Design Of Distributed System

HTML CSS JS

Task 1
In which architecture there are no distinguished clients and servers? Tick all correct answers.

- Distributed component architecture √
- Peer-to-peer architecture √
- Client Server
- Multi-tier client-server architecture

Task 2
Which of the following statements is correct?

- The original description of HTML was drafted by the World Wide Web Consortium(W3C) √
- IETF is the main organisation for producing web standards √
- The use of industry standards alone automatically results in fully standardisation solution
- RFC 2616 specifies that methods in HTTP/1.1 must be capitalised

Task 3
Which is the task of a Common Gateway Interface:

- To download and stream media files
- To execute applications on the client side
- To generate dynamic web pages √
- To communicate between server and client √
- To generate executables from web content by web server

Task 4
What is used to differentiate between the different network services of a host computer?

- Variables
- Service Name
- Ports
- Sockets √
- IP address √

Task 5
Which of the following statements are true in the context of RPC?

- The message format of RPC is text-based
- RPC provides developers a familiar programming models by extending the local procedure calls to a distributed environment √
- RPC is a asynchronous operation
- RPC is initiated by only the client √

Task 6

What is a URI? Write down its syntax.

Task 7

For each list entry, write a CSS rule that realises the following changes:

1. Change the background color of all <p> elements which are located inside of a <section> element with the class name "class-if" to color #6a8a26

Answer:

```
p . class-if {
   Background: #6a8a26;
}
```

2. For all <p> and <h3> elements, set their outer spacing to 15px and inner spacing to 5px.

Answer:

```
   p h3 {
   margin: 15px;
   padding: 15px;
}
```

3. All <img> elements with the class name "border" should have a solid black border with 3px width.

Answer:

```
.border {
   border: 3px solid black;
}
```

4. All Elements with class name "hidden" should no longer be displayed. This must not affect other elements or the layout of other elements.

Answer:

```
.hidden {
   display: block;
}
```

Task 8

For each list entry write a JavaScript expression that realises the following tasks:
Hint: only write pure JavaScript. Do not use jQuery functions!

1. Set the text content of the <h1> element that is assigned with ID "main" to "MAIN HEADING".
2. When any button on the page is clicked, the function registerClick should be called.
3. Create a variable "user" which contains an object with the attributes, "name" and "isAdmin". Set the attribute values to "John" and false.
4. When the variable "isAdmin" is true, the function displayAdminPanel should be called.
5. The variable "userlist" contains a list of all users, as described before in subtask 3. Print the name of each user from the list "userlist" in the console.

Answer:
1.
```
document.getElementById("main").innerHTML = ‘MAIN HEADING’
```
2.
```
document.querySelector("button").onclick = function() {registerClick()}
```
3.
```
const user = {name: "John", isAdmin: false}
```
4.
```
if(user.isAdmin === true){
        registerClick()
}
```
5.
```
let userlist = [
  {
    name: "john",
    isAdmin: false
  },
  {
    name: "Ebil",
    isAdmin: true
  }
]
for(let i = 0; i< userlist.length; i++){
        console.log(userlist[i].name)
}
```

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry

[page content]
```

Then, with every subsequent request to the server, the browser sends all previously stored cookies back to the server using the Cookie header.

```
GET /sample_page.html HTTP/2.0
Host: www.example.org
Cookie: yummy_cookie=choco; tasty_cookie=strawberry
```

Set-Cookie: id=a3fWa; Expires=Thu, 31 Oct 2021 07:28:00 GMT;

GET Request:

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

GET Response:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Vary: Authorization,Accept
Accept-Ranges: bytes
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

DELET Request:

```
DELETE /gbook?id=2hello.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Connection: Keep-Alive
```

DELETE Response:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
```

```
Connection: Closed

<html>
<body>
<h1>URL deleted.</h1>
</body>
</html>
```

