# JavaScript MV* Patterns

Summer@JS Patterns 讀書會

2024/03/05


# Summer

Front-End Developer

Summer。桑莫。夏天 cythilya.tw

打造高速網站，從網站指標開始！全方位提升使用者體驗與流量的關鍵

cythilya.dev cythilya cythilya@gmail.com

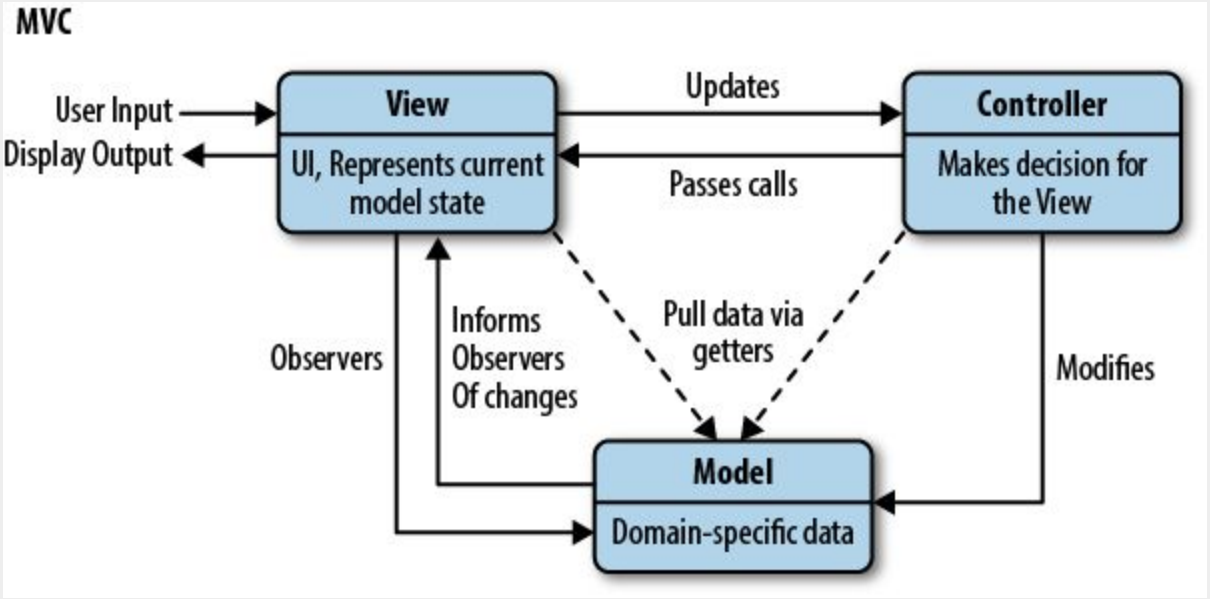
# 義大利麵程式碼

```
const num1 = prompt('請輸入第一個數字:');
const num2 = prompt('請輸入第二個數字:');
const operator = prompt('請輸入運算符號
(+, -, *, /):');

let result;

if (operator === '+') {
  result = Number(num1) + Number(num2);
} else if (operator === '-') {
  result = Number(num1) - Number(num2);
} else if (operator === '*') {
  result = Number(num1) * Number(num2);
} else if (operator === '/') {
  result = Number(num1) / Number(num2);
} else {
  console.log('無效的運算符號！');
}

console.log('結果:' + result);
```

