МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики»

**Кафедра Компьютерного проектирования и дизайна**

Направление (специальность) ⎯ 09.04.02 Информационные системы и технологии

Специализация ⎯ Веб технологии (СОП)

Научно-исследовательская работа

**ТЕМА:** Подходы к оптимизации разработки сайтов с использованием современных веб-технологий.

**3-ИЙ ЭТАП:** Разработка пользовательского интерфейса с использованием react.js

ВЫПОЛНИЛ

Студент группы \_s4207c\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.С. Кузменков

№ группы подпись, дата ФИО

ПРОВЕРИЛ \_\_к.т.н., доц.\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.А. Сокуренко

ученая степень, должность подпись, дата ФИО

САНКТ-ПЕТЕРБУРГ  
2018 г.

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| Введение...…………………………………………………………………….. | 3 |
| 1. Компонентный подход программирования……………………………… | 7 |
| 2. Обзор технологий *front-end* разработки веб-приложений ……………… | 9 |
| 3. Компонентный подход во *front-end* ……………………………………… | 14 |
| 4. Компонентная разработка с использованием react.js…………………… | 17 |
| 5. Разработка *front-end* приложения управления автоматизированным тестированием проектов……………………………………………………………… | 21 |
| Заключение…………………………………………………………………… | 23 |
| Список использованной литературы……………………………………….. | 25 |
| Приложение А. Исходный код……………………………………………… | 31 |

**ВВЕДЕНИЕ**

**Анализ современных тенденций**.

Широкое использование компьютерных информационных технологий и мультимедиа способствовало создание специальных технических средств, обеспечивающих непосредственный бесклавиатурный ввод в компьютер текстовой и графической информации. В связи с этим развивается разнообразие интерактивных систем, как по вводу информации, так и по отклику. Данные системы постепенно заполняют окружающий мир, приведем примеры: интерактивные доски, витрины, панели и многое другое. Встает вопрос о развитии управления данными системами. На данный момент широко известна технология сенсорного управления, но так же очень перспективно развивается технология жестового управления.

**Актуальность темы** (описать актуальность темы). Связь с непосредственным развитием интерактивных систем и доступа к информации путем управления визуализации с помощью новейших технологий.

Так, к примеру, управление жестами только появляются и популярности пока не имеют, но это может подтолкнуть к изучению интерфейса и способствует интересу к представленной информации, а так же её продвижению, цепной реакции рассказа друг другу о новом. Ещё можно отметить, что жест намного более информативен: необходимо использовать хороший алгоритм и аппаратные средства для восприятия образов.

Руки и тело человека являются естественными манипуляторами и обладают большим числом степеней свободы, поэтому успешное решение задачи распознавания выполняемых ими жестов раскрывает перспективы для решения широкого круга прикладных задач.

Таким образом, комбинирование методов и алгоритмов для технологии управления визуализацией информации по средствам жестовых контроллеров может повысить эффективность использования новейших интерактивных систем для донесения информации пользователю. Результаты данного исследования позволят использовать данную технологию в различных сферах, например: образование, медицина, реклама, компьютерное зрение (взаимодействие с компьютерами и робототехникой).

**Постановка задачи** (кратко сформулировать основную задачу исследования по теме ВКР)

Постановка следующих задач:

* Исследовать наиболее популярные технологии управления визуализацией информации для интерактивных систем.
* Провести апробацию технологии управления жестами.
* Проанализировать результаты апробации по необходимым критериям распознавания жестов.
* Изучить методы распознавания жестов.
* Исследовать комбинации данных методов для реализации рекомендаций по улучшению критериев распознавания динамических сложных жестов.

**Объект исследования** – процесс *front-end* разработки веб-приложений и сайтов различной сложности с использованием современных технологий и тенденций программирования (*JS*-фреймворков, и библиотек).

**Предмет исследования** – подходы и методы оптимизации *front-end* разработки сайтов и веб-приложений, реализованные в конкретной технологии, а также использование ими компонентно-ориентированного подхода.

**Теоретическая и прикладная значимость**

Теоретическая значимость работы состоит в том, что все методы и алгоритмы, а так же их комбинирование, будут тщательным образом описаны и предоставлены научному сообществу, в результате чего с их использованием могут быть проведены новые исследования. В открытом доступе находится минимальное количество информации по данному вопросу, что обуславливает ценность данного исследования.

