Lab19

¿En qué consiste el control de acceso basado en roles?

Un ejemplo sería un Sistema de Gestión de Recursos Humanos (HRMS), donde los usuarios se dividen en roles como "Empleado", "Gerente de Departamento" y "Administrador de Recursos Humanos", y los permisos se asignan según estos roles.

Investiguen y describan 2 sistemas, uno que aplique RBAC y uno que no. Realicen un análisis de las ventajas y desventajas de cada uno con respecto al control de acceso.

Sistema con RBAC (Control de Acceso Basado en Roles):
En este sistema, las acciones de los usuarios se gestionan en función de los roles que tienen asignados. Cada usuario se asigna a un rol específico y luego se otorgan permisos a esos roles para acceder a determinados recursos o realizar ciertas acciones. Por ejemplo, en un sistema de gestión de recursos humanos, podrías tener los roles de "Administrador", "Gerente" y "Empleado", y cada uno tendría diferentes niveles de acceso y permisos dentro del sistema.

Sistema sin RBAC (Control de Acceso No Basado en Roles):
En un sistema sin RBAC, los permisos se asignan directamente a los usuarios individuales en lugar de a roles predefinidos. Esto puede llevar a una gestión más compleja de los permisos, ya que cada usuario puede tener configuraciones de acceso únicas. Por ejemplo, en un sistema de gestión de archivos sin RBAC, cada usuario tendría permisos específicos sobre archivos y carpetas determinados, sin importar su función o posición en la organización. Esto puede ser menos escalable y más difícil de administrar en comparación con un sistema basado en roles.

Lab 18

¿Qué otras formas de autentificación existen?

Autenticación de dos factores (2FA): Esta forma de autenticación requiere dos tipos diferentes de credenciales para verificar la identidad del usuario. Por lo general, esto implica algo que el usuario sabe (como una contraseña) y algo que el usuario posee (como un código generado por una aplicación de autenticación en su teléfono).

Autenticación biométrica: Esta técnica utiliza características físicas o de comportamiento únicas de un individuo para verificar su identidad. Ejemplos incluyen huellas dactilares, reconocimiento facial, escaneo de iris, reconocimiento de voz o incluso patrones de escritura.

Token de seguridad: Un token de seguridad es un dispositivo físico o una aplicación móvil que genera códigos de acceso únicos y temporales que se utilizan junto con una contraseña estática para autenticar al usuario. Estos códigos cambian periódicamente y son únicos para cada sesión de autenticación.

Autenticación basada en certificados: Esta forma de autenticación utiliza certificados digitales emitidos por una autoridad de certificación confiable. Los usuarios poseen un certificado digital que se utiliza para autenticar su identidad en lugar de una contraseña.

Autenticación de un solo uso (OTP): Similar a la autenticación de dos factores, la autenticación de un solo uso implica el uso de una contraseña temporal generada por una aplicación móvil, un dispositivo físico o enviado por mensaje de texto. Estos códigos son válidos solo para una sesión de autenticación y luego se vuelven inútiles.

Lab17

¿Qué ventajas tiene escribir el código SQL únicamente en la capa del modelo?

Escribir código SQL únicamente en la capa del modelo de una aplicación presenta varias ventajas:

Separación de preocupaciones: Mantener la lógica de acceso a la base de datos en la capa del modelo ayuda a separar las preocupaciones relacionadas con la presentación de los datos de las preocupaciones relacionadas con el acceso a los datos. Esto hace que el código sea más modular y fácil de mantener.

Reutilización de código: Al centralizar las consultas SQL en la capa del modelo, se pueden reutilizar en diferentes partes de la aplicación. Esto promueve la consistencia y reduce la duplicación de código.

Seguridad: Al limitar el acceso directo a la base de datos desde otras capas de la aplicación, se reduce la superficie de ataque para posibles vulnerabilidades de seguridad, como la inyección de SQL.

¿Qué es SQL injection y cómo se puede prevenir?

