descreva o significado dos conceitos backend e frontend no desenvolvimento de uma aplicação web

Em uma aplicação web, os termos "frontend" e "backend" referem-se a diferentes partes do sistema que trabalham juntas para criar uma experiência de usuário completa. Aqui está uma descrição de cada um:

1. \*\*Frontend\*\*: Também conhecido como "cliente", o frontend é a parte da aplicação web que o usuário vê e interage. Ele é construído usando tecnologias como HTML (para estruturação de conteúdo), CSS (para estilização e layout) e JavaScript (para interatividade e funcionalidade). Ele também pode incluir frameworks e bibliotecas JavaScript mais avançadas, como Angular, React e Vue.js, que ajudam a construir interfaces de usuário mais ricas e interativas. O frontend é executado no navegador do usuário e lida com a apresentação dos dados ao usuário e a coleta de entradas do usuário.

2. \*\*Backend\*\*: Também conhecido como "servidor", o backend é a parte da aplicação web que o usuário não vê, mas que desempenha um papel crucial para garantir que a aplicação funcione corretamente. Ele é responsável por processar as solicitações do frontend, interagir com bancos de dados para armazenar e recuperar dados, realizar cálculos complexos e fornecer respostas para o frontend. O backend é construído usando linguagens de programação como JavaScript (Node.js), Python, Java, Ruby, PHP, entre outras, e também envolve o uso de sistemas de gerenciamento de banco de dados como MySQL, PostgreSQL, MongoDB e outros. O backend é executado em um servidor, possivelmente em um data center ou na nuvem.

Em resumo, o frontend e o backend trabalham juntos para formar uma aplicação web completa. O frontend coleta entrada do usuário e a envia ao backend para processamento. O backend processa a entrada, interage com o banco de dados conforme necessário e retorna uma resposta para o frontend, que então apresenta os resultados ao usuário.

No desenvolvimento web indique o que entende pelo termo MEAN stack

MEAN stack é um acrônimo que representa um conjunto de tecnologias comumente utilizadas no desenvolvimento web. Cada letra do acrônimo representa uma tecnologia específica:

M - MongoDB: é um banco de dados NoSQL orientado a documentos. Ele armazena dados em formato JSON-like, o que o torna flexível e escalável.

E - Express.js: é um framework web para Node.js. Ele fornece recursos para facilitar o desenvolvimento de aplicativos web e APIs RESTful, simplificando o gerenciamento de rotas, middleware e interações com o servidor.

A - Angular: é um framework front-end desenvolvido pelo Google. Ele permite a criação de interfaces de usuário dinâmicas e interativas, seguindo o padrão MVVM (Model-View-ViewModel).

N - Node.js: é um ambiente de tempo de execução JavaScript do lado do servidor. Ele permite a execução de código JavaScript fora do navegador, possibilitando a construção de aplicativos web completos no lado do servidor.

A combinação dessas tecnologias, MongoDB, Express.js, Angular e Node.js, forma a stack MEAN. Essas tecnologias são todas baseadas em JavaScript, o que permite o uso da mesma linguagem de programação tanto no lado do cliente quanto no lado do servidor. A MEAN stack oferece uma abordagem de desenvolvimento de ponta a ponta, permitindo que desenvolvedores construam aplicativos web completos, desde o banco de dados até a interface do usuário, usando tecnologias modernas e escaláveis.

As operações CRUD estão relacionadas com o desenvolvimento de vários componentes web. Indique do que se tratam estas operações e 2 casos de uso na programação em ambiente web

CRUD é um acrônimo para Create, Read, Update e Delete. Essas são as quatro operações básicas que você pode realizar em qualquer dado persistente.

1. \*\*Create\*\* (Criar): Refere-se à criação de novos registros. Na web, isso geralmente ocorre quando submetemos um formulário para adicionar um novo produto, postar um novo comentário, criar uma nova conta de usuário, entre outros.

2. \*\*Read\*\* (Ler): Refere-se à leitura e recuperação de registros. Por exemplo, visualizar um blog post, listar produtos em uma loja online, ver o perfil de um usuário etc.

3. \*\*Update\*\* (Atualizar): Refere-se à atualização de registros existentes. Por exemplo, editar as informações de perfil, atualizar a quantidade de um item no carrinho de compras, etc.