Практическая значимость работы заключается в том, что в результате должно быть предложено такое комбинирование методов распознавания жестов, которое позволит улучшить автоматизацию технологии управления жестами интерфейсом интерактивных систем. Влияющей на такие критерии как скорость отклика системы на жест, распознавание более мелких и сложных жестов, а так же восприятия большего количества людей в области обхвата камеры глубины.

Раздел 1

Раздел 2

Раздел 3 Решение задачи (Описать предложенный метод/технологию и т.д. при помощи которой будет достигнута поставленная цель. Если для решения задачи/метода используются программные продукты, то их так же необходимо описать)

**ЗАКЛЮЧЕНИЕ**

По результатам проведенных исследований можно сделать вывод о широком распространении *JavaScript* в различных областях разработки *front-end* и *back-end* частей современных веб-приложений. Развитие стандарта *ECMOScript* позволяет в значительной степени улучшить процесс разработки, а так же использовать современные подходы к написанию кода различной сложности. Так последний, поддерживаемый разработчиками браузерных движков, стандарт *ES6* позволяет использовать объектно-ориентированных подход в более приближенном к языкам программирова

Так же JavaScript нашол свою нисшу для разработки мобильныйх приложений (*React Native*, *JQuery mobile* и др.). Постоянное развитие и совершенствование языка позволяет не только решить проблемы вызванные несогласованностью разработчиков браузеров, но и в значительной степени расширить его возможности. Разработчики стандартов *ECMOScript* стараются учитывать при расширении функциональных возможностей языка тенденции в развитии современного программирования. Это позволяет использовать преимущества нескольких подходов и опустить недостатки, а также обеспечивает легкий переход программистов с других языков.

Развитие современных фреймворков и библиотек позволяет в значительной степени ускорить разработку за счет использования готовых решений типовых задач. Во многом это достигается в использовании готовых компонент, с заложенным для выполнения конкретной задачи функционалом.

Однако при этом высокие темпы развития приводят к высоким темпам устаревания решений. Так такие технологии как *jQuery*, *Bootstrap*, *ExtJS* на данный момент морально устарели, хотя и имеют широкую сферу применения за счет обширных библиотек компонентов и использования в связки с другими технологиями. Таким образом использование компонентного подхода обеспечивает существование данных технологий и их использование для решения узкого спектра задач.

Однако наиболее широкого применения данный подход достиг в библиотеке *react.js*. Концепция данной технологии с одной стороны не нова, с другой очень проста в использовании. На момент своего появления технологии получила достаточно противоречивые отзывы, однако она представляет собой будущее *front-end* разработки. Неожиданным результатом оказалось удобство подхода для использования в мобильной разработке.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

**Приложение А. Исходный код некоторых компонент**

App.js:

import React, { Component } from 'react';

import { Router, Switch, Route } from 'react-router'

import createHistory from 'history/createBrowserHistory';

import Header from './components/Header';

import Footer from './components/Footer';

import HomePage from './pages/HomePage';

import AboutMePage from './pages/AboutMePage';

import ItemPage from './pages/ItemPage';

import ErrorPage from './pages/ErrorPage';

import LogoPage from './pages/LogoPage';

import TaskManagerPage from './pages/TaskManagerPage';

import \* as PropTypes from 'prop-types';

class App extends Component {

static PAGE\_LINKS = [

{ "link": "/task-manager", "title": "Task manager" },

{ "link": "/about", "title": "About me" }

];

getChildContext() {

return {

pageLinks: App.PAGE\_LINKS

}

}

render() {

return (

<div className="app">

<Header text="Front-end EPAM training" />

<Router history={ createHistory() }>

<Switch>

<Route exact path="/" component={ HomePage } />

<Route path="/about" component={ AboutMePage } />

<Route path="/task-manager" component={ TaskManagerPage } />

<Route path="/item/:name/:environment" component={ ItemPage } />

<Route path="/logo.txt" component={ LogoPage }/>

<Route path="/\*" component={ ErrorPage } />

</Switch>

</Router>

<Footer date="2017" title="Kuzmiankou Anatoli" />

</div>

);

}

static childContextTypes = {

pageLinks: PropTypes.arrayOf(PropTypes.shape({

link: PropTypes.string,

title: PropTypes.string

}))

}

}

export default App;

