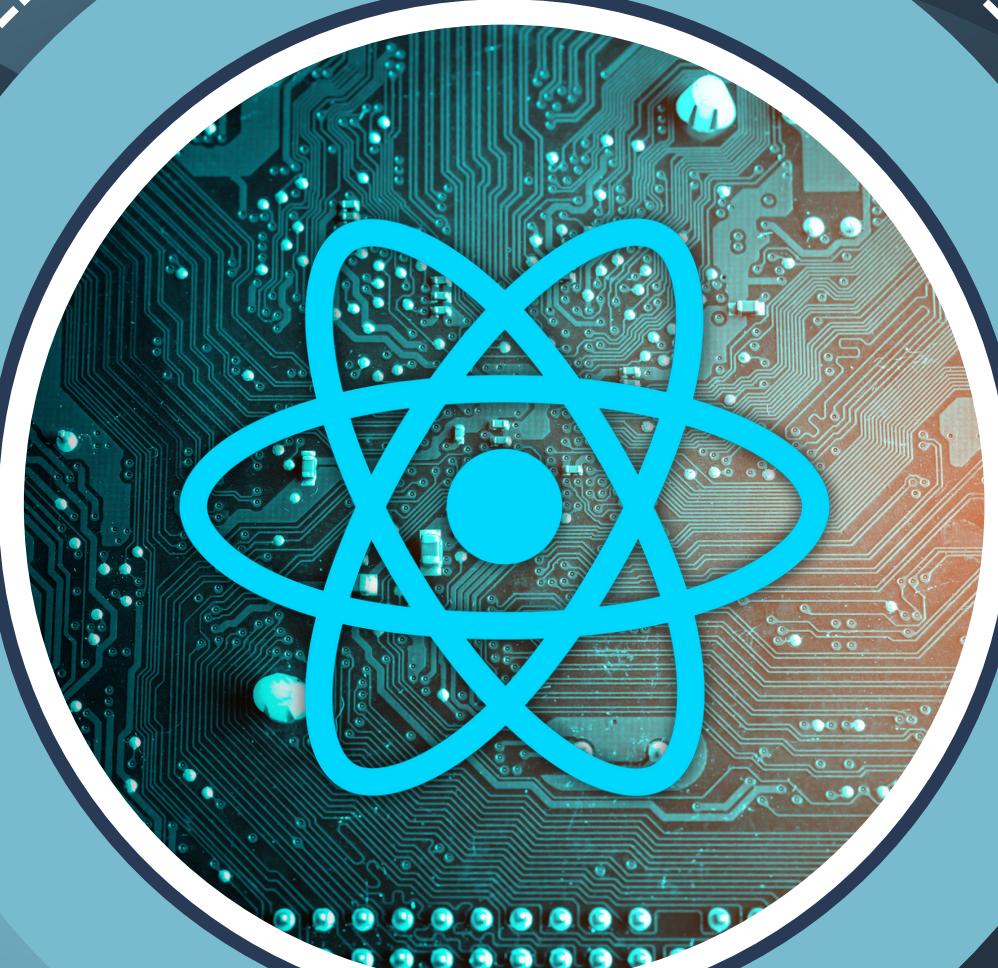


Chain of Responsibility

دیزاین پترن هادر React

Behavioral Patterns

به زبان ساده همراه با مثال های عملی
کاربرد دیزاین پترن ها برای برنامه نویسان React



Behavioral Patterns

Chain of Responsibility

الگوی **Chain of Responsibility** به ما این امکان را می دهد که یک درخواست را در طول زنجیره ای از پردازشگرها (Handlers) ارسال کنیم. هر پردازشگر یا درخواست را پردازش می کند یا آن را به پردازشگر بعدی در زنجیره انتقال می دهد.

فرض کنید می خواهیم در یک فرم فیلد شماره موبایل را اعتبارسنجی کنیم. برای این کار باید چندین شرط را بررسی کنیم. به عنوان مثال شماره موبایل باید عدد باشد، باید با ۰۹ شروع شده باشد، باید ۱۱ رقم باشد.

در ادامه می خواهیم مراحل این اعتبارسنجی را به صورت مرحله به مرحله انجام دهیم تا دقیقا به کاربر بگیم مشکل کجاست.

بخش Handler

ابتدا یک Interface به شکل زیر ایجاد می کنیم.



```
● ● ●  
export default interface Handler {  
  setNext(handler: Handler): Handler;  
  validate(value: string): string | null;  
}
```

setNext

validate

برای تنظیم مرحله بعدی
اعتبارسنجی

اگر اعتبارسنجی موفق باشد، مقدار
null برگردانده می شود.

اولین پردازشگر - باید عدد باشد

اولین پردازشگر خود را که برای اعتبارسنجی عدد بودن مقدار ورودی می باشد، را به شکل زیر می سازیم.