4. \*\*Delete\*\* (Excluir): Refere-se à remoção de registros existentes. Por exemplo, deletar uma postagem de blog, remover um produto do carrinho de compras, deletar uma conta de usuário etc.

Essas operações são a base de qualquer aplicação web que interaja com um banco de dados ou outra forma de armazenamento de dados persistente. A maioria, se não todas, as aplicações web realizam algum tipo de operações CRUD.

Aqui estão dois exemplos de casos de uso em programação de ambiente web:

1. \*\*Gestão de usuários em uma plataforma de mídia social\*\*: As operações CRUD podem ser usadas para criar uma nova conta de usuário (Create), visualizar o perfil de um usuário (Read), atualizar as informações do perfil do usuário (Update), e excluir uma conta de usuário (Delete).

2. \*\*Gestão de produtos em uma loja online\*\*: As operações CRUD podem ser usadas para adicionar um novo produto ao catálogo (Create), visualizar detalhes do produto (Read), alterar o preço ou detalhes do produto (Update), e remover um produto do catálogo (Delete).

o uso de padroes de software de desenvolvimento de aplicações de web +e considerada uma boa pratica. Considere o padrao de software MVC. Descreve em que consiste este padrao e como o pode aplicar uma aplicação que utiliza ExpressJS

MVC (Model-View-Controller) é um padrão de design de software que divide uma aplicação em três componentes interconectados. Isso é feito para separar as preocupações internas de uma aplicação, tornando-a mais fácil de manter e entender. Aqui está o que cada componente faz:

1. \*\*Model (Modelo)\*\*: Representa os dados e as regras de negócios da sua aplicação. É responsável por ler e gravar dados, bem como transformá-los em informações úteis para a sua aplicação.

2. \*\*View (Visualização)\*\*: É a interface do usuário - aquilo que o usuário vê e com o que interage. A View pega os dados do Model e os apresenta de uma forma que faz sentido para a interação do usuário.

3. \*\*Controller (Controlador)\*\*: Atua como um intermediário entre o Model e a View. Ele recebe a entrada do usuário através da View, processa essa entrada usando o Model e retorna os resultados para a View.

Agora, como você pode aplicar o padrão MVC em uma aplicação Express.js?

\*\*Modelo\*\*: No Express.js, o modelo é normalmente representado por módulos que interagem com um banco de dados ou com outras formas de armazenamento de dados. Você pode usar um ORM (Object-Relational Mapping) como Sequelize ou Mongoose para interagir com o banco de dados.

var mongoose = require('mongoose');

var UserSchema = new mongoose.Schema({

nome: String,

email: String

});

module.exports = mongoose.model('User', UserSchema);

```

\*\*Visão\*\*: No Express.js, a visão é normalamente composta por templates renderizados no servidor. Express suporta vários motores de template, como Pug, Mustache e EJS.

app.set('view engine', 'pug');

app.get('/', function(req, res){

res.render('index', { title: 'Express' });

});

```

\*\*Controlador\*\*: No Express.js, os controladores são representados por rotas. As funções middleware em suas rotas recebem os objetos de solicitação e resposta, e interagem com o modelo para processar a solicitação e enviar a resposta.

var User = require('../models/user');

app.get('/users', function(req, res) {

User.find({}, function(err, users) {

if (err) throw err;

res.render('users', { users: users });

});

