Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Занятие 13

Структура проекта. Rendering Компоненты

План занятия

ПЕРЕДОВАЯ
ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

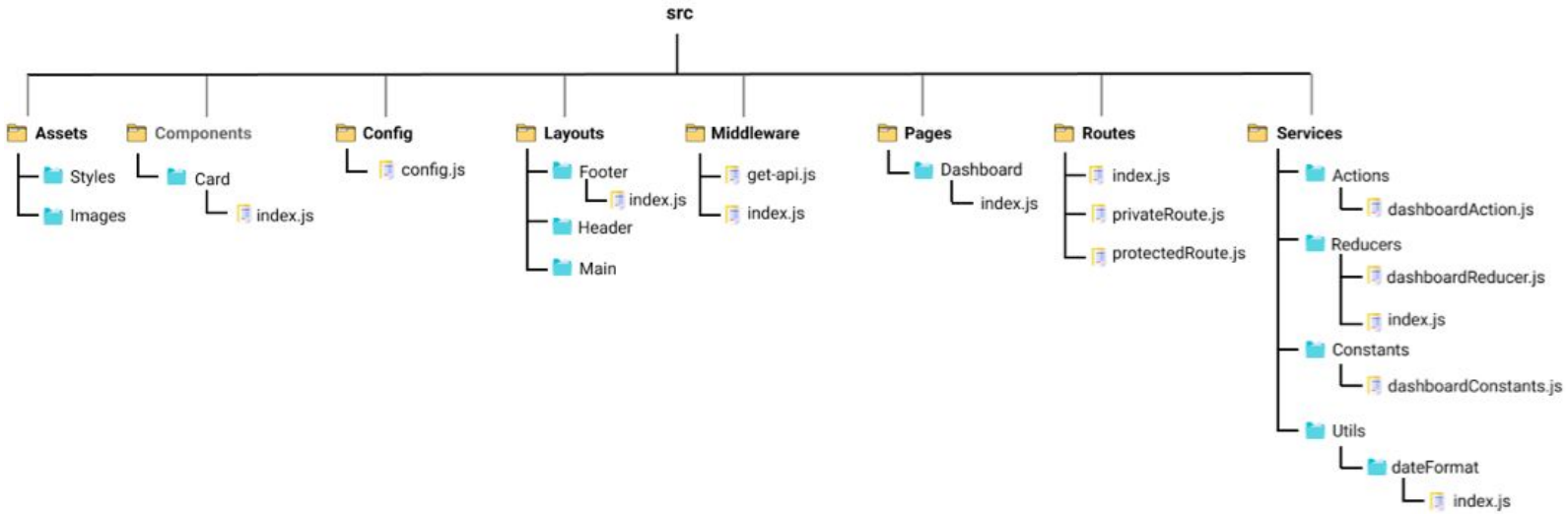
1. Структура проекта
2. Rendering
3. Компоненты

Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Структурирование больших приложений на React по папкам и файлам достаточно сложная задача

Определенного и четкого подхода нет.



Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

```
my-app/
├ README.md
├ ...
├ src/
|  ├ components/
|  |  ├ Button.js
|  |  ├ Button.css
|  |  ├ Button.test.js
|  |  ├ Home.js
|  |  ├ Home.css
|  |  └ Home.test.js
|  ├ hooks/
|  |  ├ useApi.js
|  |  └ ...
|  ├ App.js
|  ├ index.css
|  └ ...
└ ...
```

Структура для малых проектов

Отличный подход для проектов с 10-20 компонентами, таких как отдельная страница портфолио или целевая страница меньшего размера.

Хорошей практикой является создание папки components/ и hooks/ в папке src/

Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

components/

Папка содержит все компоненты приложения. Однако, поскольку проект выходит за рамки 20 компонентов, эта папка может стать неорганизованной, и управлять ею будет сложно.

В сложных структурах папок компоненты распределены по нескольким папкам и организованы структурированным образом.

hooks/

Папка содержит пользовательские хуки, используемые в проекте. Эта папка полезна при любом размере проекта, поскольку пользовательские перехваты обычно используются во многих местах по всей кодовой базе. Централизованное расположение всех пользовательских хуков может упростить их поиск и управление ими.

Структура проекта

ПЕРЕДОВАЯ
ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Преимущества

Наиболее значительным преимуществом этой структуры папок является ее простота. Она проста и понятна, что делает ее отличным выбором для небольших проектов.

Структура проекта

Недостатки

Одним из основных недостатков этой структуры папок является то, что она не предоставляет руководства по обработке файлов, таких как изображения, служебные функции или контексты React.

Это связано с тем, что в небольших проектах обычно не так много файлов такого типа, поэтому достаточно хранить их в корне папки src. Однако по мере роста проекта он может быстро стать неорганизованным и трудным в управлении. Вот почему для любого проекта, выходящего за рамки небольшого размера, рекомендуется использовать по крайней мере промежуточную структуру папок, чтобы все было организовано и легко находилось.

Начиная работать над проектом, состоящим менее чем из 20 компонентов, крайне важно организовать ваши файлы простым и понятный образом. Это может упростить понимание кодовой базы и навигацию по ней, а также поможет избежать потенциальных проблем с обслуживанием и масштабируемостью. Однако по мере роста размера и сложности проекта, возможно, придется рассмотреть возможность использования промежуточных организационных структур.

Структура проекта


```
my-app/
├ README.md
├ ...
├ src/
|  ├ assets/
|  ├ components/
|  |  ├ Form/ // ui folder for form elements according to your design system
|  |  |  ├ Button.js
|  |  |  ├ Button.css
|  |  |  ├ Button.test.js
|  |  |  └ ...
|  |  ├ Map/ // custom component for integration of Maps Provider, like Google
|  |  |  └ ...
|  |  ├ LoginForm/ // module that is used for the login page
|  |  |  ├ index.js
|  |  |  ├ LoginForm.js
|  |  |  ├ useLogin.js
|  |  |  └ ...
|  ├ context/ // Sometimes called store, redux or similar
|  ├ data/
|  ├ hooks/
|  ├ layouts/
|  ├ pages/
|  |  |  ├ Home/
|  |  |  |  └ index.js
|  |  |  ├ Login/
|  |  |  |  └ index.js
|  |  |  ├ SignUp/
|  |  |  └ ...
|  ├ services/ // sometimes called apis
|  ├ utils/
|  ├ App.js
|  ├ index.js
|  └ ...
└ ...
```

Промежуточная структура папок для небольших стартапов

Отлично подходит для крупных проектов, над которыми будут работать несколько разработчиков. У каждой важной части вашего проекта есть своя папка, что улучшает одновременную работу с приложением. Страницы теперь разделены.

Наиболее заметным изменением по сравнению с простой структурой папок является то, что разделяем проект на страницы и дополнительные подкомпоненты, где каждая страница содержит всю логику, связанную с конкретной страницей, в одном файле, вместо того, чтобы искать по нескольким папкам и несвязанным файлам. Кроме того, создается все больше и больше повторно используемых компонентов.

Структура проекта

При такой структуре лучше избегать размещения каких-либо файлов в корне папки src, за исключением файлов типа index.js.

assets/
Папка содержит все файлы, не связанные с кодом, такие как изображения, глобальные файлы CSS, файлы шрифтов и другие, которые являются важными компонентами проекта. Эта папка служит центральным местом для хранения всех таких файлов.

components/
Папка в этом примере структурирована на вложенные папки, что является существенным изменением. Этот подход полезен для того, чтобы компоненты были организованы в отдельные разделы, вместо того чтобы быть сведенными в единую массу.

Например, в Form/ папке хранятся все компоненты, относящиеся к форме, такие как кнопки, флажки, элементы выбора, ..., в то время как Map/ папка содержит настроенные компоненты внешних библиотек, таких как Google Maps.