IsNumberHandler

```
import Handler from '../handler.interface';

export default class IsNumberHandler implements Handler {
    private nextHandler: Handler | null = null;

    setNext(handler: Handler): Handler {
        this.nextHandler = handler;
        return handler;
    }

    validate(value: string): string | null {
        if (!/\d+/.test(value)) {
            return 'Only numbers are allowed.';
        }
        return this.nextHandler ? this.nextHandler.validate(value) : null;
    }
}
```



دومین پردازشگر-باید با '09' شروع شده باشد

دومین پردازشگر خود را که برای اعتبارسنجی این است که مقدار ورودی با '09' شروع شده باشد را به شکل زیر می سازیم.



```
import Handler from '../handler.interface';

export default class PrefixHandler implements Handler {
    private nextHandler: Handler | null = null;

    setNext(handler: Handler): Handler {
        this.nextHandler = handler;
        return handler;
    }

    validate(value: string): string | null {
        const prefix = '09';
        if (!value.startsWith(prefix)) {
            return `It must start with ${prefix}.`;
        }
        return this.nextHandler ? this.nextHandler.validate(value) : null;
    }
}
```

09

سومین پردازشگر - باید 11 رقم باشد

سومین پردازشگر خود را که برای اعتبارسنجی این است که مقدار ورودی 11 رقم باشد را به شکل زیر می سازیم.

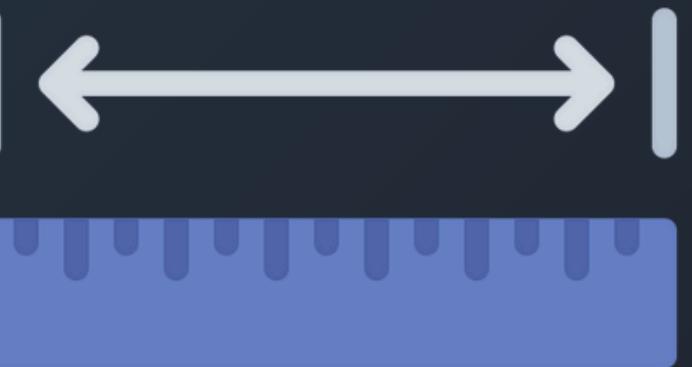
LengthHandler

```
import Handler from '../handler.interface';

export default class LengthHandler implements Handler {
  private nextHandler: Handler | null = null;

  setNext(handler: Handler): Handler {
    this.nextHandler = handler;
    return handler;
  }

  validate(value: string): string | null {
    const len = 11;
    if (value.length !== len) {
      return `It must be ${len} digits.`;
    }
    return this.nextHandler ? this.nextHandler.validate(value) : null;
  }
}
```



بخش MobileNumberForm

یک Component به شکل زیر می سازیم و با استفاده از پردازشگرهایی که ساختیم عملیات اعتبارسنجی را پیاده سازی می کنیم.



MobileNumberForm

```
import { useState } from 'react';
import IsNumberHandler from './handlers/is-number.handler';
import PrefixHandler from './handlers/prefix.handler';
import LengthHandler from './handlers/length.handler';

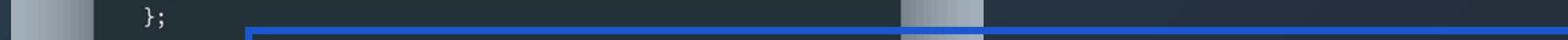
const MobileNumberForm: React.FC = () => {
  const [mobile, setMobile] = useState<string>('');
  const [error, setError] = useState<string | null>(null);

  const numberHandler = new IsNumberHandler();
  const prefixHandler = new PrefixHandler();
  const lengthHandler = new LengthHandler();

  numberHandler.setNext(prefixHandler).setNext(lengthHandler);

  const handleSubmit = (event: React.FormEvent) => {
    event.preventDefault();
    const validationError = numberHandler.validate(mobile);
    setError(validationError);
    if (!validationError) {
      alert('The mobile number is valid!');
    }
  };
  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label htmlFor='mobile'>Mobile Number:</label>
        <input
          id='mobile'
          type='text'
          value={mobile}
          onChange={(e) => setMobile(e.target.value)}
        />
      </div>
      {error && <p style={{ color: 'red' }}>{error}</p>}
      <button type='submit'>Check</button>
    </form>
  );
}

export default MobileNumberForm;
```



```
<form onSubmit={handleSubmit}>
  <div>
    <label htmlFor='mobile'>Mobile Number:</label>
    <input
      id='mobile'
      type='text'
      value={mobile}
      onChange={(e) => setMobile(e.target.value)}
    />
  </div>
  {error && <p style={{ color: 'red' }}>{error}</p>}
  <button type='submit'>Check</button>
</form>
```

طرز استفاده از MobileNumberForm

در پایان به شکل زیر از MobileNumberForm استفاده می کنیم.



ChainOfResponsibilityDesignPattern



```
import MobileNumberForm from './pattern/MobileNumberForm';

const ChainOfResponsibilityDesignPattern = () => {
  return (
    <div>
      <MobileNumberForm />
    </div>
  );
};

export default ChainOfResponsibilityDesignPattern;
```