```
import { z } from "zod";

const userSchema = z.object({
  name: z.string(),
  age: z.number().min(18),  // Age must be at least 18
  email: z.string().email() // Must be a valid email
});
```

3. Use the Schema with Data

```
const userData = {
  name: "John Doe",
  age: 25,
  email: "john@example.com",
};
```

```
// ✅ Safe Parsing
const result = userSchema.safeParse(userData);
console.log(result.success); // true

// ❌ Handling Invalid Data
const invalidData = { name: "Alice", age: 16, email: "not-an-email" };
const invalidResult = userSchema.safeParse(invalidData);
console.log(invalidResult.success); // false
console.log(invalidResult.error.format()); // Detailed error messages
```

Additional Zod Features:

- Nested Objects

```
const orderSchema = z.object({
  customer: z.object({
    name: z.string(),
    email: z.string().email(),
  }),
  items: z.array(
    z.object({
      name: z.string(),
      quantity: z.number().min(1),
    })
  ),
});

const orderData = {
  customer: { name: "John", email: "john@example.com" },
  items: [{ name: "Laptop", quantity: 1 }],
};
```

- Zod supports advanced validation like regex, custom functions, and more.

```
const passwordSchema = z
  .string()
  .min(8, 'Password must be at least 8 characters')
  .regex(/[A-Z]/, 'Password must contain at least one uppercase letter')
  .regex(/[0-9]/, 'Password must contain at least one number');
```

- You can mark fields as optional or nullable.

```
const profileSchema = z.object({
  name: z.string().min(1, 'Name is required'),
  bio: z.string().optional(), // Optional field
  age: z.number().nullable(), // Nullable field
});
```

- Composing Schemas: You can combine schemas to create reusable validation rules.

```
const emailSchema = z.string().email('Invalid email address');
const passwordSchema = z.string().min(6, 'Password must be at least 6 c

const loginSchema = z.object({
  email: emailSchema,
  password: passwordSchema,
});
```

- Refine with custom logic: Zod supports asynchronous validation for scenarios like checking if a username is already taken.

```
const usernameSchema = z.string().refine(
  async (username) => {
    const isAvailable = await checkUsernameAvailability(username);
    return isAvailable;
  },
```

```
  { message: 'Username is already taken' }
);
```

- Specify a default value

```
const configSchema = z.object({
  timeout: z.number().default(5000),
});

console.log(configSchema.parse({})); // { timeout: 5000 }
```

Assignments

1. **Objective:** Create a Zod schema to validate a **user registration form** with the following fields:

- `username` (string, required, min 3 characters, max 20)
- `email` (valid email format)
- `password` (string, min 8 characters, must contain at least one number)

📌 **Bonus Challenge:**

- Add a **confirm password** field that must match `password`.
- Ensure `username` contains only **letters, numbers, or underscores**.

2. **Objective:** Validate an **array of guest objects** where each guest has:

- `name` (string, required, min 2 characters)
- `age` (number, required, must be 18 or older)
- `gender` (enum: "Male", "Female", "Other")

📌 **Bonus Challenge:**

- Limit the number of guests to a **maximum of 5**.

- Ensure `name` only contains **letters and spaces**.

3. **Objective:** Create a schema for **validating product data** with the following fields:

- `title` (string, required, min 5 characters)
- `price` (number, required, greater than 0)
- `stock` (number, required, must be an integer)
- `category` (enum: "Electronics", "Clothing", "Books", "Toys")
- `tags` (optional array of strings, max 5 items)

📌 **Bonus Challenge:**

- Ensure `price` has at **most 2 decimal places**.
- Ensure `stock` is always **greater than or equal to 0**.