Ajax

```
<html>
<body>
<div id="demo">
<h1>The XMLHttpRequest Object</h1>
<button type="button"
onclick="loadDoc('ajax_info.txt', myFunction)">Change Content
</button>
</div>
<script>
function loadDoc(url, cFunction) {
  var xhttp;
  xhttp=new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      cFunction(this);      document.getElementById(‘demo’).innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", url, true);   xhttp.open("DELETE", url+”?id=”+id, true);
  xhttp.send();
}
function myFunction(xhttp) {
  document.getElementById("demo").innerHTML =
  xhttp.responseText;
}
</script>
</body>
</html>
```

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 403: "Forbidden" 404: "Page not found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

```
<!DOCTYPE html>
<html>
<body>
<div id="demo">
<h1>The XMLHttpRequest Object</h1>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>
<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>
</body>
</html>
```

Web Components

index.html

```
<html>

<head>

</head>

<body>

<h3>Web components basics</h3>

<user-card name="Md Tajul Islam" avatar="my.jpg">

   <div slot="email">email@gmail.com</div>

   <div slot="phone">555-55-555</div>

</user-card>

<user-card name="Md Tajul Islam" avatar="my.jpg">

   <div slot="email">email@gmail.com</div>

   <div slot="phone">555-55-555</div>

</user-card>

<script src="userCard.js"></script>

</body>

</html>
```

userCard.js

```
const template = document.createElement('template');

template.innerHTML = `

<style>

   .user-card{

       font-family: 'Arial', sans-serif;

       background: #f4f4f4;

       width: 50px;
       display: grid;

       grid-template-columns: 1rf 2fr;

       grid-gap: 10px;

       margin-bottom: 15px;

       border-bottom: darkorchid 5px solid;

   }


   .user-card img{

       width: 100%;

   }

   .user-card button{

       cursor: pointer;

       background: darkorchid;

       color: #fff;

       border: 0;

       border-radius: 5px;

       padding: 5px 10px;

   }

</style>


<div class="user-card">

   <img/>

   <div>

       <h3></h3>

       <div class="info">

           <p><slot name="email"/></p>

           <p><slot name="phone"/></p>

       </div>
       <button id="toggle-info">Hide info</button>

   </div>

</div>

`;


class userCard extends HTMLElement {

   constructor() {

       super();

       this.showInfo = true;


       this.attachShadow({

           mode: 'open'

       });

       this.shadowRoot.appendChild(template.content.cloneNode(true));

       this.shadowRoot.querySelector('h3').innerText =
this.getAttribute('name');

       this.shadowRoot.querySelector('img').src = this.getAttribute('avatar');

   }

   toggleInfo() {

       this.showInfo = !this.showInfo;

       const info = this.shadowRoot.querySelector('.info');

       const toggleBtn = this.shadowRoot.querySelector('#toggle-info');


       if (this.showInfo) {

           info.style.display = 'block';

           //info.style.display = 'block';

           toggleBtn.innerText = 'Hide Info';

       } else {

           info.style.display = 'none';
```

```
            toggleBtn.innerText = 'Show Info';

        }

    }


    connectedCallback() {

        this.shadowRoot.querySelector('#toggle-info').addEventListener('click',
() => this.toggleInfo());

    }

    disconnectedCallback() {

        this.shadowRoot.querySelector('#toggle-info').removeEventListener();

    }

}


window.customElements.define('user-card', userCard);
```

Web Sockets

index.html

```
<!DOCTYPE html>

<html>

<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>
```

```
<script>

   const ws = new WebSocket("ws://localhost:8082");


   ws.addEventListener("open",() => {

       console.log("We are connected!!");


       ws.send("I am client sending data to  server!!")

   })

</script>

</body>

</html>
```

Server - index.js

```
const webSocket = require("ws");


const wss = new webSocket.Server({port: 8082});


wss.on("connection", ws => {

   console.log('New client connected!!')


   ws.on("message", data =>{

       console.log(`Client send data to server: ${data}`);

   });


   ws.on("close", () =>{

       console.log("Client has disconnected!!");

   });

});
```