INTRODUCCIÓN AL DESARROLLO FRONTEND

CON REACT

2023

ANDÉN Centro de Innovación y Emprendimientos Tecnológicos

SECRETARÍA DE EXTENSIÓN UNIVERSITARIA UTN - FRC SEU

UTN Facultad Regional Córdoba

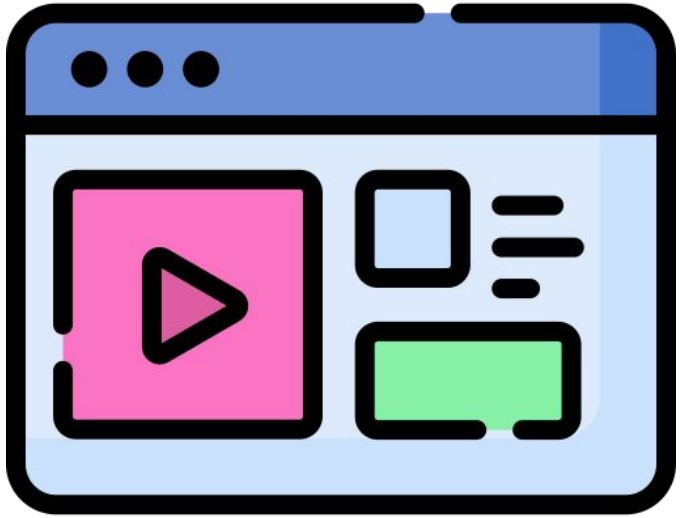
Agencia CÓRDOBA JOVEN

CÓRDOBA entre todos

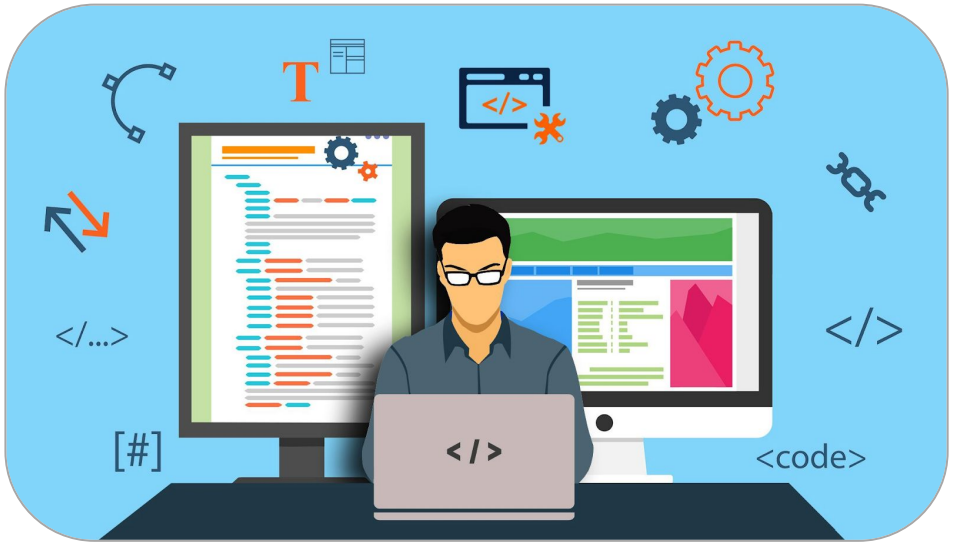
Clase 08 - Contenido

Desarrollo seguro en aplicaciones web

- Introducción a la seguridad en aplicaciones web
- Principales amenazas y vulnerabilidades
- Buenas prácticas de seguridad en el desarrollo web



Objetivos de la Clase

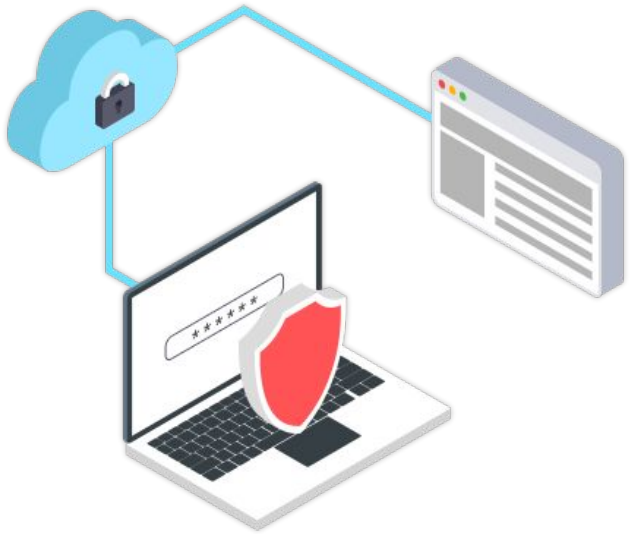


1. **Conciencia sobre Seguridad:** Entender la crucial importancia de incorporar prácticas de seguridad desde el inicio del desarrollo de una aplicación web.
2. **Introducción a React:** Familiarizarse con las características y ventajas de React como framework frontend y su relación con la seguridad web.
3. **Amenazas Comunes:** Identificar las principales amenazas y vulnerabilidades en el desarrollo web y reconocer cómo pueden afectar aplicaciones creadas con React.
4. **Implementación de Seguridad en React:** Aprender las mejores prácticas y herramientas específicas para implementar seguridad en aplicaciones React.
5. **Aplicación Práctica:** Desarrollar la habilidad de aplicar de manera efectiva las estrategias y técnicas aprendidas para mitigar riesgos y proteger aplicaciones React contra amenazas comunes.