});

```

Esta é uma explicação simplificada de como você pode aplicar o padrão MVC em uma aplicação Express.js. Dependendo da complexidade da sua aplicação, você pode querer dividir suas rotas e modelos em vários arquivos e diretórios para manter seu código organizado.

considere a segurança de aplicações web. Indique de forma detalhada como podemos adicionar autenticação e autorização a uma aplicação web que use a framework para o fronted e backend

Adicionar autenticação e autorização a uma aplicação web é fundamental para garantir a segurança dos dados e para controlar o acesso a certas partes da aplicação. Embora as implementações específicas possam variar dependendo das tecnologias e frameworks utilizados, aqui estão algumas diretrizes gerais para adicionar autenticação e autorização a uma aplicação web.

**Autenticação:**

A autenticação é o processo de verificar a identidade de um usuário. Uma forma comum de implementar a autenticação é através do uso de nomes de usuário e senhas.

1. **Crie um Formulário de Login:** O primeiro passo é criar um formulário de login onde os usuários podem inserir suas credenciais. Este formulário deve ser protegido contra ataques de Cross-Site Request Forgery (CSRF) usando tokens CSRF.
2. **Verifique as Credenciais:** Quando o formulário de login é submetido, o servidor deve verificar as credenciais fornecidas pelo usuário. Isso normalmente envolve a busca do nome de usuário na base de dados e a comparação da senha fornecida com a senha armazenada. A senha armazenada deve ser uma hash segura da senha original, para proteger as senhas dos usuários no caso de uma violação de dados.
3. **Crie uma Sessão ou um Token:** Se as credenciais estiverem corretas, o servidor cria uma sessão ou um token que representa a sessão do usuário. O ID da sessão ou o token é então enviado de volta para o cliente.
4. **Configure Cookies ou Local Storage:** No lado do cliente, o ID da sessão ou o token pode ser armazenado em um cookie ou no local storage. O cookie ou o token deve ser enviado com cada solicitação subsequente para que o servidor possa identificar o usuário.

**Autorização:**

A autorização é o processo de verificar se um usuário autenticado tem permissão para realizar uma determinada ação.

1. **Defina os Papéis do Usuário:** Defina diferentes papéis de usuário (por exemplo, usuário regular, administrador, moderador, etc.) e quais ações cada papel pode realizar.
2. **Verifique as Permissões:** Cada vez que uma solicitação é feita ao servidor para realizar uma ação que requer permissões, o servidor deve verificar se o usuário autenticado tem o papel necessário para realizar a ação. Isso pode ser feito verificando o papel do usuário na base de dados.
3. **Proteja as Rotas:** Certifique-se de que as rotas que requerem permissões específicas são protegidas e só podem ser acessadas por usuários com as permissões adequadas.

Note que essas são apenas diretrizes gerais. A implementação específica pode variar dependendo do framework e da biblioteca que você está usando. Além disso, é importante lembrar que a segurança é um campo complexo e em constante mudança, e que a autenticação e autorização são apenas duas peças do quebra-cabeça. Outras considerações de segurança, como criptografia, segurança de dados, proteção contra ataques de força bruta e injeção SQL, entre outros, também são muito importantes.

observe o excerto de código abaixo que representa uma aplicaçao web. Escrava o conteúdo do ficheiro "demo.html" de forma a o exemplo ficar funcional e apresente a mensagem de "Sucess" caso a informação de login seja valida e "Failure" para os restantes casos

const expres = require("express");

const path = require("path");

const bodyParser = require("body-parser");

const app = express();

app.use(bodyParser.urlencoded({ extended: false }));

app.get("/", function (*res*, *req*) {

*res*.sendFile(path.join(\_dirname, "demo.html"));

});

app.post("/result", function (*req*, *res*) {

  const username = *req*.body.username;

  const password = *req*.body.password;

  if (username === "paw@estg" && password === "paw2022") {

*res*.send("Sucess");

  }

*res*.send("Failure");

});

app.listen(3000);

---------------------------------------------------------

Com base no excerto de código que você forneceu, um exemplo de um arquivo "demo.html" que irá funcionar corretamente pode ser assim:

<!DOCTYPE html>

<html>

<head>

    <title>Login</title>

</head>

<body>

    <form action="/result" method="post">

        <label for="username">Username:</label><br>

        <input type="text" id="username" name="username"><br>

        <label for="password">Password:</label><br>

        <input type="password" id="password" name="password"><br>

