**FULL STACK DEVELOPER ASSESSMENT**

1. Write a JavaScript program to sort a list of elements using Quick sort. Quick sort is a comparison sort, meaning that it can sort items of any type for which a "less-than" relation (formally, a total order) is defined. Note: do not use the array native function “sort”
2. Write a JavaScript program to sort a list of elements using Insertion sort. Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. Note: do not use the array native function “sort”
3. Write a JavaScript program to sort a list of elements using Merge sort. According to Wikipedia "Merge sort (also commonly spelled mergesort) is an O (n log n) comparison-based sorting algorithm. Most implementations produce a stable sort, which means that the implementation preserves the input order of equal elements in the sorted output." Note: do not use the array native function “sort”
4. Write a JavaScript program to compare two objects to determine if the first one contains equivalent property values to the second one.
5. Write a JavaScript program to measure the time taken by a function to execute
6. Write a JavaScript program to get the volume of a Cylinder with four decimal places using object classes. Volume of a cylinder : V = πr2h

Where r is the radius and h is the height of the cylinder.

1. Write a JavaScript program to demonstrate a class inheritance.
2. Write a ReactJS code snippet on implementing props vs state. Put a comment accordingly
3. Write a ReactJS code snippet on implementing hooks. Put a comment accordingly
4. Write a ReactJS code snippet sample of stateless vs stateful component. Put a comment accordingly
5. Having this sample document

| {  "\_id": MongoId("56902f7f31de51cdcfc03f07"),  "post": "Hello World",  "likes": 10,  "timestamp": 1572266114118 } |
| --- |

Create an aggregation query to get the average ‘like’ of post per day

| {  "\_id": {"month": 10, "day": 20, "year": 2019},  "totalLikes": 100  } |
| --- |

1. Having this sample document

| {  "\_id": MongoId("56902f7f31de51cdcfc03f07"),  "sms": "Привет, мир",  "recipient": "+74457771234",  "timestamp": 1572266114118 } |
| --- |

Create an aggregate query that will count the total number of SMS by 132 bytes per count

| {  "\_id": null,  "smsCount": 100 } |
| --- |
|  |
|  |

1. Having this document, create an update query to increase the count of an item(Apple) inside an array

| {  "\_id": MongoId("56902f7f31de51cdcfc03f07"),  "items": [  {  "code": 9643372659,  "name": "Apple",  "count": 100  },  {  "code": 7969928269,  "name": "Orange",  "count": 100  }  ] } |
| --- |
|  |

1. Design a sample JSON document that will represent a One-to-Many relationship in a non relational database such as MongoDB.
2. Design a sample JSON document that is capable of efficient querying. Meaning that if the whole collection reach more than 10M, its execution time is still less than 1sec. Example a query needs to fetch the last 100 added documents in the collection. Note: also include the index json and the index type, and assume that the collection is non capped collection type

# ANSWER SHEET:

ANSWERS:

1. function sortItems1() {

const quickSort = (list) => {

if (list.length < 10) {

return list;

}

}

};

1. function sortItems2() {

const insertSort = (nums) => {

for (let x = 1; x < nums.length; x++) {

let y = x -1

if (y = nums[x]) {

return nums;

}

}

}

}

1. function sortItems3(leftSide, rightSide) {

let x = 0;

let y = 0;

let output = [];

while (x < leftSide.length || y < rightSide.length) {

if (x === leftSide.length) {

output.push(rightSide[y]);

y++;

}

else if (y === rightSide.length || leftSide[x] <= rightSide[y]) {

output.push(leftSide[x]);

x++;

}

else {

output.push(rightSide[y]);

y++;

}

}

return output;

}

1. function compareItems() {

let x = 1;

let y = 1;

if ( x === y ) {

return (“The Same!”);

}

else {

return(“Not The Same”);

}

}

1. const timeCheck = callback => {

const output = callback();

return output;

}

console.log(timecheck(() => Math.pow(2, 10))+” ms”);

1. class Cylinder {

constructor(radius, height) {

this.radius = radius;

this.height = height;

}

getVolume() {

const r = this.radius;

const h = this.height;

const volume = Math.PI \* r \* r \* h;

return volume.toFixed(2);

}

}

const cylinder = new Cylinder(2, 4);

console.log(cylinder.getVolume()); // shows output “50.27”

1. class Phone {

constructor(brand) {

this.phonename = brand;