TaskManagerPage.js:

import React, { Component } from 'react';

import AddComponentWindow from '../components/AddComponentWindow';

import TableBox from '../components/TableBox';

import TaskComponentManager from '../components/TaskComponentManager';

import Counter from '../components/Counter';

import Main from '../components/Main';

import \* as PropTypes from 'prop-types';

import { connect } from 'react-redux'

import { loadTasks, loadTasksSuccess } from '../redux/action/TaskAction';

class TaskManagerPage extends Component {

componentDidMount() {

this.props.loadingData();

}

createTaskManager(data, number) {

const { name, environments } = data;

return (

<TaskComponentManager taskName={ name } key={ name } tasks={ environments } />

);

}

static ENVIRONMENTS = [

{ name: "int", title: "INT" },

{ name: "qa", title: "QA" },

{ name: "staging", title: "Staging" },

{ name: "production", title: "Production" }

];

getChildContext() {

return {

environments: TaskManagerPage.ENVIRONMENTS

};

}

render() {

const { loading, tasks } = this.props;

return (

<Main name="Task manager" loading={ loading }>

<Counter title="Shared Services / Component" name="components-count" count={ tasks.length } />

<AddComponentWindow />

<TableBox>

{ tasks.map( (item, number) => this.createTaskManager(item, number) ) }

</TableBox>

</Main>

);

}

static childContextTypes = {

environments: PropTypes.arrayOf(PropTypes.shape({

name: PropTypes.string,

title: PropTypes.string

}))

}

}

const mapStateToProps = state => {

const { TaskReducer } = state;

return { ...TaskReducer };

}

const mapDispatchToProps = dispatch => ({

loadingData: () => {

dispatch(loadTasks());

fetch('http://localhost:9999/data/projects')

.then(res => res.json())

.then(json => dispatch(loadTasksSuccess(json)));

}

});

export default connect(mapStateToProps, mapDispatchToProps)(TaskManagerPage);

Task.js:

import \* as BodyFactory from "./lib/TaskBodyFactory";

import \* as StatusFactory from "./lib/TaskStatusFactory";

import TaskStatus from "./lib/TaskStatus";

import \* as PropTypes from 'prop-types';

import { connect } from 'react-redux';

import { refreshTask, refreshTaskSuccess } from '../../redux/action/TaskAction';

import autoBind from 'react-autobind';

class Task extends Component {

constructor(props) {

super(props);

autoBind(this);

}

refreshAction() {

const { name, env } = this.props;

this.props.refreshAction({ name, env });

}

shouldComponentUpdate(nextProps, nextState) {

for(let index in this.props) {

if(this.props[index] !== nextProps[index]) {

return true;

}

}

return false;

}

render() {

const { env, link, data } = this.props;

const { status, version } = this.props.data;

return(

<td className={ `task task-${ env } task-${ TaskStatus[status].toLowerCase() }` }>

<div className="header">

<div className="version">{ version }</div>

{ StatusFactory.createStatus(status) }

</div>

{ BodyFactory.createBodyElem(data) }

<div className="control-panel">

<a href={ link } className="link" ><span className="icon-link" /></a>

<button className="refresh icon-refresh" onClick={ this.refreshAction } />

</div>

</td>

);

}

static defaultProps = {

name: "PROJECT",

env: "int",

data: {

version: "v-.-.-",

status: TaskStatus.MESSING

}

}

static propsTypes = {

name: PropTypes.string,

env: PropTypes.string,

data: PropTypes.shape({

version: PropTypes.string,

status: PropTypes.number,

timestamps: PropTypes.string,

testResult: PropTypes.shape({

total: PropTypes.number,

failed: PropTypes.number,

passed: PropTypes.number,

skipped: PropTypes.number

}),

logo: PropTypes.string

}),

link: PropTypes.string

};

}

const mapDispatchToProps = dispatch => ({

refreshAction: params => {

dispatch(refreshTask());

fetch('http://localhost:9999/data/refresh', {

method: 'post',

headers: {

'Accept': 'application/json',

'Content-Type': 'application/json'

},

body: JSON.stringify(params)

}).then(res => res.json())

.then(data => dispatch(refreshTaskSuccess(params, data)));

}

});

export default connect(null, mapDispatchToProps)(Task);

TaskBodyFactory.js:

import React from 'react';

import Status from './TaskStatus';

import moment from 'moment';

const viewQueue = status => (

<div className="inqueue" key="inqueue">In queue</div>

)

const viewTests = tests => (

<div key="tests">

{ `${ tests.failed } failed, ${ tests.passed } passed, ${ tests.skipped } skipped, ${ tests.total } total` }