        <input type="submit" value="Login">

    </form>

</body>

Este é um simples formulário HTML com dois campos, username e password. O atributo action no elemento form está definido como "/result", que é a rota que a aplicação Express.js está à espera de receber os dados do formulário através do método POST.

Quando o usuário submete o formulário, os dados são enviados para o servidor, que então verifica se as credenciais são válidas. Se o nome de usuário for "paw@estg" e a senha for "paw2022", o servidor irá responder com a mensagem "Sucess". Para todas as outras credenciais, irá responder com "Failure".

observe o excerto de código abaixo de uma aplicação Angular. Indique qual a sua utilidade e funcionalidade. Como pode ser utilizados por componentes em Angular e que padroes de software estão a ser utilizados para gerir os pedidos HTTP.

const endpoint = "http://localhost:3000/api/v1/"

const httpOptions = {

     headers: new HttpHeadrs({

         "Content-Type" : " application/json"

    })

}

@Injectable({

   providedIn: "root"

})

export class RestService{

    constructor (private *http*:HttpClient) {}

    getProduct(*id*:string) : Observable<Product>{

          return this.http.get<Product>(endpoint+"product/"+*id*)

   }

  addProduct(*product*:Product):Observable<Product>{

           return this.http.post<Product>(endpoint+"products", *product*, httpOptions)

   }

   getProductList():Observable<Product[]>{

          return this.http.get<Product[]>(endpoint+"products", httpOptions)

    }

}

Este é um exemplo de um serviço no Angular, que é uma classe com um propósito bem definido, seguindo o princípio da responsabilidade única (um dos princípios SOLID). Serviços são comumente usados para compartilhar métodos e propriedades comuns entre vários componentes, facilitando a reutilização de código e a manutenção.

No código fornecido, `RestService` é um serviço que manipula todas as operações HTTP relacionadas a produtos. Ele tem três métodos:

1. `getProduct(id: string)`: faz uma solicitação GET para obter um produto específico pelo ID.

2. `addProduct(product: Product)`: faz uma solicitação POST para adicionar um novo produto.

3. `getProductList()`: faz uma solicitação GET para obter uma lista de todos os produtos.

Estes métodos retornam um Observable. No Angular, Observables são usados para lidar com operações assíncronas. Eles são uma parte importante da biblioteca RxJS (Reactive Extensions for JavaScript), que é amplamente usada no Angular.

Os serviços em Angular são geralmente injetáveis, como demonstrado pelo decorador `@Injectable`. Isso significa que eles podem ser injetados em outras partes da aplicação (como componentes) usando a injeção de dependência do Angular. Isso torna o código mais modular e testável.

Quanto ao padrão de software que está sendo usado para gerenciar solicitações HTTP, este é o padrão Service. Em aplicações Angular, serviços são comumente usados para encapsular a lógica de interação com dados, como solicitações HTTP. Este serviço encapsula a lógica de realização de solicitações HTTP para um endpoint de API e provê uma interface simples para componentes recuperarem ou alterarem dados.

Por fim, a constante `httpOptions` é usada para definir as opções da solicitação HTTP. Neste caso, o cabeçalho 'Content-Type' é definido como 'application/json', que é o tipo de mídia comum para solicitações de API REST.

Observe o excerto de código e descreva de forma sucinta a funcionalidade implementada na pagina HTML. Caso identifique erros na pagina indique como podem ser resolvidos

<!DOCTPE>

html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1" />

  </head>

  <body>

    <div id="formulario">

      <input id="arg1" type="number" /> <br />

      <input class="arg2" type="text" /> <br />

      <button>Multiplica</button>

    </div>

    <div id="resultado"></div>

    <script>

      function multiplicar() {

        let a1 = documento.getElementById("arg1").value;

        let a2 = documento.getElementById("arg2").value;

        let res = a1 \* a2;

        document.getElementByID("resultado").innerHTML = "Resultado" + res;

      }

      document

        .getElementByID("calcular")

        .addEventListener("click", multiplicar, false);

    </script>

  </body>

</html>

O excerto de código fornecido pretende implementar uma funcionalidade simples de multiplicação entre dois valores inseridos pelo usuário. A ideia é capturar os valores inseridos nos campos de input, multiplicá-los e exibir o resultado na div com o id "resultado".

No entanto, existem alguns erros no código que precisam ser corrigidos para que a funcionalidade funcione conforme esperado:

1. A tag de início DOCTYPE está com erro de digitação. Deve ser <!DOCTYPE html> em vez de <!DOCTPE> html>.