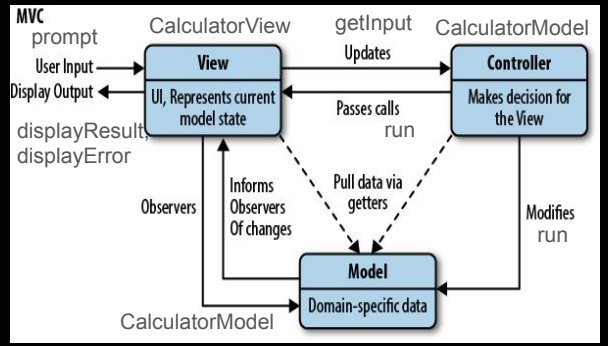
https://codepen.io/cythilya/pen/vYPVMPX

# 關注點分離


圖片來源：https://medium.com/@onuryanar/separation-of-concerns-eca786e75f5a


MVC
User Input
Display Output
View
UI, Represents current model state
Updates
Passes calls
Controller
Makes decision for the View
Observers
Informs Observers Of changes
Pull data via getters
Modifies
Model
Domain-specific data

```
class CalculatorModel {
  constructor() {
    this.num1 = null;
    this.num2 = null;
    this.operator = null;
    this.result = null;
  }
}
```




```
class CalculatorView {
  getInput() {
    const num1 = parseFloat(prompt('請輸入第一個數字:'));
    const num2 = parseFloat(prompt('請輸入第二個數字:'));
    const operator = prompt('請輸入運算符號 (+, -, *, /):');
    return { num1, num2, operator };
  }

  displayResult(result) {
    alert('結果:' + result);
  }

  displayError(error) {
    alert('錯誤:' + error.message);
  }
}
```

```
class CalculatorController {
  constructor(model, view) {
    this.model = model;
    this.view = view;
  }

  run() {
    // 略...
  }

  calculate() {
    // 略...
  }
}

// Main
const model = new CalculatorModel();
const view = new CalculatorView();
const controller = new
CalculatorController(model, view);
controller.run();
```