Структура проекта

Кроме того, более важные функции приложения хранятся в виде папки.

Например, LoginForm/ с собственными пользовательскими хуками и компонентами. Компоненты LoginForm/ затем могут импортировать более общие компоненты из других папок, например кнопку из Form/.

Слишком большое количество папок часто приводит к появлению хаоса, например дублирования кода, потому что другие команды просто не знают, что этот компонент уже существует в другом месте.

Хороший совет - не делайте папки размером более 8 элементов для каждого уровня структуры. В противном случае людям будет трудно запомнить, что они прочитали в первый раз.

Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

context/

Папка предназначена для всех файлов контекста React, которые используются на нескольких страницах проекта.
В крупных проектах, будет несколько контекстов, которые будете использовать в своем приложении, что делает эту папку бесценным ресурсом.
Однако, если используете отдельное глобальное хранилище данных, такое как Redux, можете заменить эту папку более подходящим набором папок, специально предназначенных для хранения ваших файлов Redux.

data/

Папка предназначена для хранения ресурсов данных, таких как файлы JSON, которые содержат информацию, используемую в коде, например, элементы хранилища или информацию о теме.

Кроме того, в этой папке также может храниться файл, содержащий глобальные постоянные переменные. Это полезно, когда у вас есть множество констант, которые вы используете в своем приложении, например переменные среды.

Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

hooks/

Папка остается неизменной по сравнению с простой структурой папок. Однако в этой расширенной структуре будут храниться только глобальные хуки, которые используются на нескольких страницах, в отличие от хранения каждого хука глобально в простом приложении. Некоторые из хуков будут перемещены в конкретные компоненты в папке components.

layout/

Папка используется для определения структуры макета страниц в приложении.

Макеты - это скелет страницы, и они содержат общие элементы, которые являются общими для нескольких страниц, такие как верхние и нижние колонтитулы, меню навигации и так далее. Определяя макет страницы по умолчанию, можно избежать повторения одного и того же кода на всех страницах и обеспечить согласованность во всем приложении.

Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

pages/
Папка для каждой страницы вашего приложения. В каждой папке страницы должен быть один корневой файл (обычно index.js).

Такое отделение кода, специфичного для конкретной страницы, от более общего глобального кода является основным преимуществом этой системы по сравнению с простой структурой папок. При разделении страниц легче понять основную функциональность приложения.

services/
Папка связана с взаимодействием с внешними API или сервисами.
Сервисы обычно используются для абстрагирования деталей взаимодействия с внешними API или сервисами от остального кода приложения, упрощая его обслуживание и тестирование. В папке код, который обрабатывает такие задачи, как аутентификация, выборка данных и преобразование данных.

Структура проекта

utils/

Папка, которая предназначена для хранения всех служебных функций, таких как средства форматирования. Папка довольно проста, и все файлы, содержащиеся в ней, должны быть такими же. Если используете utils/ папку, рекомендуется разбивать код на более мелкие и специфические подпапки, такие как stringUtils, mathUtils или dateUtils, чтобы упростить поиск и понимание кода. Также важно регулярно просматривать и очищать utils/папку, чтобы удалить любой ненужный или дублированный код.

Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Преимущества

Основное преимущество этой системы заключается в том, что у каждой части проекта есть своя собственная папка, что гарантирует, что в корневой src/ папке будет очень мало файлов. Еще одним существенным преимуществом является то, что на страницах теперь отображаются основные функциональные возможности страницы, и повторно использование компонентов.

По мере расширения проекта становится все более важным хранить используемые файлы в одном месте, поскольку это упрощает понимание, написание и чтение кода за счет уменьшения объема глобального кода, хранящегося в общих компонентах, перехватах и других папках.

Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Недостатки

Основным недостатком этой системы является то, что по мере того, как приложение будет становиться все больше, components/ папка станет раздутой, и ее будет трудно расширить. Это связано с тем фактом, что по мере расширения приложения, включающего все больше страниц, одна функция часто используется на нескольких страницах вместо одной. В таких случаях нужно переместить код в другие папки приложения и реорганизовать функциональный компонент, чтобы его можно было использовать на нескольких страницах, что начинает приводить к беспорядку. Типичные крупные проекты могут содержать сотни пользовательских компонентов.

По мере роста проекта почти весь код будет размещаться на нескольких страницах, что потребует использования расширенной структуры папок.

Структура проекта


```
my-app/
├ README.md
├ ...
├ src/
|  ├ assets/
|  ├ packages/ // earlier components/ folder. Sometimes called lib
|  |  ├ UI/
|  |  |  ├ components/
|  |  |  |  ├ Form/
|  |  |  |  |  ├ Button.js
|  |  |  |  |  ├ Button.css
|  |  |  |  |  ├ Button.test.js
|  |  |  |  |  └ ...
|  |  |  |  └ DataDisplay/
|  |  |  |  |  ├ Card.js
|  |  |  |  |  └ ...
|  |  |  ├ services/
|  |  |  ├ hooks/
|  |  |  ├ context/
|  |  |  ├ package.json
|  |  |  └ ...
|  |  ├ Map/ // external library customized and imported as a package
|  |  └ ...
|  ├ context/
|  ├ data/
|  ├ modules/ // sometimes called plugins or features
|  |  ├ LoginForm/
|  |  |  ├ index.js
|  |  |  ├ components
|  |  |  |  ├ LoginForm.js
|  |  |  ├ hooks/
|  |  |  |  ├ useLogin.js
|  |  |  |  |  └ ...
|  |  |  ├ context/
|  |  |  ├ services/
|  |  |  ├ ...
|  |  |  └ package.json
|  |  └ ...
|  ├ hooks/
|  ├ layouts/
|  ├ pages/
|  |  ├ Home/
|  |  |  └ index.js
|  |  ├ Login/
|  |  |  └ index.js
|  |  ├ SignUp/
|  |  |  └ ...
|  ├ services/
|  ├ utils/
|  ├ App.js
|  ├ index.js
|  └ ...
└ ...
```

Расширенная структура папок для более крупных проектов

Для продуктов корпоративного масштаба с несколькими отделами и кросс-функциональными командами.

Папки packages/ и modules/ - это соглашение, которое некоторые проекты React используют для организации кода в повторно используемые пакеты или библиотеки, что делает кодовую базу более модульной, поддерживаемой и доступной для совместного использования.

packages/ и modules/ обеспечить одновременную работу многих отделов.

В то же время повышается сложность развертывания, управления версиями, адаптации разработчиков и общего управления.

Структура проекта

Есть заметное различие, которое заключается в замене папки components/ на packages/ и добавлении modules/ папки.

modules/
Папка обеспечивает более усовершенствованный подход к организации аналогичного кода и, в отличие от components/ папки в структуре промежуточных папок, не сталкивается с проблемами значительного перекрытия между модулями, поскольку маловероятно, что ваш модуль будет иметь значительное перекрытие.

Структура проекта

packages/
Папка - это не стандартная часть проекта React, а скорее соглашение, которое некоторые проекты используют для организации кода в повторно используемые пакеты или библиотеки.

Пакеты - это, по сути, независимые модули кода, которые можно импортировать и использовать в других частях приложения. Организуя код в пакеты, можно создать более модульную и повторно используемую кодовую базу, что упростит обслуживание и обновление приложения с течением времени.

В packages/папке можно найти несколько вложенных папок, каждая из которых содержит свой собственный исходный код, документацию, хуки, сервисы, тесты и другие связанные файлы.

Использование пакетов таким образом также может упростить совместное использование кода в нескольких проектах или командах. Если есть набор утилит или компонентов, которые обычно используются в разных проектах, можно упаковать их и предоставить общий доступ к ним через частный или общедоступный реестр пакетов, например npm.

Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

modules/
Каждый модуль может представлять отдельную функцию или компонент приложения, например, аутентификацию пользователя, функциональность поиска или корзину покупок.

Модули могут содержать несколько файлов, таких как компоненты, хуки, служебные функции, стили и тесты, все относящиеся к одной и той же функции или компоненту. Разбивая код на модули, можно создать более модульную и обслуживаемую кодовую базу, что упростит анализ и обновление приложения с течением времени.

Использование модулей также может упростить совместное использование кода в разных частях приложения или с другими разработчиками. Если есть модуль, который используется в нескольких местах приложения, можно просто импортировать его там, где это необходимо, вместо того, чтобы дублировать код в нескольких местах.

Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

services/

В этой структуре, большинство сервисов теперь являются частью либо пакета, либо модуля. Могут отличаться от проекта к проету.

utils/

Папка в проекте React не является изначально плохой практикой, поскольку может быть полезным способом организации небольших повторно используемых функций и утилит, которые используются во всем проекте.

Однако в крупномасштабном проекте React папка utils/ может стать свалкой для несвязанного или плохо организованного кода, что может привести к проблемам с ремонтопригодностью, удобочитаемостью и раздуванием кода.

Структура проекта

Преимущества

Используя папку packages/ и modules/, можно разделять связанный код на более мелкие фрагменты, что упрощает поиск и обновление определенных частей приложения.

Когда код организован в packages/ и modules/, компоненты и логику можно легко повторно использовать во всем приложении. Например, может быть модуль, который обрабатывает аутентификацию пользователя, который может использоваться в нескольких местах по всему приложению.

Когда несколько разработчиков работают над крупномасштабным проектом, может быть сложно поддерживать согласованность кода и избегать конфликтов. Используя пакеты и модули, можно упростить разработчикам независимую работу над различными частями кодовой базы, не мешая друг другу.

Поскольку проект растет и развивается с течением времени, поддерживать хорошо организованную кодовую базу может быть непросто.

Разбиение приложения на более мелкие модульные части может упростить организацию и обслуживание кодовой базы с течением времени для нескольких команд в более крупной организации.

Структура проекта

Недостатки

Организация кода в меньшие пакеты и модули может повысить уровень сложности проекта, что затруднит его понимание и навигацию, особенно для новых членов команды.

Разбиение более специфичных пакетов и модулей на части может быть чрезмерной разработкой, особенно для небольших проектов. Это может привести к ненужному раздуванию кода и усложнению.

Поскольку проект растет и меняется с течением времени, поддерживать организацию и структуру кодовой базы может быть непросто, что может привести к путанице и ошибкам.

Без надлежащего управления может легко создать дублирующуюся функциональность в нескольких пакетах или модулях, что может привести к проблемам с обслуживанием и ошибкам.

В целом, решение об использовании packages/ и modules/ папок в крупномасштабном проекте должно основываться на конкретных потребностях проекта и команды. Хотя усовершенствованная структура может предложить преимущества, особенно для нескольких кросс-функциональных команд, важно тщательно рассмотреть потенциальные недостатки и убедиться, что кодовая база проекта остается управляемой и поддерживаемой с течением времени.

Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Выводы

В целом, решение какую использовать структуру для вашего проекта должно основываться на конкретных потребностях проекта и команды. Можно настроить эти папки в любой части так, как вам больше подходит.

В конечном счете, главное - найти организационную структуру, которая наилучшим образом подходит для вашего проекта и команды.

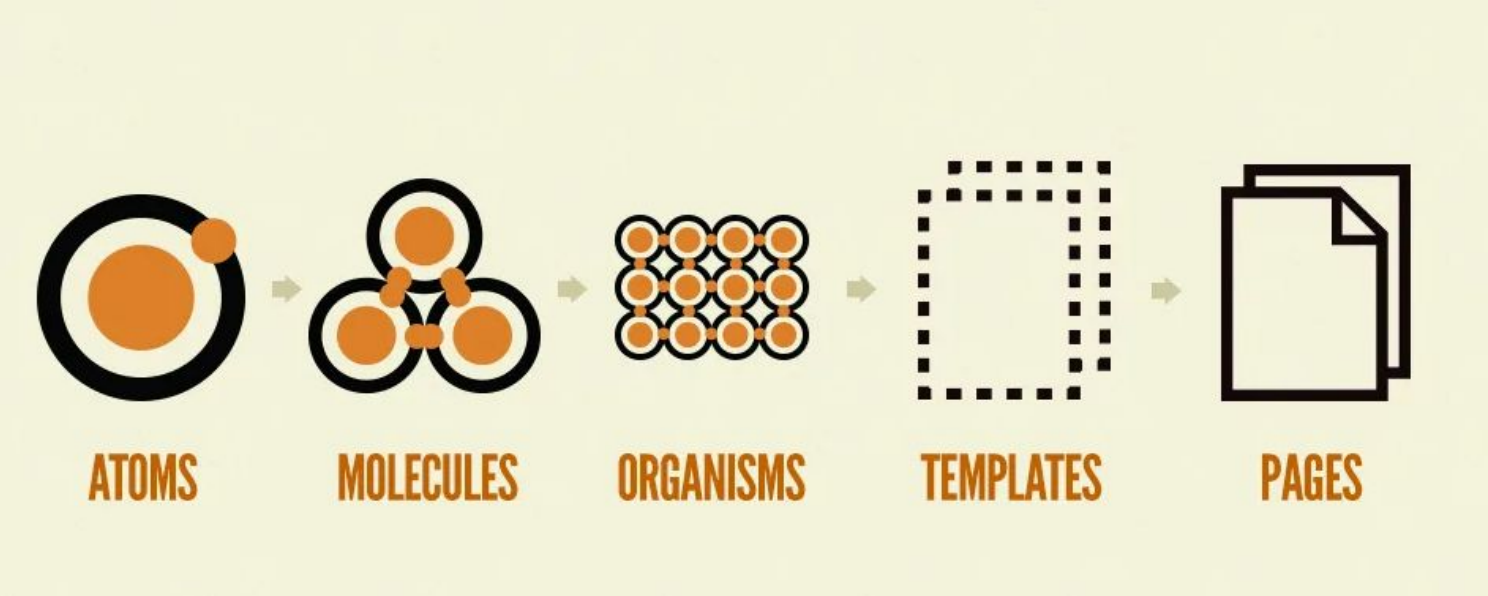
Несколько подходов react:

- Атомарный дизайн Брэда Фроста
- Monorepo и инструменты, такие как Turborepo
- Микрофронты и Микросервисы

Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

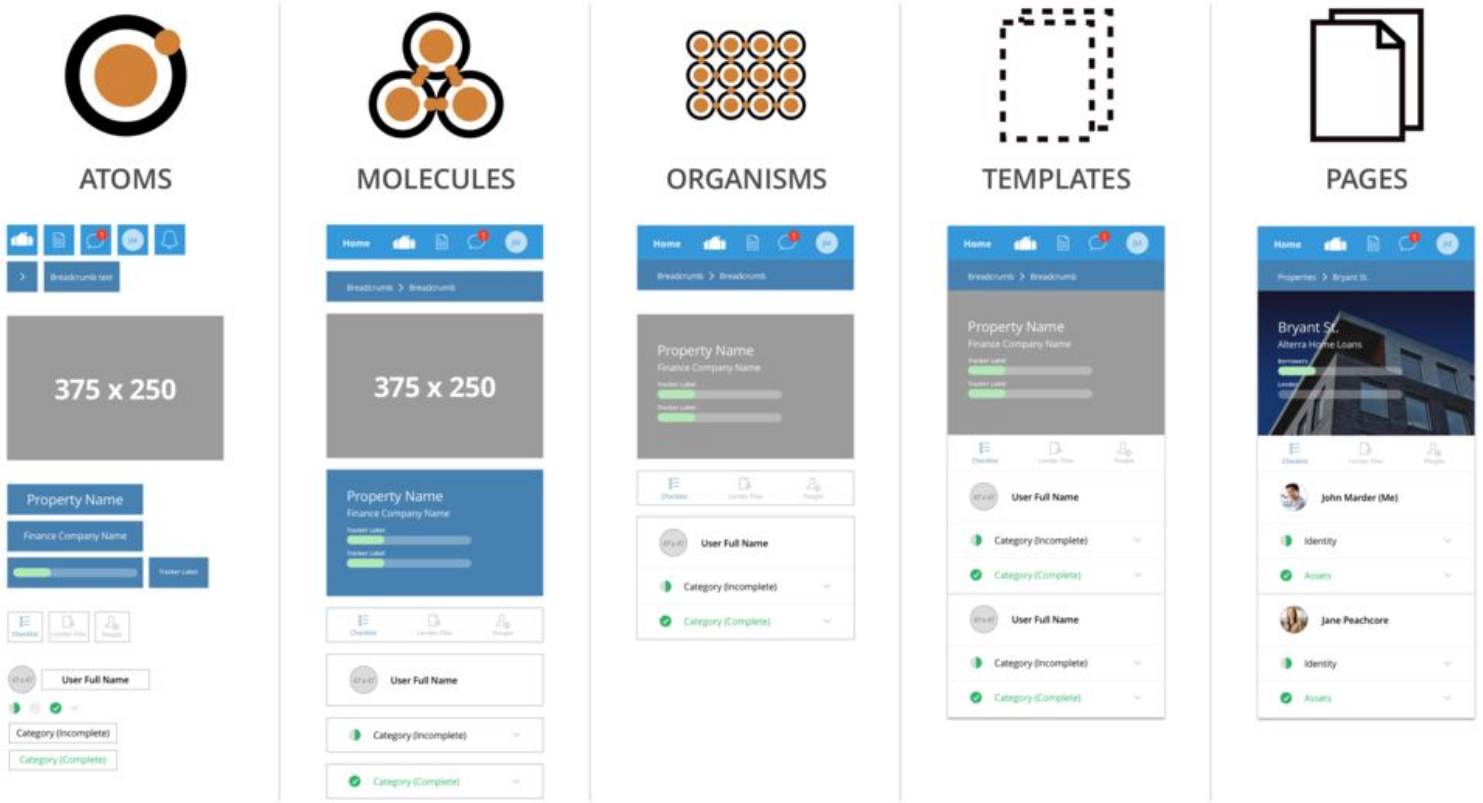
Атомарный дизайн Брэда Фроста



Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

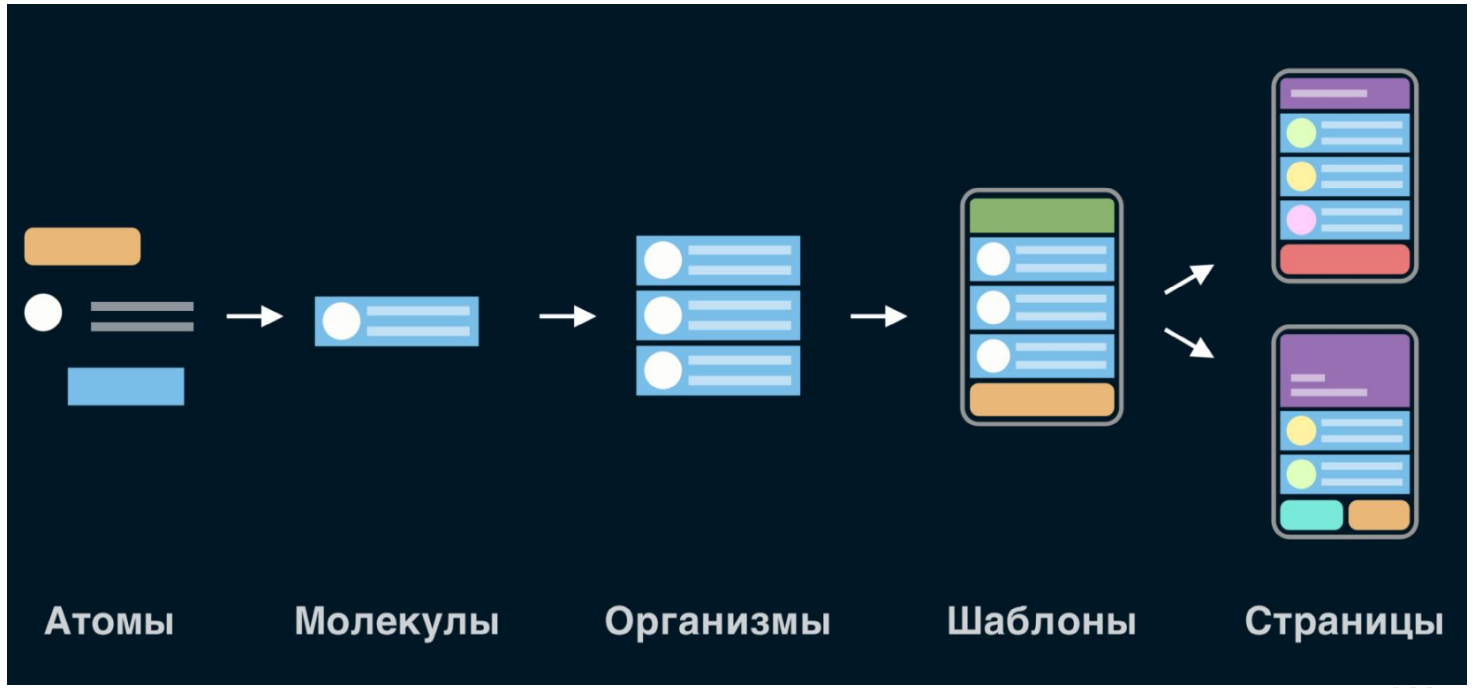
Атомарный дизайн Брэда Фроста



Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

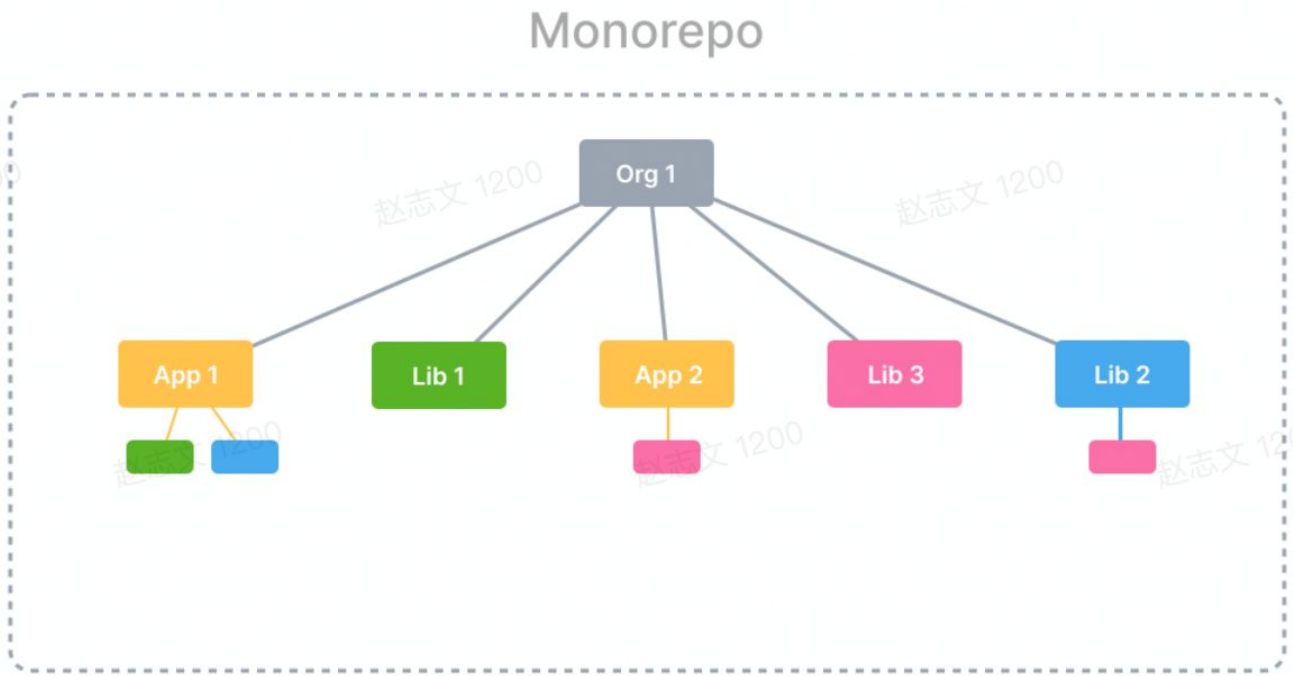
Атомарный дизайн Брэда Фроста



Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

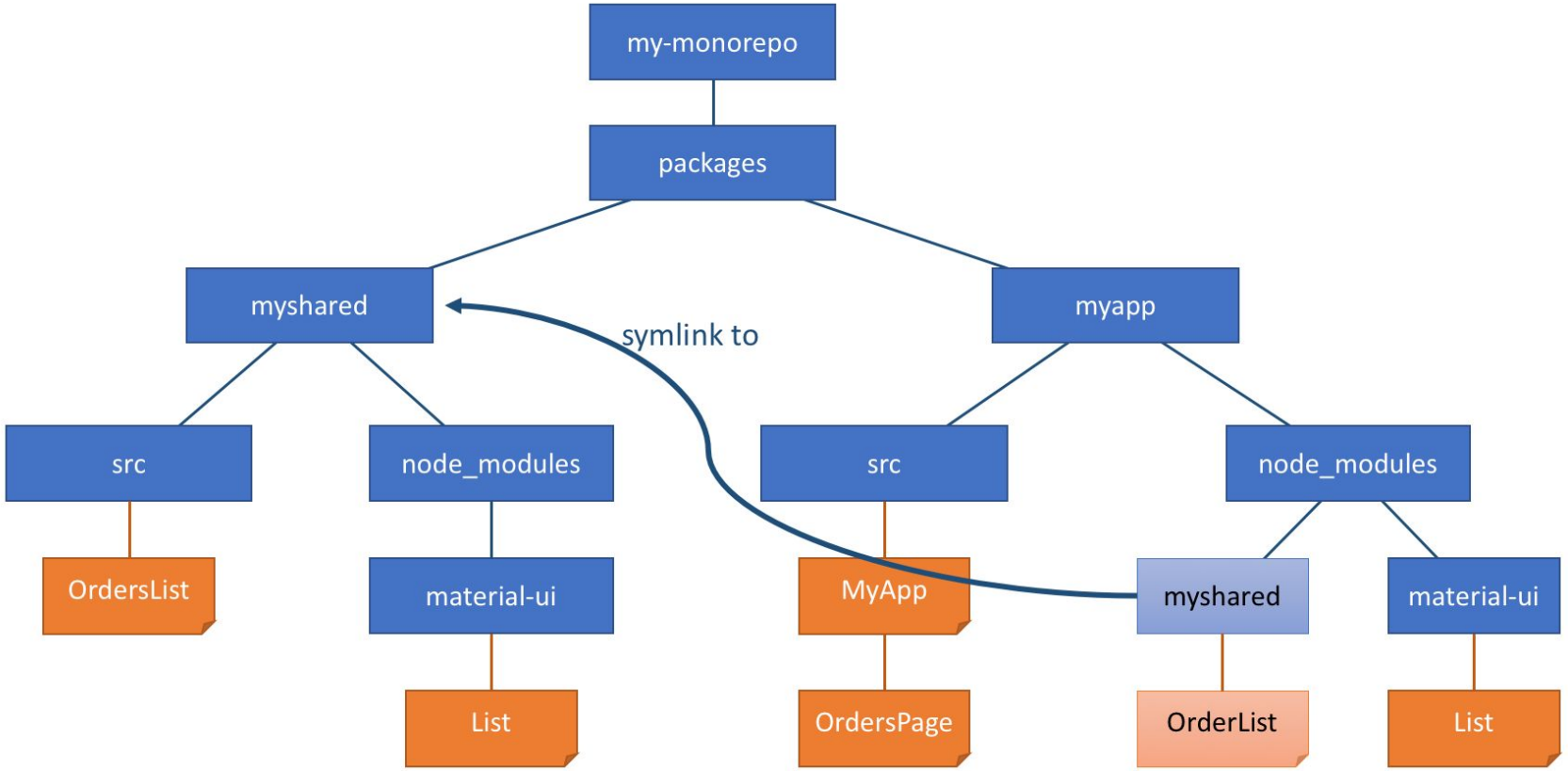
Монорепа



Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

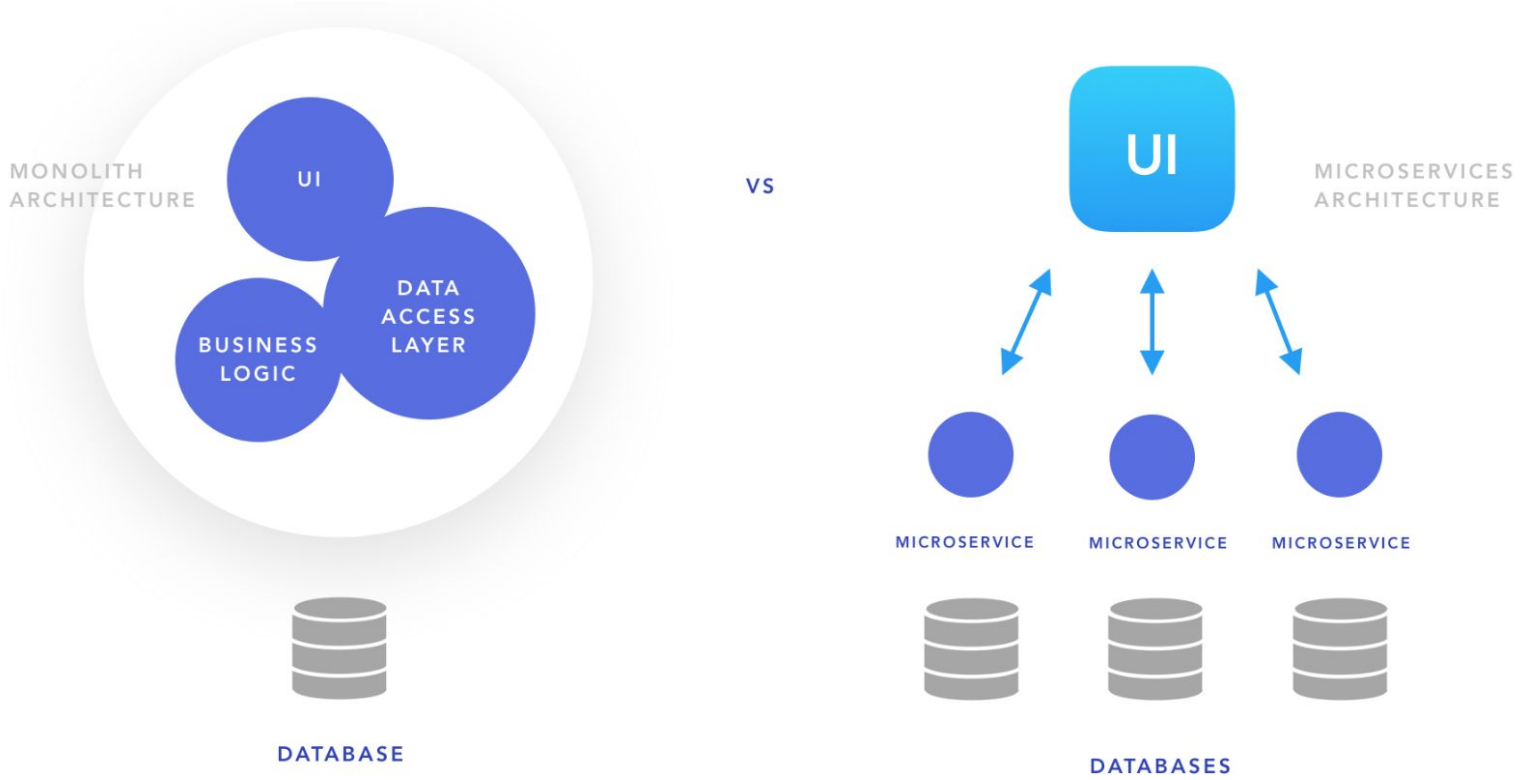
Монорепа



Структура проекта

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Микросервисы



Rendering

С внедрением Virtual DOM React делает обновления пользовательского интерфейса максимально эффективными. Это упрощает работу с веб-приложениями.

В основе React лежит синтаксис JSX и мощная способность React создавать и сравнивать виртуальные домены. С момента своего выпуска React оказал влияние на многие другие интерфейсные библиотеки.