2. O segundo campo de input tem uma classe arg2, mas no script está sendo referenciado como um id. Deveria ter id="arg2" em vez de class="arg2".
3. O método JavaScript getElementById está mal escrito como getElementByID e documento. A primeira letra "d" de "Id" deve ser minúscula e "document" é a forma correta.
4. O botão que deve disparar a função de multiplicação não tem o id "calcular", que é referenciado no script. O botão deve ter id="calcular".

Com as devidas correções, o código ficaria assim:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset ="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1" />

</head>

<body>

    <div id="formulario">

         <input id="arg1" type="number">

         <br>

        <input id="arg2" type="number">

        <br>

        <button id="calcular">Multiplica</button>

    </div>

    <div id="resultado">

     </div>

    <script>

             function multiplicar(){

                   let a1 = document.getElementById("arg1").value;

                   let a2= document.getElementById("arg2").value;

                   let res = a1 \* a2;

                  document.getElementById("resultado").innerHTML = "Resultado: " + res;

             }

           document.getElementById("calcular").addEventListener("click", multiplicar, false);

     </script>

</body>

</html>

Também notei que o segundo campo de input tinha type="text". Para realizar a multiplicação, este campo deve ser do type="number" para evitar que o usuário insira caracteres não numéricos.

indique as verdadeiras:   
a) HTML é uma linguagem de programação de web   
b) Css permite estruturar o conteudo de pagina web   
c) JS apenas permite formatar os elementos da pagina web   
d) É possivel submeter um formulário numa pagina HTML sem utilizar CSS ou JS

Aqui estão as respostas corretas:

a) Falso. HTML (HyperText Markup Language) não é uma linguagem de programação, mas sim uma linguagem de marcação utilizada para estruturar o conteúdo de páginas web.

b) Falso. CSS (Cascading Style Sheets) é usado para estilizar e fazer o layout de páginas web - basicamente, controla a aparência da página. Não é usado para estruturar o conteúdo (isso é feito com HTML).

c) Falso. JavaScript é uma linguagem de programação que permite adicionar funcionalidades interativas a páginas web. Embora possa ser usado para alterar a formatação e estilo dos elementos da página (de maneira similar ao CSS), suas capacidades vão muito além disso.

d) Verdadeiro. É absolutamente possível submeter um formulário em uma página HTML sem a necessidade de CSS ou JavaScript. O HTML por si só pode lidar com a criação de um formulário e a submissão de seus dados. O CSS seria usado apenas para estilizar o formulário, enquanto o JavaScript poderia ser usado para adicionar funcionalidades interativas, como validação de entrada de dados em tempo real.

indique as verdadeiras sobre template engines:

a) Angular possui um template engine que gera as paginas diretamente no browser, a partir dos componentes presentes na pagina apresentada ao utilizador

b) Podemos utilizar o template engine mongoose para gerar paginas HTML dinamicamente no servidor

c) O template engine EJS, num projeto gerado atraves do express generator, guarda as os templates dentro da pasta views do projeto

d) Para utilizar Template engines numa aplicação web, necessitamos de utilizador, obrigatoriamente APIs REST e operações CRUD

As respostas para as afirmações acima são as seguintes:

a) Verdadeiro. Angular é uma framework que utiliza um sistema de templates para gerar páginas HTML a partir de componentes. Este processamento é feito diretamente no browser do usuário.

b) Falso. Mongoose é uma biblioteca para trabalhar com MongoDB em Node.js, e não um template engine. Para a geração de páginas HTML dinamicamente no servidor, pode-se usar template engines como EJS, Pug (antes conhecido como Jade), Handlebars, entre outros.

c) Verdadeiro. No padrão de projetos gerados através do Express Generator, os arquivos de template (como os do EJS) são armazenados na pasta "views".

d) Falso. A utilização de template engines em uma aplicação web não necessita obrigatoriamente de APIs REST ou operações CRUD. Embora essas técnicas possam ser frequentemente encontradas juntas em aplicações web modernas, elas não são interdependentes. Template engines podem ser usados para gerar qualquer tipo de conteúdo baseado em templates, não apenas páginas web resultantes de operações CRUD ou APIs REST.

Atendendo as caracteristicas do framework ExpressJS, quais as verdadeiras e falsas:

a) ExpressJS é uma framework para desenvolver aplicações que correm exclusivamente no browser

b) O package manager npm pode ser utilizador para gerir as dependências da nossa aplicação

