УДК 003.26

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

МЕТОД ПЕРЕХВАТА ОШИБОК В REACT.COMPONENT

В публикации речь пойдет о методе перехвата ошибок из компонента и отображении резервного компонента («ErrorComponent») в случае возникновения ошибки при отображении компонента. Для демонстрации данного метода будем работать с событиями жизненного цикла:

– работа с «**componentDidCatch**»;

– использование события жизненного цикла «**getDerivedStateFromError**».

Метод опишем через сценарий варианта использования, который представим через реализацию следующих шагов:

– создадим компонент «EmployeeDetails», который содержит поле ввода;

– если пользователь оставляет в поле ввода пробел, компонент должен вернуть ошибку: пробелы не допустимы;

– развернем механизм, который отслеживает возвращаемую ошибку. При появлении ошибки компонент должен отображать «ErrorComponent», а не «EmployeeDetails».

Приступаем к работе с компонентом «EmployeeDetails». Сначала создадим компонент «EmployeeDetails», который выдаёт ошибку, если поле ввода содержит пробел (листинг 1).

```
export class EmployeeDetails extends React.Component {
  updateName(event) {
  if(event.target.value.indexOf(" ") > -1) {
    throw "Name Cannot Contain blank Spaces";
  }
 }
 render() {
  return (
    <div>
      Enter Name: <input type="text" placeholder="Enter Name" on-
Change={this.updateName} />
    </div>
  )
 }
}
```

Листинг 1 – Компонент «EmployeeDetails»

Каждый раз, когда мы вносим какие-то изменения в поле ввода, в этом компоненте вызывается функция «updateName». Она проверяет

текущее обновлённое значение текстового поля на предмет наличия пробелов. Если пробелы имеются, компонент возвращает ошибку. Обработки ошибки не происходит, поэтому приложение завершит работу с ошибкой «Name Cannot Contain blank Spaces» («Имя не может содержать пробелы»).

Добавление механизма отслеживания ошибок с помощью «componentDidCatch» будем осуществлять следующим образом. Для перехвата ошибки введём понятие «граница ошибки». Создадим компонент, выступающий в роли границы, для перехвата о шибок, которые будут вызваны всеми его дочерними компонентами. Предположим, у нас есть компонент «EmployeeDetails», выдающий ошибку на основе определённого условия. Для перехвата ошибки, вызванной внутри него, нужно создать родительский компонент, в котором будет находиться этот вызывающий ошибки дочерний компонент. Теперь создадим компонент, который можно представить как «**ErrorBoundaries**» (листинг 2).

```
import React from 'react';
export default class ErrorBoundaries extends React.Component {
   componentDidCatch(error, info) {
      console.log("Component Did Catch Error");
   }
   render() {
      return (
         <div>
            <EmployeeDetails />
         </div>
      )
   }
}
```

Листинг 2 – Компонент «ErrorBoundaries»

Созданный компонент «ErrorBoundaries» может отображать дочерние компоненты. Класс реализует метод «componentDidCatch», который будет вызван, как только какой-либо из дочерних компонентов вернёт ошибку. Все ошибки, вызванные внутри дочернего компонента, будут перехватываться в этом методе. Поэтому «ErrorBoundaries» можно считать внешним компонентом, который не позволяет ошибкам просачиваться за его границы.

Вернёмся к компоненту «ErrorComponent». В том случае, когда компонент возвращает ошибку, он должен отображать «ErrorComponent». Этот компонент содержит заголовок, по которому можно понять, что пошло не так при использовании «EmployeeDetails». Чтобы развернуть этот резервный механизм во

время появления ошибки, можно реализовать событие жизненного цикла «getDerivedStateFromError».

Понятие жизненного цикла «getDerivedStateFromError». Это событие жизненного цикла можно использовать для определения нового состояния после получения ошибки.

Как только ошибка получена, будет выполнено следующее событие жизненного цикла, что может обновить состояние показанного выше компонента «ErrorBoundaries». Это состояние можно настроить так, чтобы оно давало понять, как выглядит ситуация ошибки в приложении. После обновления состояния компонент «ErrorBoundaries» будет повторно отображён. На основе состояния компонента можно будет развернуть «ErrorComponent». Расширим компонент «**ErrorBoundaries**» и реализуем его (листинг 3).

```
import React from 'react';
export default class App extends React.Component {
 constructor() {
  super();
  this.state = {
   hasError: false
  }
 }
 componentDidCatch(error, info) {
    console.log("Component Did Catch Error");
 }
  static getDerivedStateFromError(error) {
  return { hasError: true };
 }
  render() {
     return (
       <div>
         { !this.state.hasError && <EmployeeDetails /> }
         { this.state.hasError && <ErrorComponent /> }
       </div>
     )
  }
}
```

Листинг 3 – Расширенный компонент «ErrorBoundaries»

В приведённом выше компоненте мы создали переменную состояния «hasError». Её можно использовать для отслеживания ошибки внутри дочерних компонентов. В начальной фазе никаких ошибок нет, поэтому исходным значением «hasError» будет «false». Как только в компоненте «EmployeeDetails» появляется ошибка, вызывается функция «getDerivedStateFromError», которая устанавливает для «hasError» значение «true». Компонент настраивает новое состояние, поэтому он

будет отображён повторно. В функции «render» мы указали условие, которое определяет, что будет отображаться: «EmployeeDetails» или «ErrorComponent». Так как для «hasError» установлено значение «true», отображён будет «ErrorComponent». В этом сценарии ошибка обработана, поэтому приложению ничто не мешает продолжать работу.

УДК 519.71

И.К. Асмыкович, доц., канд. физ.-мат. наук (БГТУ, г. Минск)

О СВЕРХУСТОЙЧИВОСТИ РЕГУЛЯРНЫХ ДЕСКРИПТОРНЫХ СИСТЕМ

Некоторое время считалось, что все основные вопросы качественной теории управления для линейных динамических систем рассмотрены и решены в 70-80 годы двадцатого века. Но при внимательном рассмотрении выяснилось, что ряд стандартных задач, как-то задача стабилизации линейной системы обратной связью по выходу, задача робастной стабилизации, задача одновременной стабилизации набора систем являются невыпуклыми и NP-сложными. Это означает, что эффективных способов их решения нет и быть не может, если под эффективным способом понимать такой который приводит к точному решению (если оно существует) с произвольной точностью за «разумное» время [1].

Известно, что первым требованием к реальной системе управления является требование устойчивости [2]. Отметив, что множество устойчивых линейных систем невыпукло в пространстве параметров матрицы системы, было предложено [1, 3] выделить класс сверхустойчивых линейных систем, определив это понятие не через собственные числа матрицы, а непосредственно через параметры матрицы системы, а именно, потребовав для непрерывных систем выполнения для элементов матрицы системы неравенства $\sigma(A) = \min(-a_{ii} - \sum_{j \neq i} |a_{ij}|) > 0$ а для дискретных систем $\|A\| = \max \sum_{j} |a_{ij}| < 1$. Эти условия являются достаточными для асимптотической устойчивости по Ляпунову, их решения обладают хорошими свойствами сходимости и для них многие задачи качественной теории управления, такие как синтез статической обратной связи по выходу, одновременная стабилизация, робастная стабилизация, подавление возмущений сводятся к задачам линейного программирования и имеют достаточно хорошее численное решение. Но достаточно ясно, что сверхустойчивые системы составляют весьма узкий подкласс устойчивых по Ляпу-