</div>

);

const viewLogo = link => (

<div key="link">See <a href={ `/${ link }` } target="\_blank">{ link }</a></div>

);

const viewTime = time => (

<div key="date">{ moment(time).fromNow() }</div>

);

export const createBodyElem = data => (

<div className="body">

{ data.status === Status.QUEUE && viewQueue(data.status) }

{ data.testResult && viewTests(data.testResult) }

{ data.logo && viewLogo(data.logo) }

{ data.timestamps && viewTime(data.timestamps) }

</div>

);

TaskStatus.js:

const TaskStatus = {

QUEUE: 'QUEUE',

MISSING: 'MISSING',

RUNNING: 'RUNNING',

FAILED: 'FAILED',

SUCCESS: 'SUCCESS',

W\_O\_FAILED: 'W\_O\_FAILED'

}

export default TaskStatus;

TaskStatusFactory.js:

import React from 'react';

import Status from './TaskStatus';

const titles = {

[Status.QUEUE]: "IN QUEUE",

[Status.MISSING]: "MISSING AURA.JSON",

[Status.RUNNING]: "RUNNING",

[Status.FAILED]: "FAILED",

[Status.W\_O\_FAILED]: "FAILED"

}

const icons = {

[Status.SUCCESS]: "icon-ok",

[Status.W\_O\_FAILED]: "icon-warning"

}

export const createStatus = status => (

<div className={ `status ${ icons[status] }` }>{ titles[status] }</div>

);

AddComponentWindow.js:

import React, {Component} from 'react';

import TextForm from './TextForm';

import ModalWindow from './ModalWindow';

import \* as PropTypes from 'prop-types';

import { addProjectTasks, addProjectTasksSuccess } from '../redux/action/TaskAction';

import { connect } from 'react-redux';

import autoBind from 'react-autobind';

class AddComponentWindow extends Component {

constructor(props) {

super(props);

autoBind(this);

}

getChildContext() {

const { environments } = this.context;

const fields = [{ name: "name", labelValue: "Item name" }];

environments && environments.forEach( item =>

fields.push({

name: item.name,

labelValue: `${ item.title } envirment url`

})

);

return { fields };

}

formAction(data) {

this.hideWindow();

this.props.addComponent(data);

this.form.resetAction();

}

hideWindow() {

this.modalWindow.hide();

}

showWindow() {

this.modalWindow.show();

}

render() {

return (

<figure className="add-component" name="add-component">

<button name="add-component" onClick={ this.showWindow }

className="btn-add-component">Add Component</button>

<ModalWindow ref={ window => this.modalWindow = window } title="Add component item">

<TextForm ref={ form => this.form = form } submitName="Add" submitAction={ this.formAction }/>

</ModalWindow>

</figure>

);

}

static contextTypes = {

environments: PropTypes.arrayOf(PropTypes.shape({

name: PropTypes.string,

title: PropTypes.string

})),

};

static childContextTypes = {

fields: PropTypes.arrayOf(PropTypes.shape({

name: PropTypes.string,

labelValue: PropTypes.string

}))

};

}

const mapDispatchToProps = dispatch => ({

addComponent: params => {

dispatch(addProjectTasks(params.name));

fetch('http://localhost:9999/data/add-project', {

method: 'post',

headers: {

'Accept': 'application/json',

'Content-Type': 'application/json'

},

body: JSON.stringify(params)

}).then(res => res.json())

.then(json => dispatch(addProjectTasksSuccess(json)));

}

});

export default connect(null, mapDispatchToProps)(AddComponentWindow);

ModalWindow.js:

import React, {Component} from "react";

import \* as PropTypes from 'prop-types';

import autoBind from 'react-autobind';

class ModalWindow extends Component {

constructor(props) {

super(props);

autoBind(this);

}

componentDidMount() {

if(this.props.startVisible) {

this.show();

}

else {

this.hide();

}

}

hide() {

this.window.style.display = "none";

}

show() {

this.window.style.display = "block";

}

render() {

const { id, name, title, children } = this.props;

return (

<figure

id={ id }

name={ name }

className={ "modal modal-" + name }

ref={ window => this.window = window }

>

<div className="modal-content">

<h2>{ title }</h2>

<button className="btn-close" onClick={ this.hide }>x</button>

{ children }

</div>

</figure>

);