c) O ficheiro package json guarda contém, entre outros, informação sobre as dependências da nossa aplicação e informação sobre comandos para serem executados no terminal com a ferramenta npm

d) ExpressJS é uma framework que cria aplicações que devem correr com um runtime do JS com o NodeJS

As respostas para as afirmações acima são as seguintes:

a) Falso. ExpressJS é uma framework para o desenvolvimento de aplicações web do lado do servidor em Node.js, e não é executado exclusivamente no navegador (browser).

b) Verdadeiro. O npm (Node Package Manager) é uma ferramenta essencial no ecossistema Node.js/Express.js para gerir dependências de uma aplicação.

c) Verdadeiro. O arquivo package.json em uma aplicação Node.js/Express.js armazena várias informações sobre a aplicação, incluindo suas dependências (na seção "dependencies" e "devDependencies") e scripts que podem ser executados com o npm (na seção "scripts").

d) Verdadeiro. ExpressJS é uma framework usada para criar aplicações web que rodam no ambiente de runtime Node.js, permitindo a criação de APIs REST, servidores web, entre outros.

quais as verdadeiras e falsas sobre framework Angular:

a)Angular é uma framework para desenvolvimento de aplicares no backend

b) Apenas permite o uso de um componente por pagina/rota

c)Em angular nao é possível utilizar os formularios de Html com o elemento >form></form>

d) a linguagem de programação de uma aplicação angular é typescript

As respostas para as afirmações acima são as seguintes:

a) Falso. Angular é uma framework para o desenvolvimento de aplicações web do lado do cliente (frontend), não do backend.

b) Falso. Angular suporta o uso de múltiplos componentes em uma única página/rota. Esses componentes podem interagir entre si através de serviços, entradas/saídas, e outros mecanismos.

c) Falso. Angular permite a utilização de formulários HTML normais, bem como oferece sua própria abordagem de modelagem de formulários (formulários reativos e formulários template-driven) que oferecem mais funcionalidades e controle.

d) Verdadeiro. Angular é construído e normalmente escrito em TypeScript, uma linguagem de programação que é um superconjunto de JavaScript com a adição de tipos estáticos.

Indique as afirmações verdadeiras sobre o desenvolvimento de serviços REST:

a) em serviços rest a informação é trocada em formato binario entre cliente e servidor para melhor performance

b) Serviços rest não permitem o uso do formato XML para troca de informação entre cliente e servidor

c) é possivel usar os metodos HTTP como GET, POST, PUT, DELETE para mapear operaçóes CRUD sobre uma base de dados

d) O modulo swagger ui permite documentar e criar um ambiente de testes para APIs TEST na framework ExpressJS

As respostas para as afirmações acima são as seguintes:

a) Falso. Embora seja possível trocar informações em formato binário em um serviço REST, o mais comum é usar formatos textuais como JSON ou XML, pois são mais fáceis de manipular e debugar.

b) Falso. Embora JSON seja o formato mais comum utilizado em serviços REST, XML também pode ser usado. A decisão depende principalmente das necessidades e restrições do projeto.

c) Verdadeiro. Um dos princípios fundamentais do REST é a utilização dos métodos HTTP padrão para executar operações específicas. Normalmente, GET é usado para leitura, POST para criação, PUT para atualização e DELETE para remoção - mapeando assim para as operações CRUD.

d) Verdadeiro. Swagger UI é uma ferramenta popular para documentar e testar APIs REST. Ela pode ser integrada em muitas frameworks, incluindo ExpressJS no ambiente Node.js. Isso permite que desenvolvedores e usuários visualizem e interajam com a API de maneira fácil e intuitiva.

-------------------------------------

<% if (newList && newList .length >0 ) { %>

    <% newList.forEach(function(news)[ %>

          <div>

                 <p> <%= news.tile %> </p>

                 <p> <%= news.shortDescription %> </p>

                 <a href = '/news/<%=news .link %>'> Open </a>

          </div>

     <% }); %>

<% } % >

Tendo em conta o excerto de código acima, quais as verdadeiras e falsas:

a) o codigo é executado por um template engine no servidor gerando um ficheiro html que é enviado para o cliente

b) o código é enviado para o cliente e depois executado par amostrar a informação no browser

c) se a variavel newsList estiver vazia nenhuma informação é mostrada no browser do cliente

d) Independentemente do numero de news dentro da variável newsList, apenas a primeira é mostrada no browser do cliente

a) Verdadeiro. O código parece estar usando um mecanismo de template, como o EJS, para gerar dinamicamente um HTML no servidor com base nos dados da variável newList. Este HTML gerado é então enviado para o cliente.

