

Template Method

دیزاین پترن ها در React

Behavioral Patterns

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



Behavioral Patterns

Template Method



الگوی **Template Method** ساختار یک الگوریتم را در یک کلاس پایه تعریف می‌کند و اجازه می‌دهد زیربخش‌های خاص الگوریتم در کلاس‌های فرزند بازنویسی شوند، بدون اینکه ساختار کلی تغییر کند.

فرض کنید می‌خواهیم دو فرم برای لاین و ثبت نام بسازیم، هر دو فرم دارای یک سری مراحل هستند که از لحاظ انتزاعی شبیه به هم هستند اما از لحاظ پیاده‌سازی متفاوت هستند.

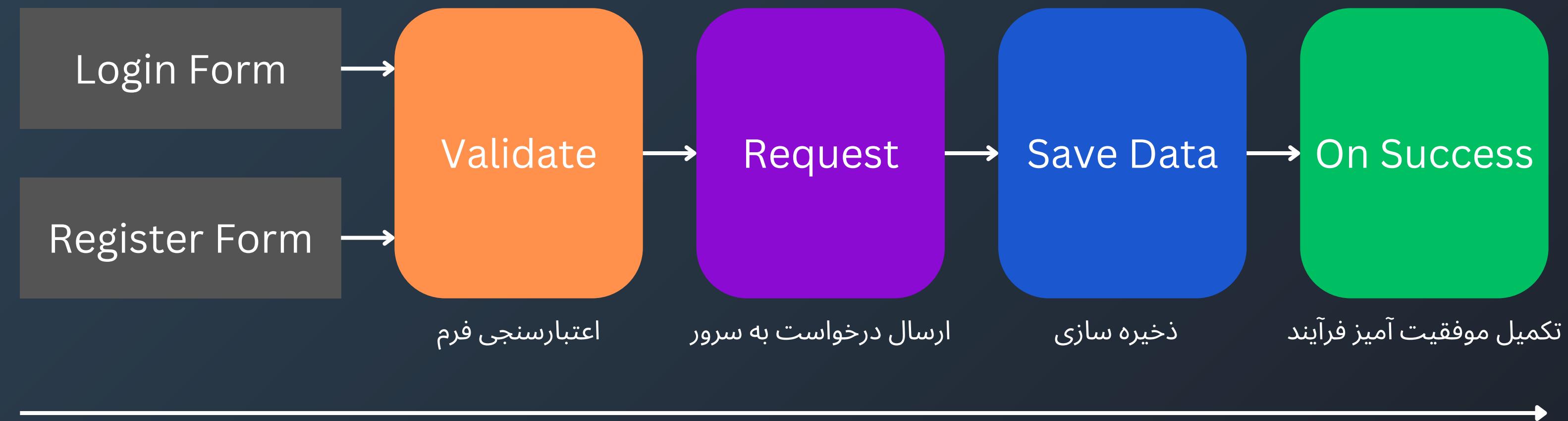
در این مثال قصد داریم یک کد یکپارچه و ساختار قابل اعتماد برای مدیریت این فرم‌ها فراهم کنیم.

در نهایت به سادگی و به دور از پیچیدگی در کامپوننت تنها با فراخوانی یک متد تمام مراحل برای هریک از فرم‌ها اجرایشوند.



مراحل فرم‌ها

در این بخش مراحلی که برای تمام فرم‌ها باید وجود داشته باشند را مشاهده می‌کنیم.



با اینکار اطمینان حاصل می‌شود که تمامی فرم‌های ما باید این مراحل را پیاده سازی و طی کنند.

بخش FormTemplate

در این بخش یک Class به صورت Abstract می سازیم، در اینجا الگو و قالب مراحلی که برای فرم ها لازم هستند را مشخص می کنیم و در نهایت تنها با فراخوانی `handleSubmit` تمامی مراحل برای فرم مورد نظر اجرا می شوند.



