National University of Computer and Emerging Sciences, Lahore Campus

Course	Web Programming (CS 4032)	Semester	Fall 2022
Program	BS (Computer Science)	Duration	3 Hours
Date	21 - December - 2022	Total Marks	60
Exam Type	Final	Page(s)	15
Section	ALL	Weightage	40

Roll Number: 19L-2313 **Section:** CS-7B

Important Instructions:

- The quality of the code will affect the marks.
- Students will receive **ZERO** marks if the answers are plagiarized.
- Use of **mobile phones, internet,** and **ANY type of smart devices** during the exam is strictly prohibited.
- This is an **open notes and open books** exam.
- Discussion with other students is not allowed.
- Exchange of notes, books, and stationery with other students are not allowed.

If any of the above rules is violated by the student. The invigilator has the right to file DC case against that student and the invigilator also has the right to take your exam away and ask you to leave the exam hall.

Question No	*1*	*2*	*3*	*Total*
Maximum Marks	20	30	10	60
Marks Obtained	20	28	5.5	48.5

Question 1

Consider the web page below.

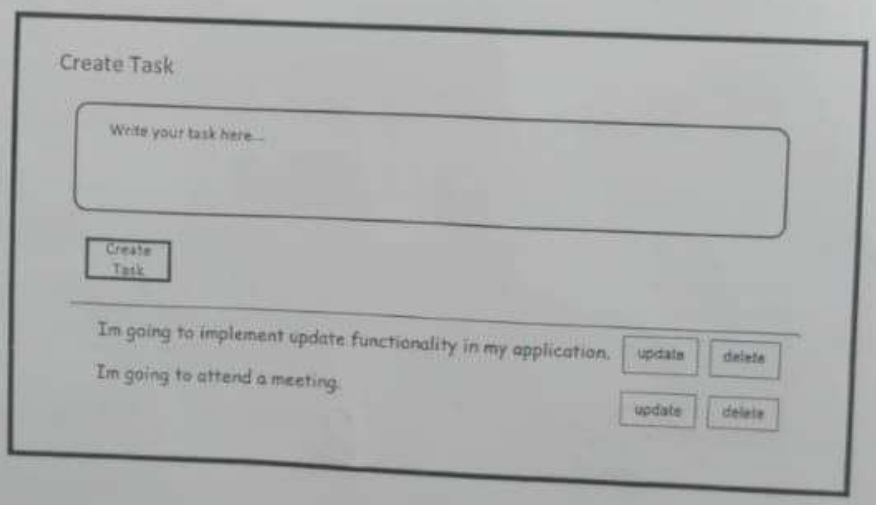
Part A. You must implement the following features in ReactJS code:

1. One label "Create Task"
2. One text area, that must be implemented using the forms
3. One button "Create Task" to add the task to add data(task) in database
4. A display area, that shows the list of tasks
5. Each task in the display area, must have three other elements in it:
   a. One text display that shows the task
   b. One update button, to update the task
   c. One deletes button to delete that task

Remember: You also must implement the functionality of create task, update, and delete button.

Part B. After completing the part A, you must implement the following features in NodeJS code:

1. One API/Route to update the task
2. One API/Route to delete a task
3. One API/Route to get the list of all tasks
4. One Express server + MongoDB Connectivity with mongoose



Update.js

```
import React from "react";
const Update=(props)=> {
return (<button type="button" className=
"ui positive basic button" onClick={props.handle
Click}>

Update </button>
export default Update;
```

Delete.js

```
import React from "react";
const Delete=(props)=> {
return (< button type="button" className
= "ui negative basic button" onClick={props.
handleClick}> Delete </button>
export default Delete;
```

Create.js

```
import React from "react";
const Create = (props) => {
return (<button type="~~button~~ submit" classNa
me
="ui positive basic button" onClick={props.
handleClick}> Create </button>
export Default Create.js
```

index.js

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import {useState ,useEffect} from 'react'
import Update from './Update'
import Delete from './Delete'
import Create from './Create'
import "semantic-ui"
const root = ReactDOM.createRoot(document.
getElementById('root'));
```