b) Falso. O código não é executado no cliente. Ele é processado no servidor e o HTML resultante é enviado para o cliente.

c) Verdadeiro. Se a variável newList estiver vazia ou não definida, nenhum conteúdo será gerado e, portanto, nada será exibido no navegador do cliente.

d) Falso. O código usa um loop forEach para iterar sobre todos os elementos em newList e gerar um bloco de conteúdo para cada um. Portanto, se newList contiver mais de um item, todos serão mostrados no navegador do cliente.

…………………

var mangoose = require('mangoose');

var EmployeeSchema = new mangoose.Schema({

    name: String,

    address: String,

    position: String,

    classification:String,

    updated\_at: {type: Date, default:Date.now},

});

module.exports = mangoose.model('Employee', EmployeeSchema)

Tendo em conta o excerto de código acima, quais as verdadeiras e falsas:

a) é uma aplicaçoo desenvolvida na framework Angular

b) é uma aplicaçoo desenvolvida na framework ExpressJS

c) cria um modelo de dados e um objeto para interagir com uma coleção na base de dados

d) Representa a criação de uma classe de modelo de dados em typescript

a) Falso. Este é um excerto de código Node.js que usa Mongoose, uma biblioteca para MongoDB. Não é código Angular.

b) Verdadeiro. Apesar de não vermos nenhuma referência direta ao Express.js neste trecho, é comum usá-lo em conjunto com o Mongoose no desenvolvimento de aplicações em Node.js. Portanto, poderia ser parte de uma aplicação Express.js.

c) Verdadeiro. Este trecho define um "Schema" Mongoose para a coleção de "Employees" em uma base de dados MongoDB. Este Schema será usado para validar dados antes de serem salvos na base de dados e para construir objetos que representam documentos nesta coleção.

d) Falso. Este código não está escrito em TypeScript, mas em JavaScript. Além disso, ele não define uma classe, mas um esquema Mongoose.

-----------------------------------------------

@Injectable({

    providedIn: "root"

 })

 export class JWTInterceptorService implements HttpInterceptor{

    constructor() {}

    intercept(*req*:HttpRequest<any>, *next*: HttpHandler): Obervable<HttpEvent<any>>{

        let user = localStorage.getItem('currentUser')

        if(user){

*req* = *req*.colne({setHeaders:{"x-acess-token" : '${user}'}})

        }

        return *next*.handle(*req*)

    }

 }

Tendo em conta o excerto de código acima, quais as verdadeiras e falsas:

a) é executado sempre que existe um pedido http na aplicação

b) cria um token JWT e envia para o cliente

c) cria um header com o nome "x-acess-token" num pedido http

d) gere o acesso de um utilizador às todas de uma pagina web

a) Verdadeiro. Este serviço funciona como um interceptor HTTP. Isso significa que ele será executado sempre que um pedido HTTP for feito usando o HttpClient do Angular.

b) Falso. Este serviço não cria um token JWT. Em vez disso, ele recupera um token que já está armazenado no armazenamento local e o anexa ao cabeçalho de uma solicitação HTTP.

c) Verdadeiro. Se um usuário estiver atualmente armazenado no armazenamento local (ou seja, se o usuário estiver autenticado), o interceptor adiciona um cabeçalho chamado "x-access-token" ao pedido HTTP.

d) Falso. Embora este serviço seja parte importante do sistema de autenticação (adicionando o token JWT aos cabeçalhos de solicitação HTTP), não gere diretamente o acesso do usuário às diferentes partes de uma página web. Isso seria normalmente gerido por serviços de autenticação ou guardas de rotas no Angular.

Quais as verdadeiras e falsa  
a) Verdadeiro. O objeto XMLHttpRequest é uma API em JavaScript que é usada para trocar dados com um servidor. Isso permite a atualização de partes de uma página web sem a necessidade de recarregar toda a página.

b) Falso. Embora tecnicamente seja possível usar o método GET para enviar informações de autenticação para o servidor, isso é considerado uma prática ruim por razões de segurança. As informações enviadas através de um pedido GET são anexadas à URL e, portanto, podem ser facilmente interceptadas ou registradas em logs. Para dados sensíveis, como informações de login, o método POST é geralmente recomendado.

c) Falso. As aplicações Angular são executadas no navegador (client-side), não no ambiente de tempo de execução do NodeJS. No entanto, o Angular pode ser servido a partir de um servidor NodeJS, e as ferramentas de desenvolvimento de Angular, como o Angular CLI, utilizam NodeJS.

d) Falso. O método PUT em um serviço REST é usado para atualizar recursos existentes com dados enviados pelo cliente. Portanto, um serviço REST com o método PUT definitivamente permitiria o envio de informações de um cliente para o servidor.