УДК 003.26

Н.А. Жиляк, доц., канд. техн. наук
(БГТУ, г. Минск)

НЕКОТОРЫЕ ПРИЕМЫ ПРИ ИСПОЛЬЗОВАНИИ БИБЛИОТЕКИ REACTJS

В публикации речь пойдет о наиболее значимых этапах и приемах разработки нового проекта с использованием библиотеки ReactJS.

Перед началом работы программиста над проектом необходимо ознакомится с техническим заданием. Просмотреть package.json, который, как правило, должен содержать в себе информацию о вашем приложении: название, версия, зависимости, скрипты и конфигурацию и тому подобное. Любая директория, в которой есть этот файл, интерпретируется как Node.js-пакет. Далее необходимо отобразить дерево проекта на бумажном носителе/планшете и т.д. или использовать Developer Tools. Таким образом, визуализируем состояние. React Developer Tools – это дополнение к браузеру Google Chrome от компании Фэйсбук, которое служит очень полезным инструментом при работе с React. Дополнение создает еще одну вкладку (React) в Инструментах разработчика (Ctrl+Shift+I), позволяющую произвести инспекцию всего реакт-приложения, включая компоненты, а также их свойства и состояния (state, props).

Затем начинается сам процесс разработки при котором необходимо выполнить следующие действия и следующим образом:

– быстрый переход к компоненту или функции: клик с зажатой клавишей CMD прямо в JSX (VSCode);

– быстрый переход к родителю: CMD+SHIFT+F (глобальный поиск по проекту в VSCode). Искать <Название_компонента;

– быстрый просмотр списка родителей: React dev tools;

– создать чеклист возможных состояний каждого компонента (ошибка, нет данных и т. д. – полный список);

– для дебаггинга использовать debugger. Также полезен метод console.assert;

– работать с фиктивными данными и фиктивным API (json-server, json-graphql-server);

– примененить одни и тех же фиктивные данные для Storybook, тестов и фиктивного API;

– передать в функцию объект и использовать деструктурирование для получения именованных параметров. Деструктурировать можно прямо в сигнатуре функции, это позволяет сразу же документировать ожидаемые параметры;

– Storybook driven development – создать и протестировать каждый компонент в отдельности.

Также при разработке важно обращать на моменты, связанные с оформлением проекта, например, соблюдать определенные отступы при оформлении текста, обеспечивать адаптивный дизайн. Можно использовать только выражения, которые что-то возвращают. Для сложной логики есть несколько вариантов:

- ранний return (удобно для лоадеров и ошибок);
- выделение логики рендера в отдельную функцию, где можно использовать все возможности Javascript (if/else/switch).

Производительность играет также немаловажную роль. Разработчик может позволить себе некоторые вольности в проекте, т.е. делать то, что удобно. Чаще всего это достаточно хорошо работает. Для этого удобно применять инлайн-функции. Также можно не беспокоится о рендере. Например, элемент описывает то, что вы хотите увидеть на экране: const elem = <h1>Hello. World </p1>.

В отличие от DOM-элементов, элементы React – это простые объекты, не отнимающие много ресурсов. React DOM обновляет DOM, чтобы он соответствовал переданным React-элементам.

Допустим, в вашем HTML-файле есть : . Мы назовём его «корневым» узлом DOM, так как React DOM будет управлять его содержимым.

Обычно в приложениях, написанных полностью на React, есть только один корневой элемент. При встраивании React в существующее приложение вы можете рендерить во столько независимых корневых элементов, во сколько посчитаете нужным. Также необходимо помнить, что рендер компонента – не равно изменению DOM-дерева. React работает с виртуальным DOM. Процесс применения изменений выглядит так: рендер (render) -> согласование (reconcile) -> применение (commit). Если DOM не изменяется, то и проблем с производительностью не будет, поэтому перестань волноваться о перерендерах. React достаточно умный, чтобы изменять только то, что требует изменения.

Для рендеринга React-элемента в корневой узел DOM необходимо вызвать ReactDOM.render() с React-элементом и корневым DOM-узлом в качестве аргументов:

Производительность контекста будет нормальной в том случае, если он будет меняться крайне редко. В любом случае необходимо тестирование для того, чтобы добиться идеальной производительности.

Также не рекомендуется использовать везде, где можно, useMemo, shouldComponentUpdate, PureComponent. Только если это действительно необходимо. У них есть накладные расходы, так как совершаются дополнительные действия. Если бы они были быстрыми, то использовались бы по умолчанию.