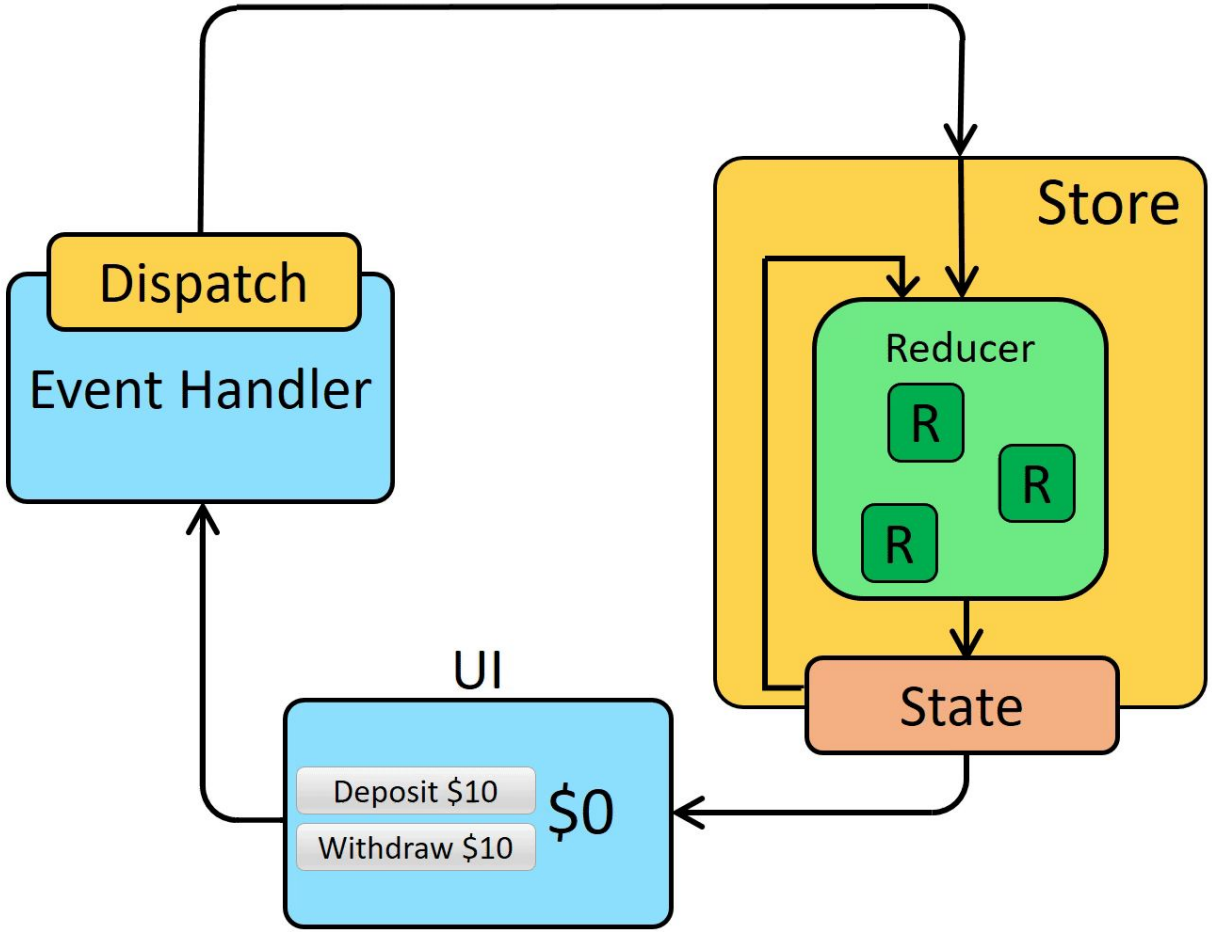
https://codepen.io/cythilya/pen/xxByevb

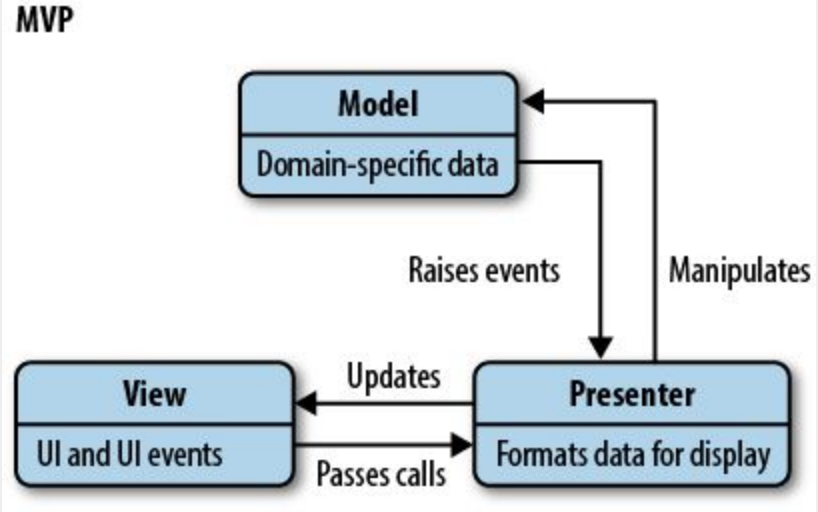
# React 元件中的 MVC

- M：state、context、Redux
- V：渲染與使用者互動的畫面、顯示資料
- C：元件的生命週期 lifecycle method 或 hook、Redux 中的 actionc、custom hook

```
const TodoList = () => {
  const [todos, setTodods] = useState([]);

  useEffect(() => {
    const fetchTodos = /* 略... */

    const fetchData = async () => {
      const todos = await fetchTodos();
      setTodods(todos);
    };
    fetchData();
  }, []);

  return (
    <div>
      <h2>Todos:</h2>
      {todos.map((item) => {
        return <div key={item}>{item}</div>;
      })}
    </div>
  );
};
```

# React App 中的 MVC




MVP
Model
Domain-specific data
Raises events
Manipulates
View
UI and UI events
Updates
Passes calls
Presenter
Formats data for display

```
// View
class CalculatorView {
  getInput() {
    const num1 =
parseFloat(prompt('請輸入第一個數字
:'));
    const num2 =
parseFloat(prompt('請輸入第二個數字
:'));
    const operator = prompt('請輸入
運算符號 (+, -, *, /):');
    return { num1, num2, operator
};
  }
  // 略...
}
```

https://codepen.io/cythilya/pen/poBzapM

以 MVP
改寫 MVC

```
// View
class CalculatorView {
  getFirstInput() ...

  getSecondInput() ...

  getOperator() ...
}

// Presenter
class CalculatorPresenter {
  // 邏輯都抽出來給 P
  run() {
      const num1 = this.view.getFirstInput();
      const num2 = this.view.getSecondInput();
      const operator = this.view.getOperator();
      this.model.num1 = num1;
      this.model.num2 = num2;
      this.model.operator = operator;
      this.calculate();

this.view.displayResult(this.model.result);

  }
}
```

```
// MVC
describe('Calculator', () => {
  it('should get 3 when perform addition by adding 1 and
2', () => {
    calculator.model.num1 = 1;
    calculator.model.num2 = 2;
    calculator.model.operator = '+';

    calculator.calculate();

    expect(calculator.model.result).toBe(3);
  });
});
```