```
const App=()=>{
  const [text, setText] = useState('Write your task rem.
  const handleOnChange = (event)=>{
    setText(event.target.value)
  }
  Const [data, setData]=useState(null)
  Const submitHandler = asyn(e)=>{ e.preventDefault()
    try{
      const {text} = await axios.post('/api/text',{
      text})
    catch(err){
      console.log(err)}}


  return(<form onSubmit={submitHandler}>
      <label> Create Task </label>
        <textarea ... text" value={text} onChange=
        {handleOnChange} rows="8"> </textarea>
        </Create>
        </form>
        <hr>
        {data && data.forEach(i=>{
        <p> {i.text} </p>
        < Update handleClick={update(id)} />
        < Delete handleClick={Delete(id)} />
        })}
      );
useEffect(()=>{ fetch("/api/text") then (response
    => response.json() then(data=> {setData(data)}})
    ,[data]) ;
root.render(<App/>)
```

Notes: handles are on last page

TextModel.js

```
const mongoose=require("mongoose");
const schema=mongoose.Schema;
const TextSchema=new Schema({
  text:String});
module.exports=mongoose.model('Text', Schema);
```

TextController.js

```
const express=require("express")
const router=express.Router();
router.get('/', (req, res)=>{
        Text.find({}).exec(
function(err, text){
    if(err){ res.send("error occured")}
    else{ res.json(text)
        }})})
router.post('/', (req,res)=>{ var newText=newText(
        newText.text=req.body.text;
        newText.save(function(err, text){
            if(err){ res.send('error saving text')
                }else{ res.send(text)
                }})})
router.put('/:id', function(req, res){
    Text.findOneAndUpdate({_id: req.params.id},
        (function(err, text){
        if(err){ res.send('error saving Text')
            }else{ res.send(text)}})})})
```

```
router.delete('/:id',
(req, res)=>{
~~Book~~ Text.findone
AndRemove({
  _id: req.params.id
  }, (err, ~~book~~text)
=>{ if(err)
  res.send("Error");
  }
  else{
    res.status(204)
  }
  }))})
module.exports=
  router;
```


on last Page;

Question 2 (Marks: 30)

Consider a cloud-based Food delivery application (FDA), which intends to organize and analyze customer order information, for improved service delivery. For this scenario, we are creating help center for FDA. For simplicity, consider the following information for feedback:

- **CustomerId:** The id of the client website where the data originated.
- **OrderId:** The id of the order, that the customer has placed.
- **AgentId:** The id of the agent, with whom the customer is communicating.
- **Message:** A descriptive content that customer can specify
- **Feedback:** customer's feedback for the agent.

Show how to implement the above function for the webservice based on MERN Stack. The function will accept as **POST** request as JSON message, comprising the information mentioned above, parse and store the data in a database, and return a response message indicating success or failure.

1. Develop request and response message structures with proper algorithms (10 points)
2. Server-side code with proper MVC architecture (10 points)
3. Work out a client for this webservice with proper authentication and relevant web forms. (10 points)

Algorithm:

The above problem can be handle by creating sessions. Relation b/w different tables

→ When user clicks on feedback session created

→ user give feedback then everything store into database.

Backend

Models

Agent.js

```
const mongoose=require
("mongoose");
const AgentSchema=new
mongoose.Schema({
AgentName:string
})
const Agent=mongoose.
model('Agent',AgentSchema);
module.exports=Agent;
```

Customer.js

```
const mongoose =
require('mongoose')
const CustomerSchema
=new mongoose.Schema
({ CustomerName: String
})
const Customer=
mongoose.model(
'Customer' CustomerSchema),
module.exports = Customer
```

Order.js

```
const mongoose
= require("mongoose")
const OrderSchema
=new mongoose.
Schema({
OrderName: String
})
const Order= mongoose
.model('Order',
OrderSchema);
module.exports=
Order;
```

Feedback.js