}

static defaultProps = {

name: "window",

title: "Modal window",

startVisible: false

};

static propTypes = {

id: PropTypes.string,

name: PropTypes.string,

title: PropTypes.string,

startVisible: PropTypes.bool

};

}

export default ModalWindow;

TaskComponentManager.js:

import React, {Component} from 'react';

import Task from './task/Task';

import \* as PropTypes from 'prop-types';

import { connect } from 'react-redux';

import { removeProjectTasks, removeProjectTasksSuccess } from '../redux/action/TaskAction';

import autoBind from 'react-autobind';

class TaskComponentManager extends Component {

constructor(props) {

super(props);

autoBind(this);

}

createTask(key, task) {

const name = this.props.taskName;

return (

<Task

name={ name }

key={ key }

env={ key }

data={ task }

link={ `/item/${ name }/${ key }` }

/>

)

}

createEmptyTask(key) {

return <td key={ key }><div className="task-empty" /></td>

}

createTasks(tasks) {

return this.context.environments.map(item => {

const name = item.name;

const task = tasks[name];

return task ? this.createTask(name, task) : this.createEmptyTask(name);

});

}

removeComponent() {

this.props.removeAction({ name: this.props.taskName });

}

render() {

const { name, taskName, tasks } = this.props;

return (

<tbody className={ `task-component-manager manager-${ name }` }>

<tr>

<td className="task-manager">

<h2 className="title">{ taskName }</h2>

<ul className="settings icon-setting">

<li onClick={ this.removeComponent }>Remove</li>

</ul>

</td>

{ this.createTasks(tasks) }

</tr>

</tbody>

);

}

static defaultProps = {

name: "task-manager"

};

static contextTypes = {

environments: PropTypes.arrayOf(PropTypes.shape({

name: PropTypes.string,

title: PropTypes.string

})),

};

static propTypes = {

name: PropTypes.string,

taskName: PropTypes.string,

tasks: PropTypes.objectOf(PropTypes.shape({

version: PropTypes.string,

status: PropTypes.string,

testResult: PropTypes.shape({

total: PropTypes.number,

failed: PropTypes.number,

passed: PropTypes.number,

skipped: PropTypes.number

}),

timestatmps: PropTypes.string,

logo: PropTypes.string

}))

};

}

const mapDispatchToProps = dispatch => ({

removeAction: params => {

dispatch(removeProjectTasks());

fetch('http://localhost:9999/data/remove-tasks', {

method: 'post',

headers: {

'Accept': 'application/json',

'Content-Type': 'application/json'

},

body: JSON.stringify(params)

}).then(res => res.json())

.then(data => dispatch(removeProjectTasksSuccess(params.name, data.result)));

}

});

export default connect(null, mapDispatchToProps)(TaskComponentManager);

NavBar.js:

import React from 'react';

import \* as PropTypes from 'prop-types';

const NavBar = (props, context) => {

const { name } = context

return (

<nav className="nav-bar">

<ul className="nav-bar-menu">

<li className="horisontal"><a href="/">Home</a></li>

{

(name)

? <li className="horisontal">{ name }</li>

: undefined

}

</ul>

</nav>

);

}

NavBar.contextTypes = {

name: PropTypes.string

};

export default NavBar;

TableBox.js:  
import React, {Component} from 'react';

import \* as PropTypes from 'prop-types';

class TableBox extends Component {

createTH(name) {

return (

<th className="column-tittle" key={ name }>{ name }</th>

);

}

render() {

const { environments } = this.context;

return (

<table className="table-box">

<thead>

<tr>

{ this.createTH("Name") }

{ environments && environments.map( item => this.createTH(item.title) ) }

</tr>

</thead>

{ this.props.children }

</table>

)

}

static contextTypes = {

environments: PropTypes.arrayOf(PropTypes.shape({

title: PropTypes.string

})),

};

}

export default TableBox;

Main.js:

import React, { Component } from 'react';

import \* as PropTypes from 'prop-types';

import NavBar from './NavBar';

import Loading from './Loading';

class Main extends Component {

getChildContext() {

const { name } = this.props;

return {

name

}

}

render() {

const { loading, children } = this.props;

return (

<main>

<NavBar />

{

(loading)

? <Loading />

: <div className="main-content">{ children }</div>

}

</main>

);

}

static childContextTypes = {

name: PropTypes.string,

loading: PropTypes.bool

};

}

export default Main;