改用 MVP
寫測試

```
// MVP
describe('CalculatorPresenter', () => {
  let model, view, presenter;

  it('should get 3 when adding 1 and 2', () => {
    view.getFirstInput.mockReturnValue(1);
    view.getSecondInput.mockReturnValue(2);
    view.getOperator.mockReturnValue('+');

    presenter.run();
    expect(view.displayResult).toHaveBeenCalledWith(3);
  });
});
```

# React 元件中的 MVP


```
const Calculator = () => {
  const [num1, setNum1] = useState('');
  const [num2, setNum2] = useState('');
  const [operator, setOperator] = useState('+');
  const [result, setResult] = useState('');

  const handleChange = (e) => // 略...

  const calculateResult = () => {
    switch (operator) {
      case '+':
        setResult(parsedNum1 + parsedNum2);
        break;
           // 略...
    }
  };
```

```
  return (
    <div>
      <input type="number" value={num1}
onChange={handleNum1Change}
      />
      <select value={operator}
onChange={handleOperatorChange} >
        <option value="+">+</option>
             // 略...
      </select>
      <input type="number" value={num2}
onChange={handleNum2Change} />
      <button
onClick={calculateResult}>Calculate</button>
      <div>Result: {result}</div>
    </div>
  );
};
```

https://playcode.io/1784127

# MVC 寫測試

```
describe('Calculator', () => {
  it('should get 15 when 5 multiplied by 3', () => {
    const { getByTestId } = render(<Calculator />);

    // 輸入數字
    fireEvent.change(getByTestId('number1'), { target: { value: '5' } });
    fireEvent.change(getByTestId('number2'), { target: { value: '3' } });

    // 選取運算符號
    fireEvent.change(getByTestId('operator'), { target: { value: '*' } });

    // 點擊計算按鈕
    fireEvent.click(getByTestId('calculate'));

    // 驗證結果
    expect(getByTestId('result')).toHaveTextContent('15');
  });
  });
```

```
const useCalculatorHook = () => {
  const [num1, setNum1] = useState(0);
  const [num2, setNum2] = useState(0);
  const [operator, setOperator] = useState('+');
  const [result, setResult] = useState(0);

  const handleNum1Change = (e) =>
setNum1(parseFloat(e.target.value));
  const handleNum2Change = (e) =>
setNum2(parseFloat(e.target.value));
  const handleOperatorChange = (e) =>
setOperator(e.target.value);

  const calculate = () => {
    switch (operator) {
      case '+':
        setResult(num1 + num2);
        break;
        // 略...
    }
  };

  return {
    // 略...
};
```