SQL injection es una técnica utilizada por los atacantes para comprometer bases de datos a través de la inserción de código SQL malicioso en las entradas de un formulario de una aplicación web. Cuando una aplicación web no valida correctamente las entradas de usuario antes de utilizarlas en consultas SQL, los atacantes pueden manipular esas entradas para ejecutar comandos SQL no deseados.

Para prevenir la inyección de SQL, se deben seguir algunas buenas prácticas:

Parámetros de consultas preparadas: Utilizar parámetros de consultas preparadas o consultas parametrizadas es la forma más efectiva de prevenir la inyección de SQL. En lugar de concatenar directamente las entradas de usuario en la consulta SQL, se utilizan marcadores de posición y luego se pasan los valores de los parámetros de manera segura.

Validación de entrada: Validar y sanitizar todas las entradas de usuario antes de utilizarlas en consultas SQL. Esto puede incluir el uso de listas blancas para permitir solo ciertos tipos de entrada y eliminar cualquier carácter que pueda ser utilizado para ejecutar comandos SQL.

Uso de APIs seguras: Utilizar APIs o librerías específicamente diseñadas para interactuar con la base de datos de forma segura, en lugar de construir consultas SQL manualmente.

Principio de menor privilegio: Asegurarse de que las cuentas de usuario utilizadas por la aplicación tengan los permisos mínimos necesarios en la base de datos para realizar sus funciones. Esto limita el impacto de cualquier ataque exitoso de inyección de SQL.

Lab 14

¿Qué beneficios encuentras en el estilo MVC?

Separación de preocupaciones: MVC separa la lógica de presentación (Vista) de la lógica de negocio (Modelo) y la lógica de control (Controlador), lo que facilita la organización y el mantenimiento del código.

Reutilización de código: Al separar las distintas capas del sistema, se promueve la reutilización de componentes, lo que puede acelerar el desarrollo y reducir la duplicación de código.

Facilidad para realizar pruebas unitarias: Al tener componentes claramente definidos y separados, es más sencillo realizar pruebas unitarias en cada parte del sistema, lo que mejora la calidad del software.

¿Encuentras alguna desventaja en el estilo arquitectónico MVC?

A pesar de sus numerosos beneficios, el estilo arquitectónico MVC también presenta algunas desventajas potenciales:

Complejidad inicial: Al principio, puede resultar complicado entender la estructura y el flujo de datos en un sistema MVC, especialmente para desarrolladores novatos.

Acoplamiento: Aunque MVC promueve la separación de preocupaciones, un acoplamiento excesivo entre las distintas capas puede dificultar la modificación de una parte del sistema sin afectar a otras partes.

Posible sobrecarga de archivos y clases: En proyectos pequeños, la separación en tres capas distintas puede parecer excesiva y llevar a una proliferación innecesaria de archivos y clases, lo que puede aumentar la complejidad y el tiempo de desarrollo.

Lab 12

¿Qué otros templating engines existen para node?

Pug (antes conocido como Jade): Es un motor de plantillas con sintaxis simplificada y basada en indentación.

EJS (Embedded JavaScript): Permite mezclar JavaScript con HTML para generar vistas dinámicas.

Mustache: Es un sistema de plantillas lógicas sin lógica incorporada. Es muy simple y puede ser usado en varios lenguajes de programación, incluido Node.js.

Nunjucks: Es un motor de plantillas inspirado en Jinja2 (para Python) que permite la herencia, bloques y macros.

Marko: Un motor de plantillas creado por eBay que ofrece alto rendimiento y una sintaxis intuitiva.

Swig: Un motor de plantillas similar a Jinja2 que proporciona una sintaxis flexible y es compatible con Express.js.

EJS-mate: Una extensión de EJS que agrega características adicionales como layouts y bloques de contenido.

Lab11

Describe el archivo package.json.

El archivo package.json es un archivo de metadatos utilizado principalmente en proyectos de Node.js. Es un componente esencial en cualquier proyecto Node.js, ya que proporciona información sobre el proyecto, sus dependencias, scripts de ejecución y otra información relevante.