Например, Vue.js также опирается на идею виртуальных доменов.

Каждое приложение React начинается с корневого компонента (root).
Все приложение имеют вид дерева, где каждый узел является компонентом.

Компоненты в React - это "функции", которые отображают пользовательский интерфейс на основе данных. Это означает, что он получает реквизиты и состояние.

Rendering

Пользователи взаимодействуют с пользовательским интерфейсом и вызывают изменение данных.

Взаимодействия - это все, что пользователь может делать в нашем приложении.

Например, нажатие кнопки, перемещение изображений, перетаскивание элементов списка и AJAX-запросы, вызывающие API.

Все взаимодействия изменяют только данные. Они никогда не вызывают никаких изменений в пользовательском интерфейсе.

Здесь данные - это все, что определяет состояние приложения. Не только то, что мы сохранили в нашей базе данных. Частью этих данных являются даже различные состояния интерфейса, например, какая вкладка выбрана в данный момент или установлен флажок в данный момент или нет. Всякий раз, когда происходит изменение данных, React использует функции компонента для повторного рендеринга пользовательского интерфейса, но только виртуально:

```
UI1 = CF(data1)UI2 = CF(data2)
```

Rendering

React вычисляет различия между текущим пользовательским интерфейсом и новым пользовательским интерфейсом, применяя алгоритм сравнения (reconciliation) к двум версиям своего виртуального DOM.

```
Changes = Difference(UI1, UI2)
```

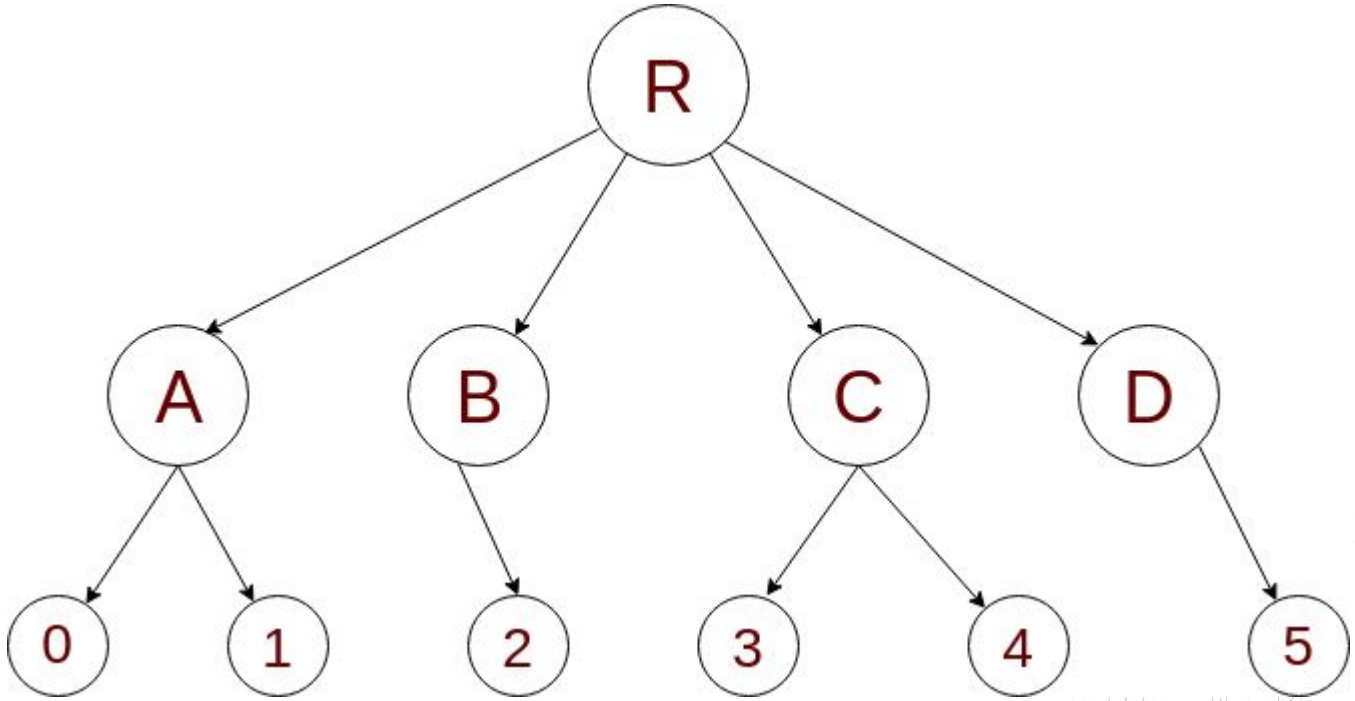
Затем React продолжает применять только изменения пользовательского интерфейса к реальному пользовательскому интерфейсу в браузере.

Когда данные, связанные с компонентом, изменяются, React определяет, требуется ли фактическое обновление DOM. Это позволяет React избегать потенциально дорогостоящих операций манипулирования DOM в браузере. Такие примеры, как создание узлов DOM и доступ к существующим без необходимости.