https://playcode.io/1781174

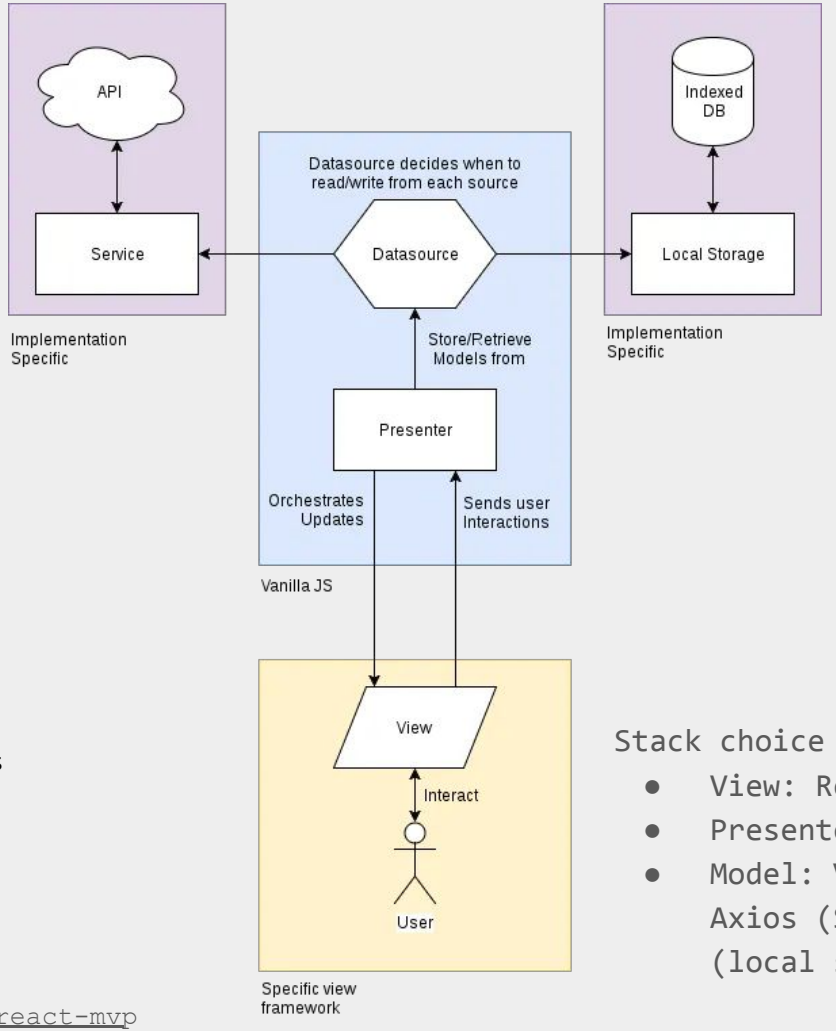
```
const Calculator = () => {
  const { calculate, handleChange, num1, num2, operator, result } =
    useCalculatorHook();

  return (
    <>
      <input
        data-test-id="number1"
        type="number"
        value={num1}
        onChange={handleChange}
      />
      <select data-test-id="operator" value={operator}
onChange={handleChange}>
        <option value="+">+</option>
        <!-- 略...-->
      </select>
      <input
        data-test-id="number2"
        type="number"
        value={num2}
        onChange={handleChange}
      />
      <button onClick={calculate}>Calculate</button>
      <div data-test-id="result">{result}</div>
    </>
  );
};
```

# MVP 寫測試

```
describe('useCalculatorHook', () => {
  test('should get 8 when add 5 and 3', () => {
    const TestComponent = () => {
      const {
        // 略 ...
      } = useCalculatorHook();

      return (
        <div>
          <input
            data-test-id="number1"
            value={num1}
            onChange={handleNum1Change}
          />
          <select
            data-test-id="operator"
            value={operator}
            onChange={handleOperatorChange}
          >
            <option value="+">+</option>
          </select>
          <input
            data-test-id="number2"
            value={num2}
            onChange={handleNum2Change}
          />
          <button data-test-id="calculate" onClick={calculate}>
            Calculate
          </button>
          <div data-test-id="result">{result}</div>
        </div>
      );
    };
```

```
 const { getByTestId } = render(<TestComponent
/>);

    // 輸入數字
    fireEvent.change(getByTestId('number1'), {
target: { value: '5' } });
    fireEvent.change(getByTestId('number2'), {
target: { value: '3' } });

    // 選取運算符號
    fireEvent.change(getByTestId('operator'), {
target: { value: '+' } });

    // 點擊計算按鈕
    fireEvent.click(getByTestId('calculate'));

    // 驗證結果

expect(getByTestId('result')).toHaveTextContent(
'8');
  });
});
```

# React 作為 V 的 MVP 專案架構




```
├── src
│   ├── components/login
│   │   ├── login-preseter.js
│   │   ├── login-view.jsx
│   │   └── ...
│   ├── core/datasource
│   │   └── ...
│   ├── util
│   │   ├── connect-component.js
│   │   └── ...
│   ├── view/layout
│   │   └── ...
│   └── ...
├── package.json
└── README.md
```