FormTemplate

```
export default abstract class FormTemplate<T> {
  public handleSubmit(data: T): void {
    this.validate(data);
    this.request(data);
    this.saveData(data);
    this.onSuccess();
  }

  protected abstract validate(data: T): void;
  protected abstract request(data: T): Promise<void>;
  protected abstract saveData(data: T): void;

  protected onSuccess(): void {
    console.log('Form submitted successfully!');
  }
}
```

در کلاس های فرزند حتما باید بازنویسی شوند ➔

در کلاس های فرزند به صورت اختیاری می توانند بازنویسی شوند ➔

ساخت Interface برای فرم‌ها

در این بخش برای فرم‌های خود Interface می‌سازیم.



Login

```
● ● ●  
export default interface ILogin {  
  username: string;  
  password: string;  
}
```



Register

```
● ● ●  
export default interface IRegister {  
  username: string;  
  email: string;  
  password: string;  
  confirmPassword: string;  
}
```

بخش LoginForm

در این بخش منطق اعتبارسنجی، فرآخوانی API و ذخیره سازی را برای فرم Login که اجباری هستند را پیاده سازی می کنیم.



```
import FormTemplate from '../form-template';
import Login from './login.interface';

export default class LoginForm extends FormTemplate<Login> {
    protected validate(data: Login): void {
        if (!data.username || !data.password) {
            throw new Error('Validation failed: Username and Password are required.');
        }
        console.log('Validation passed for LoginForm.');
    }

    protected async request(data: Login): Promise<void> {
        console.log('Call login API', data);
    }

    protected saveData(data: Login): void {
        console.log('Saving login data:', data);
    }
}
```

بخش RegisterForm

در این بخش منطق اعتبارسنجی، فرآخوانی API و ذخیره سازی را برای فرم Register که اجباری هستند را پیاده سازی می کنیم. همچنین `onSuccess` که اختیاری می باشد را نیز پیاده سازی کرده ایم.



RegisterForm

```
import FormTemplate from '../form-template';
import Register from './register.interface';

export default class RegisterForm extends FormTemplate<Register> {
  protected validate(data: Register): void {
    if (
      !data.username ||
      !data.email ||
      !data.password ||
      !data.confirmPassword
    ) {
      throw new Error(
        'Validation failed: Username, Email, Password, and Confirm Password are required.'
      );
    }
    if (data.password !== data.confirmPassword) {
      throw new Error('Passwords do not match.');
    }
    console.log('Validation passed for RegisterForm.');
  }

  protected async request(data: Register): Promise<void> {
    console.log('Call register API', data);
  }

  protected saveData(data: Record<string, any>): void {
    console.log('Saving register data:', data);
  }

  protected onSuccess(): void {
    console.log('Register successful!');
  }
}
```

بخش AuthForm

در این بخش Component مورد نظر خود را به شکل زیر می سازیم.



```
import LoginForm from './pattern/forms/login/login.form';
import RegisterForm from './pattern/forms/register/register.form';

const AuthForm: React.FC = () => {
  const handleLogin = () => {
    const loginForm = new LoginForm();
    try {
      loginForm.handleSubmit({ username: 'test', password: 'password123' });
    } catch (error) {
      console.error(error);
    }
  };

  const handleRegister = () => {
    const registerForm = new RegisterForm();
    try {
      registerForm.handleSubmit({
        username: 'test',
        email: 'test@example.com',
        password: 'password123',
        confirmPassword: 'password123',
      });
    } catch (error) {
      console.error(error);
    }
  };

  return (
    <div>
      <button onClick={handleLogin}>Submit Login Form</button>
      <button onClick={handleRegister}>Submit Register Form</button>
    </div>
  );
}

export default AuthForm;
```

توجه نمایید که به دلیل پیچیده نکردن این مثال از مواردی مانند ساخت بخش ال و فراخوانی API و... صرف نظر کرده ایم و برای مشاهده خروجی حتماً محیط Console مرورگر خود را مشاهده نمایید.

طرز استفاده از AuthForm

در پایان به شکل زیر از AuthForm استفاده می کنیم.



TemplateMethodDesignPattern

```
● ● ●  
import AuthForm from './AuthForm';  
  
const TemplateMethodDesignPattern = () => {  
  return (  
    <div>  
      <AuthForm />  
    </div>  
  );  
};  
  
export default TemplateMethodDesignPattern;
```