Rendering

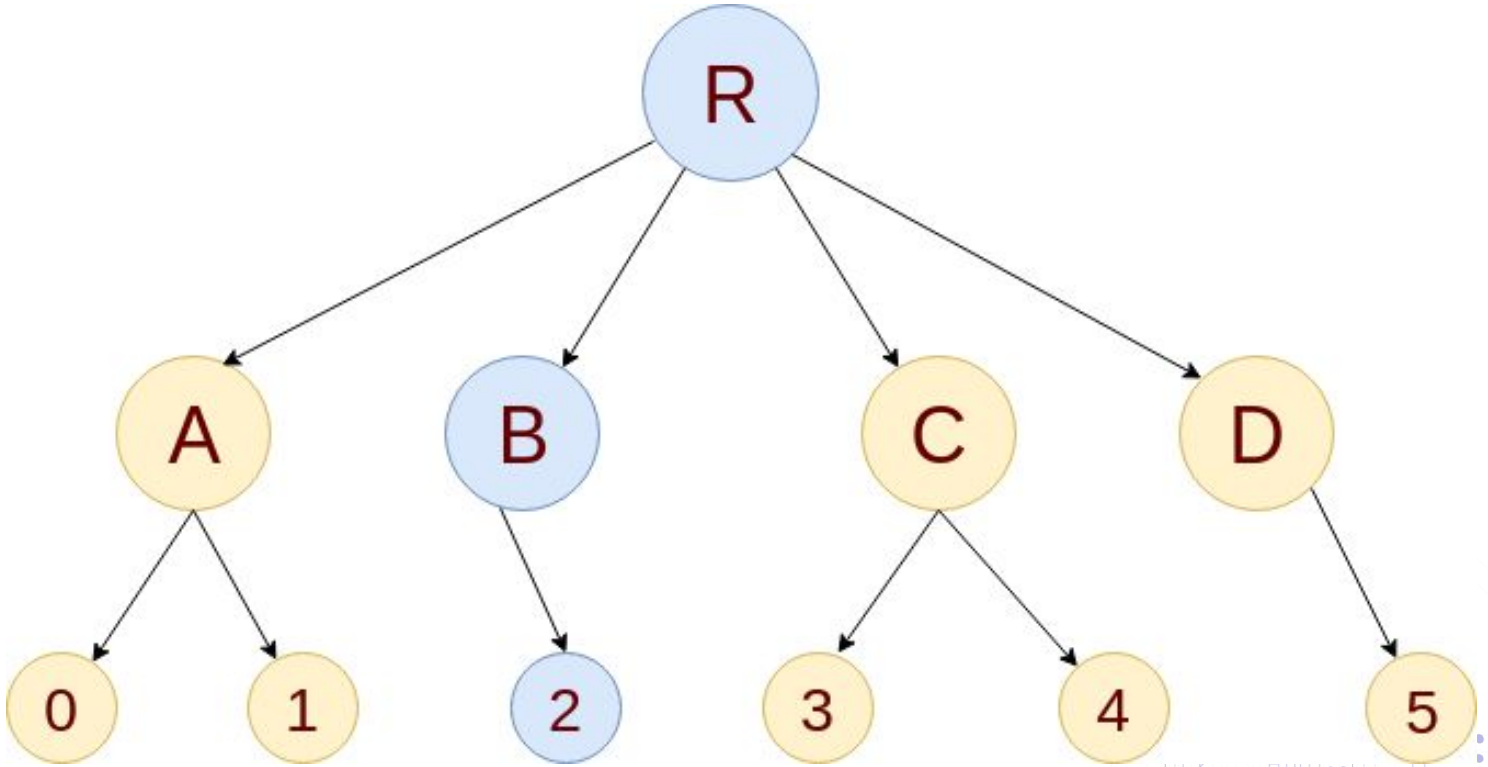
ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Во время начального процесса рендеринга React создает DOM-дерево, подобное этому —



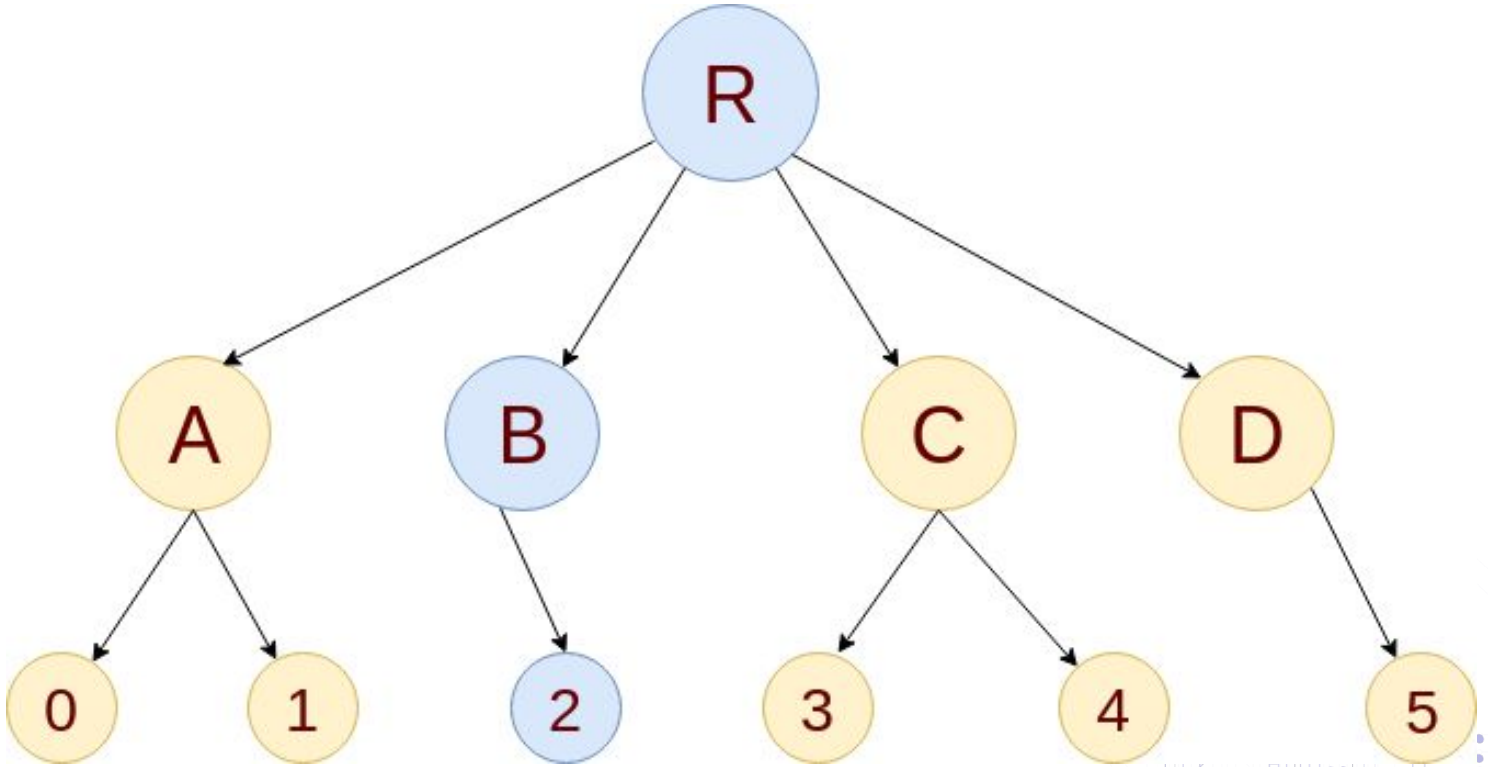
Rendering

Предположим, что часть данных изменяется.



Rendering

Предположим, что часть данных изменяется.

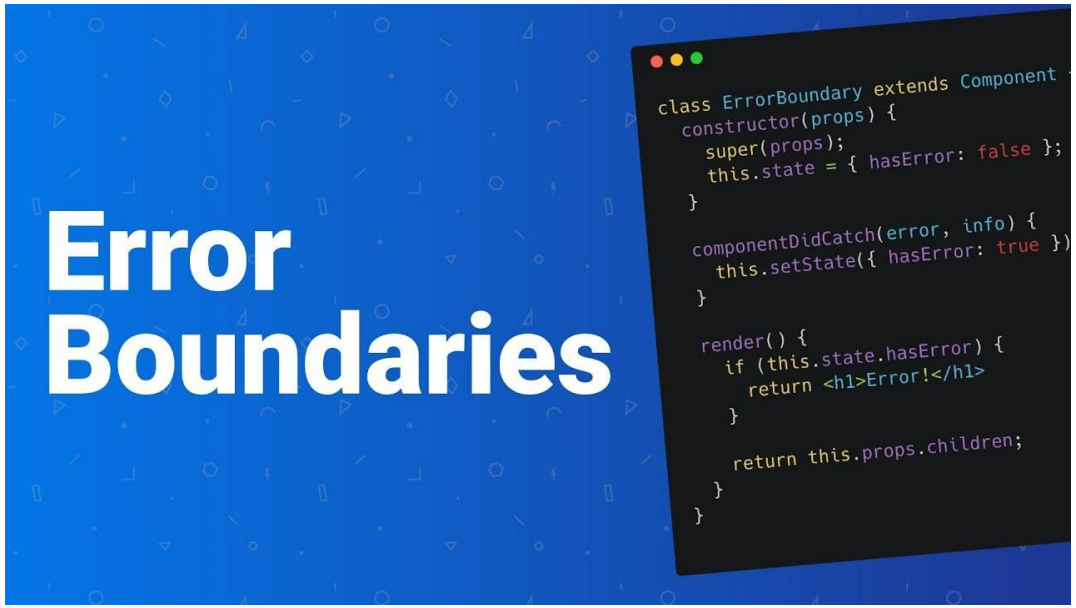


Компоненты

Обработка ошибок Error Boundary

Ошибки во время работы или белый экран с ошибками должны быть качественно обработаны.