Stack choice

- View: React
- Presenter: Vanilla JS
- Model: Vanilla JS (datasource), Axios (Service classes), PouchDB (local storage)

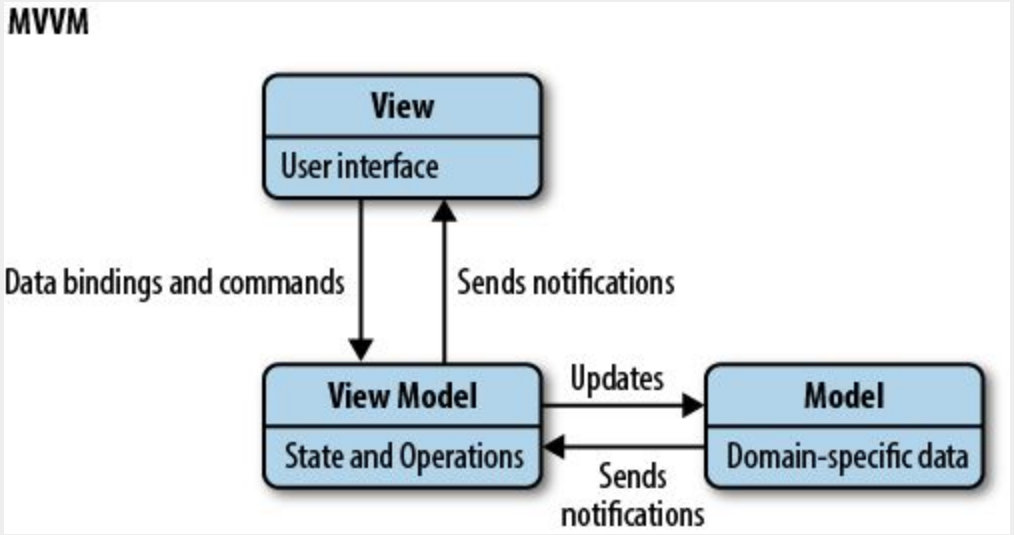
https://github.com/gabteles/react-mvp

```
class LoginView extends PureComponent {
  constructor(props) {
    super(props);

    this.presenter = props.presenter;
    this.presenter.setView(this);

    this.state = {
      email: '',
      password: '',
      loading: false,
      error: null,
    };
  }

  showProgress() // 略 ...

  hideProgress() // 略 ...

  render() {
    return (
      // 略 ...
    );
  }
};
```

```
class LoginPresenter {
  setView(view) {
    this.view = view;
  };

  async onLogin(email, password) {
    this.view.showProgress();
    const userDatasource = await UserDatasource.getInstance();
    const response = await userDatasource.signIn(email, password);
    this.view.hideProgress();

    if (response.success) {
      this.view.showLoginSuccess();
    } else if (response.fatal) {
      this.view.showLoginFatalError();
    } else {
      this.view.showLoginError(response.error);
    }
  }
}
```

```
const ConnectComponent = (view, presenter) => React.createClass({
  displayName: `ConnectComponent(${view.name}, ${presenter.name})`,
  render: function () {
    const _presenter = new presenter();
    return React.createElement(view, { ...this.props, presenter: _presenter })
  },
});
```

https://github.com/gabteles/react-mvp


MVVM
View
User interface
Data bindings and commands
Sends notifications
View Model
State and Operations
Updates
Model
Domain-specific data
Sends
notifications

```
// View
class CalculatorView {
  displayResult(result) // 略...

  // 更新
  update(data) // 略...

}

// ViewModel
class CalculatorViewModel {
  // 將 Model 資料轉換為 View 所需的格式
  get formattedData() // 略...

  run() {
    try {
      const num1 = this.view.getFirstInput();
      const num2 = this.view.getSecondInput();
      const operator = this.view.getOperator();
      this.model.num1 = num1;
      this.model.num2 = num2;
      this.model.operator = operator;
      this.calculate();
      this.view.displayResult(this.model.result);
    } }

  calculate() {
    // 略...
  }
}
```