```
const mongoose =require("mongoose");
const FeedbackSchema =new Schema ({ Message:{
type:String, required:true},
Feedback:{type:String, required :true},
CustomerId:{ type:mongoose.Schema.ObjectId,
ref:'Customer',
required:true},
AgentId:{type:mongoose.Schema.ObjectId, ref:'Agent',
required :true},
OrderId:{type : mongoose.Schema.ObjectId, ref:
'Order', required :true})}
const Feedback=mongoose.model('Feedback', FeedbackSchema);
module.exports= Feedback;
```

Controllers

Feedback Controller.js

```
const express= require ("express")
const Feedback = require("./../Models/Feedback)
const router = express.Router()
router.post ('/', async (req,res)=>
  { try {
   const customerId = (await Customer.findOne
   ({name:req.body.customer.name})._id
   const OrderId= (await Order.findOne
   ({name: req.body.Order.name})).id
   const AgentId = (await Agent.find One
   ({name: req.Agent.body.AgentName})._id

   const newFeedback = awaiit Feedback.create
   ({CustomerId, OrderId, AgentId, req.body.
     Message, req.body.Feedback});
   }
   catch(err) { console.log(err); });
module.exports= router;
```

Session Controller.js

```
Const express= require ("express")
const session = require("express-session")
const router = express.Router()
get ('/', (req, res)=> {
   req.session.name= "feedback"
   res.send ("Session-set")
   })
   get ('/destroy', (req, res)=> {
   req.session.destroy (function (err){ console.log
   (session destroy)) })
module.exports= router;
```

Index.js

```
const express= require ("express")
Const session = require ("express-session")
const bodyparser = require (Body-Parser)
const app= express()
let Port = 8000

app.use (session({ secret: "Muskil Sawal 1@!"
        resave :true,
        SaveUninitialized :true 3}))
app.use (bodyParser());

app.use("/session", Session Controller)
app.use ("/Feedback", Feedback Controller);

app.listen (Port, () => { console.log (" Server running
on port: 8000")});
```

View:

Index.js import React from 'React', import ReactDom from 'react-dom/client'

```
const App=() => {
  Const [feedback, setFeedback] = useState(fa
  const [data, setData] = use State(null)
  const [ctxt, setctxt] = use State("")
  const [ordertxt, setordertxt] = useState("")
  use Effect (() => {if(feedback)
    fetch('/session').then (data => console.log
      (data))}, [feedback])
  const submithandler = asyn(e) =>
    {
      e.preventDefault()
      try {
        await axios.post('/Feedback'
          , dataId, ctext, :
          ctext, ordertxt
          })
        awgit axios.get('/session/destroy')
        redirect("/")
      }
      catch(error)
      {
        console.log(error)
      }
  const [message, setmessage] = use State("")
  const [feedback, setfeedback] = use State("")
```

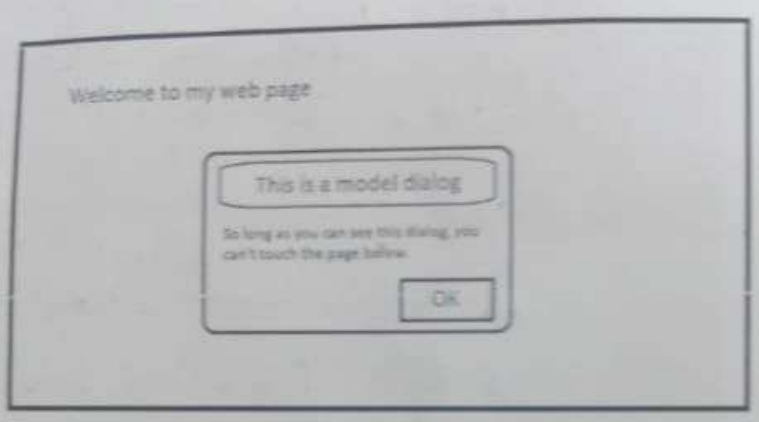
```
return(
<button type="button"  onClick={() => setFeedback
                                (true)}> </button>