}

present() {

return ‘I have an ’ + this.phonename;

}

}

class Model extends Phone {

constructor(brand, mod) {

super(brand);

this.model = mod;

}

show() {

return this.present() + ‘, it is an ‘ + this.model;

}

}

const myPhone = new Model(“iPhone”, “iPhone 20 XD”);

console.log(myPhone.show()); // shows output “I have an iPhone, it is an iPhone 20 XD”

1. import React, { Component } from “react”

import ReactDOM from ‘react-dom’;

class Car extends Component {

constructor() {

super() // refers to the parent class

this.state = {

car: ‘Toyota’ // default state

}

}

changeMsg() {

this.setState({

car: ‘Honda’ // state after onClick event

})

}

render() {

return (

<div>

<p>{this.state.car}</p>

<button onClick={() => this.changeMsg()}>

Switch Car

</button>

</div> // shows current state

)

}

}

const Name = (props) => {

return (

<div>

<p>First Name: {props.name.firstName}</p>

<p>Last Name: {props.name.lastName}</p>

</div> // displays the value of the props

);

};

function App() {

const name = {

firstName: “Rye”,

lastName: “Koralzkie”

} // props values

return (

<div>

<Name name={name} />

<Car />

</div>

); // displays the final output including props and state

};

export default App;

1. import React, { useState } from “react”; // to import useState Hook from react

const [age, setAge] = useState(0); // default age value

const increaseAge = () => { setAge(age + 1 ) }; // increment age value

return (

<div>

<h1>

{age}

<br/>

<button onClick={increaseAge}> Increase Age </button>

</h1>

</div> // display age value and button to increase age

)

1. // Stateful Component: keeps track of changes in data.

class Main extends Components {

constructor() {

super()

this.state = {

fruits: []

}

}

render() {return <FruitsList fruits={this.state.fruits} />

}}

// Stateless Component: print outs what data is given to it from props.

const FruitsList = ({fruits}) => {

return (<ul> {fruits.map(fruit => { return <li>fruit</li>})} </ul>)};

1. let recordsMongo = require(‘../models/records.mongo’);

let moment = require (‘../node\_modules/moment/moment’);

function getAverageLikes (req, res) {

const { postTitle } = req.body;

const time = moment(‘${timestamp}’).format(‘ddd’+’,’+’MMM’+’’+’YYYY);

const averageLike = ((‘likesDay1’ + ‘likesDay2’)/2);

recordsMongo.find({post: postTitle}).then(async (post) => {

if (post.length) {

await res.status(200).json(post(averageLike));

}

}).catch ( err => { console.error(‘Error:’, err);

});

}}

module.exports = {getAverageLikes} ;

12. db.collection.aggregate([

{

$match: { smsCount: {$lte: 132}}

},

{

$group: { \_id: “$category”, totalSMS: { $sum: “$smsCount”} }

}

])

13. function updateCount (req, res) {

const { code } = req.body;

const newCount = ${count} + 1;

fruitMongo.find({code: code}).then(async (apple) => {

if (apple.length) {

await fruitMongo.updateOne(

{count: newCount}

).exec();

return res.status(200).json({status: true, message: ‘Count updated!’’});

} else {

return res.status(200).json({status: false, message: ‘Code not found!’ });

}

});

}

module.exports = {updateCount} ;

14. let recordsMongo = require(‘../models/records.mongo’);

function getSeatReservations (req, res){

const { email } = req.body;

recordsMongo.find({reservedTo: email}).then(async (movies) => {

if (movies.length) {

await res.status(200).json(movies);

}

else {

return res.status(200).json({

status: false,

message: ‘No Reservations Found’

});

}

}).catch ( err => {

console.error(‘Error: ’, err)

});

}

module.exports = {getSeatReservations} ;

15. let moviesMongo = require(‘../models/movies.mongo’);

let moment = require (‘../node\_modules/moment/moment’);

const schedule = require(‘node-schedule’);

schedule.scheduleJob(’\* \* \* \* \* \*’, () -> {

const currentTime = moment().format(‘x’);

moviesMongo.find({\_id: $maxID, date: { $lte: currentTime} }).sort(

{\_id:-1}).then(async(movies) => {

if (movies.length) {

await res.status(200).json(movies);

}

}).catch ( err => {

console.error(‘Error: ’, err)

});

}