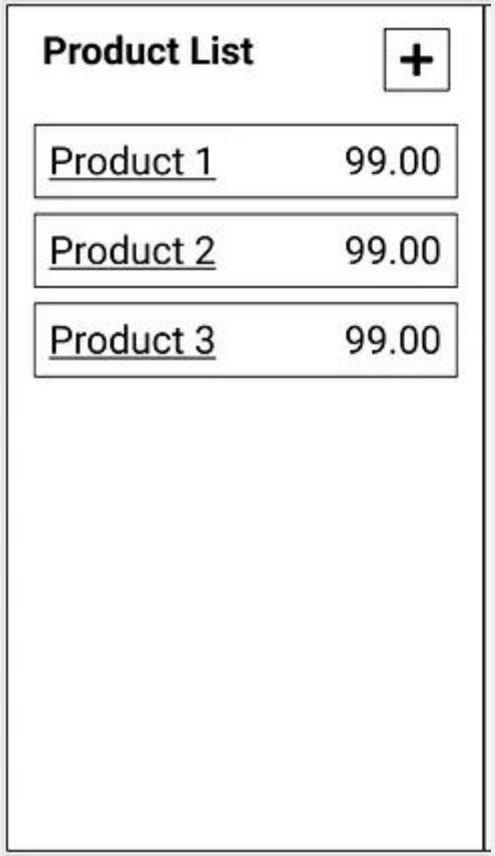
```
// Main
const model = new CalculatorModel();
const view = new CalculatorView();
const viewModel = new CalculatorViewModel(model, view);
viewModel.run();

// 當 ViewModel 的 formattedData 變更時，更新 View
Object.defineProperty(viewModel, 'formattedData', {
  get: function () {
    return `結果: ${this.model.result}`;
  },
  set: function (newValue) {
    view.update(newValue);
  },
});

// 初始化 View
view.update(viewModel.formattedData);
// 使用 Proxy 來監聽 model 的變化
const userProxy = new Proxy(model, {
  set: function (target, property, value) {
    target[property] = value;
    // 當 model 變化時，更新 view
    view.update(viewModel.formattedData);
    return true;
  },
});
```

# React 元件中的 MVVM

```
<CopyPopover
  content={
    <InputStyle
      ref={textRef}
      value={text}
      onClick={copyToClipboard}
      onChange={copyToClipboard}
      readOnly
    />
  }
  placement="top"
  popoverBody={<span>Copy Success!</span>}
  trigger="click"
/>
```

```
const useCopyToClipboar = () => {
  const textRef = useRef(null);
  const [copySuccess, setCopySuccess] = useState(false);

  const copyToClipboard = () => {
    try {
      textRef.current.select();
      document.execCommand('copy');
      setCopySuccess(true);
    } catch (error) {
      console.error(error);
      setCopySuccess(false);
    }
  };

  return {
    textRef,
    copySuccess,
    copyToClipboard,
  };
};
```

# React App 作為 V 的 MVVM 架構專案




```
├─ Data
└─ Presentation
   └─ View
      └─ Product
         ├─ List
         │  ├─ Components
         │  │  ├─ ProductList.js
         │  │  └─ AddButton.js
         │  ├─ View.js
         │  └─ ViewModel.js
         ├─ ...
         └─ index.js
```

Stack choice

- View: React
- ViewModel: React
- Model: Vanilla JS (data), local storage

https://github.com/nanosoftonline/clean-mvvm-react

```
const ProductListViewModel = ({
  GetProductsUseCase
}) => {
  const [error, setError] = useState('');
  const [products, setProducts] =
useState([]);

  async function getProducts() {
    const { result, error } = await
GetProductsUseCase.execute();
    setError(error);
    setProducts(result);
  }
  return {
    error,
    getProducts,
    products,
  };
};
```

```
const ProductList = () => {
  let navigate = useNavigate();
  const { products, getProducts } =
DI.resolve('ProductListViewModel');

  useEffect(() => {
    getProducts();
  }, []);

  return (
    <div className="page">
      <div>
        <h2>Product List</h2>
        <Button title={'New'} onClick={() =>
navigate(`/product/new`)} />
      </div>
      <List data={products} />
    </div>
  );
}
```


一刀兩斷斬
你我就此恩斷義絕