{feedback && data
    <form  onSubmit={submithandler}
      <label> Agent ~~Name~~ Id: </label>
        <h1> {data.Id} </label>
      <label> Customer Name: </label>
      <textarea value={ctxt}> </textarea>
      <label> Order Name: </label>
        <textarea value={ordertxt} </textarea>
        ~~<button type=submit> </button>~~
      <label> Message </label>
        <textarea value = {message}>
            </textarea>
      <label> Feedback </label>
        <textarea value = {feedback}}
            </textarea>
      <button type="submit"> </button>
)
```

Question 3 (Marks 3)

Part A.

Consider the web page below. You must write jQuery for the below smarter dialog such that its size is not resizable, the dialog is draggable and when the user clicks on the button 'ok', the dialog closes. So, the user can now interact with the web page.




```
$(document).ready(function(){
    $("example").dialog({
        modal: false,
        resizeable: ~~true~~ false,
        draggable: true,
        button: [{
            text: "OK",
            click: function(){
                $(this).dialog("close");
            }}],
        text: "This is model dialog"
    })
})
```

Part B

(Marks 3)

Write a program using the JavaScript event handler only such that when the web pages get completely loaded. It changes the text color of a heading to purple first and right after that, the background color of text also changes to yellow.

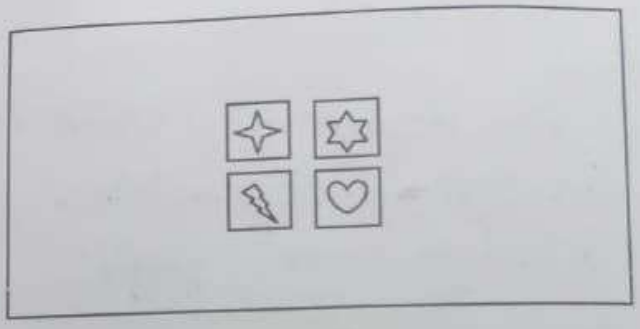
Zero

```
window.onload=fun;
fun(){
const h1= document.getElementById("heading");
    h1.style .backgroundColor= "yellow";
    h1.style.color= "purple";
}
```

(Marks 4)

Part C

Write a program (HTML + CSS) to display similar output on a browser. The web page has four images. The positioning of the images should implement, using CSS Layout – position as shown in the above image. You must use class selector to implement CSS for each image.



```
<html>
  <head>
    <title> Part C </title>
    <link rel="stylesheet" src = "style.css">
  </head>
  <body>
    <img src="image1.jpeg" class="star">
    <img src = "image2.jpeg" class = "star-side"/>
    <img src = "image 3.jpeg" class = "lightening"/>
    <img src = "image4.jpeg" class = "heart"/>

  </body>
</html>
```

Style.css

- Star {
  position: fixed
  left: 40%
  top: 40%
  }

- six-side {
  position: fixed
  right: 40%
  top: 40%
  }

- ligthening {
  position: fixed
  bottom: 40%
  left: 40%
  }

- heart {
  position: fixed
  bottom: 40%
  right: 40%
  }

```
const express= require ('express')
Const TextController = require("./TextController")
const bodyParser = require ("{bodyParser")
const mongoose = require("mongoose");
const app= express ()
app.use (bodyParser ());
app.use ('/api/text', TextController);
mongoose.connect (DB_URL, {useNewURLParser:true}
) .then (() => console.log
("DB connected successfully"));
const server = app.listen (8000, () => {
console.log ("App running on Port 8000");
});
```

Handles

```
const Update = (id, text) => {
try {
const {text} = awaiit. axios.put ('/api/text/${id}
/, {
_id: id
text
})}
catch (err)
{
console.log (err)
}
}}
const Delete = async (id) => {
try {
const status = await axios.Delete
('/api/text/${id}/, {_id: id})}
catch (err){
console.log (err)}}
```