Aquí hay una descripción detallada de los campos comunes que se encuentran en un archivo package.json:

name: El nombre del paquete. Debe ser único en el registro de paquetes npm. También puede contener el ámbito del paquete si es un paquete privado o está asociado con una organización específica.

version: La versión del paquete. Sigue el formato X.Y.Z, donde X, Y y Z son números enteros que representan el número de versión principal, secundaria y de revisión respectivamente. Las actualizaciones de versión siguen las convenciones de versionamiento semántico.

description: Una breve descripción del paquete que describe su funcionalidad.

main: La ruta relativa al archivo principal del paquete que se utilizará al requerir el paquete.

scripts: Un objeto que define varios scripts de npm que se pueden ejecutar con npm run <script-name>. Estos scripts pueden incluir comandos para ejecutar tareas como construir, probar, iniciar la aplicación, etc.

keywords: Una matriz de palabras clave que ayudan a clasificar el paquete y facilitan su descubrimiento a través de la búsqueda.

author: El nombre del autor del paquete.

license: La licencia bajo la cual se distribuye el paquete. Puede ser cualquier valor válido que represente la licencia, como una cadena de texto o un objeto que detalla la información de la licencia.

dependencies: Un objeto que enumera las dependencias del proyecto. Estas son las bibliotecas y paquetes que son necesarios para que el proyecto funcione correctamente en tiempo de ejecución.

devDependencies: Similar a dependencies, pero para dependencias que son necesarias solo durante el desarrollo, como herramientas de prueba, paquetes de construcción, etc.

optionalDependencies: Dependencias que son opcionales y se instalarán si están disponibles, pero no son requeridas para el funcionamiento básico del paquete.

Lab 6

¿Por qué es una buena práctica usar JavaScript para checar que sean válidos los inputs de las formas antes de enviar los datos al servidor?

Usar JavaScript para validar los inputs de los formularios antes de enviar los datos al servidor es una buena práctica por varias razones:}

Experiencia del usuario mejorada: La validación en el lado del cliente permite que los usuarios reciban retroalimentación inmediata sobre los errores en sus entradas. Esto ayuda a mejorar la experiencia del usuario al evitar que tengan que esperar a que se procese el formulario en el servidor para saber si hay errores.

Reducción de carga en el servidor: Al validar los datos en el navegador antes de enviarlos al servidor, se reducen las solicitudes innecesarias de formularios con datos incorrectos o incompletos.

¿Cómo puedes saltarte la seguridad de validaciones hechas con JavaScript?

Inyección de código JavaScript: Esto implica modificar el código fuente de la página web en el navegador para desactivar o modificar las validaciones. Se podría usar la consola del navegador para manipular el DOM o alterar las funciones de validación directamente.

Interceptar y modificar las solicitudes HTTP: Las validaciones también pueden llevarse a cabo en el servidor. En este caso, modificar los datos enviados al servidor antes de que se realice la validación podría permitir eludir las restricciones.

Manipulación de cookies: Algunas aplicaciones web utilizan cookies para almacenar información sobre el estado de la sesión o los datos de autenticación. Modificar o eliminar cookies podría permitir eludir ciertas validaciones.

Si te puedes saltar la seguridad de las validaciones de JavaScript, entonces ¿por qué la primera pregunta dice que es una buena práctica?

Saltarse las validaciones de JavaScript no significa que se hayan superado las medidas de seguridad en su totalidad. Un atacante podría intentar sortear estas validaciones directamente mediante técnicas como la modificación del código JavaScript en el navegador. Por eso es esencial implementar también validaciones y medidas de seguridad robustas en el lado del servidor para garantizar la integridad y seguridad de la aplicación web.

En resumen, la primera pregunta sugiere que implementar validaciones de seguridad en JavaScript es una buena práctica, pero también reconoce que estas no son suficientes por sí solas y deben complementarse con medidas de seguridad en el lado del servidor para una protección adecuada contra ataques.