При управлении состоянием необходимо «держать» его так низко, как это только возможно. При необходимости поднимать. Также избегать хранить состояние, которое может быть извлечено из другого состояния или пропсов. При работе с объектами обращаться к ним или ссылаться лучше по идентификатору, вместо того, чтобы дублировать их.

Для разрешения конфликтов именования состояния необходимо использовать _myVar конвенцию. Нельзя синхронизировать состояния, необходимо извлекать. Например, получай полное имя (full name) путем объединения имени (firstName) и фамилии (lastName) прямо в методе render.

Состояния, которые изменяются вместе, должны храниться вместе. В этом помогают редьюсеры. Группируй, используй useState. Рассмотрим движки состояний – они описывают валидные состояния, что делает невалидные состояния невозможными (например, новый клиент, у которого уже есть пять прошлых покупок, или админ без прав). Если разделить состояния, они могут рассинхронизироваться.

Возможно при работе с проектом не нужен Redux. Пробрасывание пропсов не так страшно, как его описывают. Необходимо сохранять названия пропсов, использовать спред-оператор, передавать «детей» целиком, используй мемоизацию. Контекст и useReducer могут справиться практически с любой задачей.

Необходимо вызывать setLoading(false) в блоке finally.

Сделать все пропсы обязательными. Деструктурировать пропсы в сигнатуре функции, чтобы уменьшить количество кода. Это также полезно для обработчиков событий. Но что насчет пропсов, в имени которых есть дефис типа aria-label. Просто использовать синтаксис …otherProps. Сделать пропсы максимально специфичными.

Использовать стандартизированное наименование. onX для пропсов-обработчиков событий, handleX – для функций. Хранить propTypes в одном месте. Документировать propTypes с помощью JSDoc-комментариев, они будут выводиться в автокомплите редактора. Можно даже использовать markdown. Чтобы уменьшить проблему пробрасывания пропсов, необходимо использовать спред-оператор или передавать дочерние элементы. Само существование пропса предполагает, что он равен true. Поэтому достаточно написать <Input required />.

Держись ближе к нативному API. В обработчики передавай событие целиком, а не только значение инпута. Тогда ты сможешь использовать единый обработчик изменений. Придерживайся нативных имен (onBlur, onChange). Стремись к максимальной гибкости и минимальной кривой обучения.

Что касается стилей в React, то рекомендуется сочетать разные подходы:

– инлайновые стили для динамически изменяющихся свойств;
– пространства имен (CSS-модули);
– Sass для глобальных стилей;
– используй классы, чтобы применять несколько стилей сразу;
– используй flexbox или CSS Grid вместо float;
– создавай абстракции для брейкпоинтов (например, используй bootstrap).

Для простого переиспользования компонентов существуют три ключевых правила. Необходимо подумать о том, чтобы выделить на это отдельного человека/команду. На что это влияет и для чего это необходимо: скорость работы, быстрее и проще принимать решения, меньше бандлы (Bundle (англ. Пучок) – это совокупность каких-либо программных данных (файлов), объеденных по какому-либо признаку).

Слаженность в интерфейсе ведет к хорошему пользовательскому опыту. Меньше кода – меньше багов. В процессе тестирования необходимо учитывать следующие факторы:

– RTL лучше Enzyme. Чем проще API, тем лучше. Поощряет a11y. Проще отлаживать. Можно использовать те же запросы для Cypress;
– JSDOM не рендерит, поэтому не может тестировать адаптивный дизайн. Используй Cypress для теста адаптивного поведения;
– избегай снэпшот-тестов Jest. Вместо этого используй Percy или Chromatic для визуальных тестов;
– используй шаблон выбора сценариев для запуска приложения с различными данными. Автоматизировать эти тесты можно с помощью Cypress/Selenium;
– при использовании Cypress Cy-селекторы совпадают с RTL-селекторам – не нужно менять код для поддержки тестов Cypress;
– Cypress driven development – TDD для интеграционных тестов. Используй Cypress, чтобы перейти к месту, которое нужно проверить. Используй cy.only чтобы запустить один тест. В первый раз он должен упасть, это пройдет.

В данном материале описаны лишь некоторые рекомендации для работы с библиотекой ReactJS, которые направлены на оптимизацию работы и облегчению труда команды разработчиков. Т.е. применив на практике описанные приемы, и взяв их за правило, можно добиться увеличения скорости и качества исполнения поставленных задач.