Error Boundary — это компоненты React, которые отлавливают ошибки JavaScript в любом месте деревьев их дочерних компонентов, сохраняют их в журнале ошибок и выводят запасной UI вместо рухнувшего дерева компонентов.



Компоненты

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

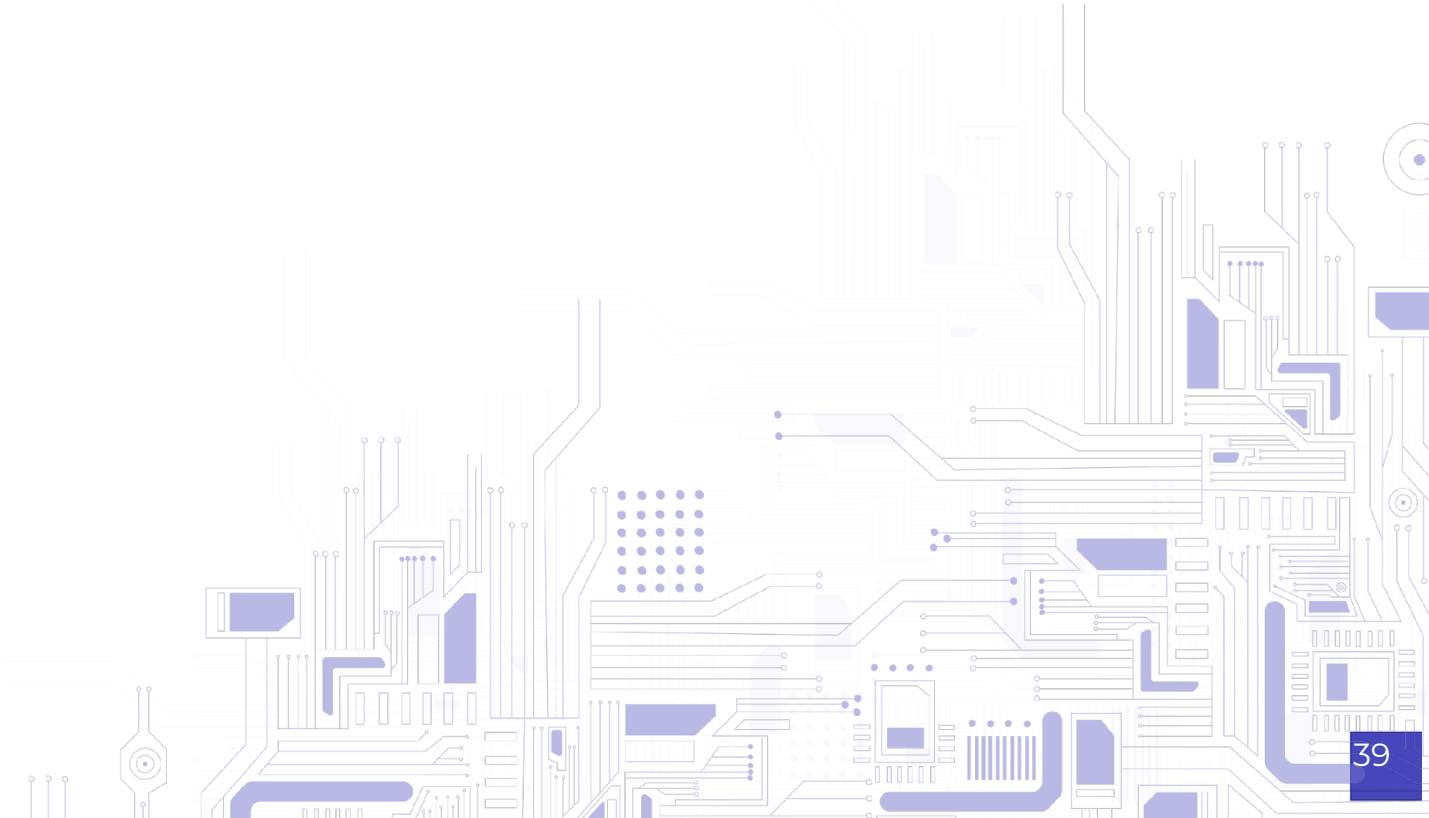
```
class ErrorBoundary extends Component {
    state = {
        error: null,
    };
    static getDerivedStateFromError(error) {
        return { error };
    }
    render() {
        const { error } = this.state;

        if (error) {
            return (
                <div>
                    <p>Seems like an error occured!</p>
                    <p>{error.message}</p>
                </div>
            );
        }
        return this.props.children;
    }
}

export default ErrorBoundary;
```

Компоненты

Отлавливаем ошибки внутри функции *getDerivedStateFromError* и помещаем их в состояние компонента.
Если ошибка произошла, то мы отображаем её текст, а если неу, то просто возвращаем компонент, который должен отображаться.



Компоненты


```
const Users = ({ userData, handleMoreDetails }) => {
    return (
        <div>
            <h1>Users List: </h1>

            <ul>
                {userData.map((user) => (
                    <div key={user.id}>
                        <p>Name: {user.name}</p>
                        <p>Company: {user.company}</p>
                        <button onClick={() => handleMoreDetails(user.id)}>
                            More details
                        </button>
                    </div>
                ))}
            </ul>
        </div>
    );
};
```

Компоненты

Компонент *User* будет работать, пока всё в порядке с получением данных из userData.
Если по какой-то причине userData будет *undefined* или *null*, приложение упадет.

Используем обработку

```
const Users = ({ userData, handleMoreDetails }) => {
    return (
        <div>
            <h1>Users List: </h1>
            <ErrorBoundary>
                <ul>
                    {userData.map((user) => (
                        <div key={user.id}>
                            <p>Name: {user.name}</p>
                            <p>Company: {user.company}</p>
                            <button onClick={() => handleMoreDetails(user.id)}>
                                More details
                            </button>
                        </div>
                    ))}
                </ul>
            </ErrorBoundary>
        </div>
    );
};
```

Компоненты

Компонент *User* будет работать, пока всё в порядке с получением данных из userData.
Если по какой-то причине userData будет *undefined* или *null*, приложение упадет.

Используем обработку

```
const Users = ({ userData, handleMoreDetails }) => {
    return (
        <div>
            <h1>Users List: </h1>
            <ErrorBoundary>
                <ul>
                    {userData.map((user) => (
                        <div key={user.id}>
                            <p>Name: {user.name}</p>
                            <p>Company: {user.company}</p>
                            <button onClick={() => handleMoreDetails(user.id)}>
                                More details
                            </button>
                        </div>
                    ))}
                </ul>
            </ErrorBoundary>
        </div>
    );
};
```

Когда ошибка произойдет, наш Error Boundary компонент отловит ошибку, и текст данной ошибки будет отображен на экране.

Error Boundary будет отображаться вместо компонента.

Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Спасибо за внимание