¿Por qué es importante la seguridad en aplicaciones web?

- **Confianza del Usuario:**
  - Las aplicaciones web seguras cultivan la confianza de los usuarios, quienes esperan que sus datos estén protegidos y que su privacidad sea respetada.
- **Protección de Datos Sensibles:**
  - Muchas aplicaciones manejan información confidencial, desde datos personales hasta información financiera. Una brecha de seguridad podría tener consecuencias catastróficas.
- **Reputación y Credibilidad:**
  - Las violaciones de seguridad pueden dañar gravemente la reputación de una empresa o marca, lo que puede traducirse en pérdidas económicas y de clientes.
- **Cumplimiento Legal y Regulatorio:**
  - En muchos países, hay leyes y regulaciones que requieren que las empresas protejan los datos de los usuarios. No cumplir con estas normas puede resultar en sanciones y multas.
- **Prevención de Ataques:**
  - Las aplicaciones inseguras son blancos fáciles para los ciberdelincuentes. La seguridad adecuada puede prevenir una variedad de ataques, desde la inyección de código hasta el robo de identidad.

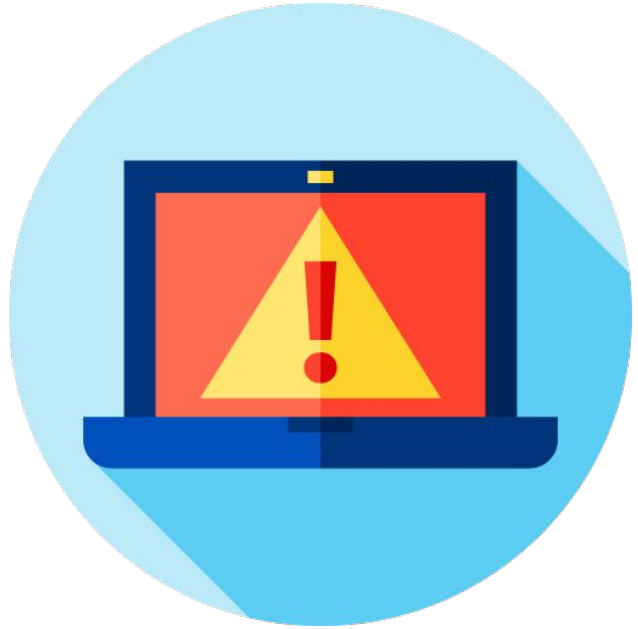
Desarrollo Seguro en Aplicaciones Web



- **Definición:**
  - El desarrollo seguro se refiere a las prácticas y procesos adoptados para crear aplicaciones web y software que están protegidos contra las amenazas y vulnerabilidades más comunes.
- **Importancia de la Prevención:**
  - Es más costoso y dañino remediar una brecha de seguridad que prevenirla. El desarrollo seguro busca identificar y mitigar riesgos desde el inicio.
- **Ciclo de Vida del Desarrollo Seguro (SDLC):**
  - Es un proceso que incorpora prácticas de seguridad en cada fase del desarrollo de software, desde la conceptualización hasta el mantenimiento.
- **Responsabilidad Compartida:**
  - La seguridad no es solo tarea del equipo de seguridad. Todos los involucrados en el desarrollo, desde diseñadores hasta desarrolladores y testers, juegan un papel crucial en la construcción de aplicaciones seguras.
- **Beneficios a Largo Plazo:**
  - Adoptar un enfoque de desarrollo seguro reduce riesgos, evita costos inesperados por brechas de seguridad y garantiza la confianza y lealtad del usuario.

Consecuencias de una Aplicación Insegura

- **Pérdida de Confianza del Cliente:**
  - Los clientes pueden dudar en usar o continuar usando un servicio si sienten que su información no está segura.
- **Daño a la Reputación:**
  - Las violaciones de seguridad pueden llevar a una mala prensa, dañando la imagen y la credibilidad de la empresa en el mercado.
- **Pérdidas Financieras:**
  - Desde multas regulatorias hasta pérdida de ingresos debido a la disminución de clientes, las implicaciones financieras pueden ser significativas.
- **Costos de Remediación:**
  - Resolver una vulnerabilidad después de que ha sido explotada es a menudo mucho más costoso que prevenirla.
- **Problemas Legales y Regulatorios:**
  - Las violaciones de datos pueden llevar a demandas, además de sanciones impuestas por organismos reguladores.
- **Impacto en Usuarios:**
  - Los datos personales pueden ser vendidos o utilizados de manera fraudulenta, lo que lleva a problemas como el robo de identidad.



Introducción a la Seguridad en Aplicaciones Web

- **Definición de Seguridad Web:**
  - Es la práctica de proteger las aplicaciones web y la información que manejan de amenazas y vulnerabilidades.
- **Elementos Clave:**
  - Las aplicaciones web interactúan con una variedad de elementos, desde bases de datos hasta navegadores y APIs. Cada interacción presenta posibles vectores de ataque.
- **Modelo de Amenaza:**
  - Antes de proteger una aplicación, es crucial entender a qué se enfrenta. Identificar posibles amenazas y cómo podrían explotarse es el primer paso.
- **Principio de Menor Privilegio:**
  - Cada proceso, usuario y programa debe operar con el menor conjunto de privilegios necesario para completar su función.
- **La Realidad del Paisaje de Amenazas:**
  - Las amenazas evolucionan constantemente. Lo que es seguro hoy puede no serlo mañana. La educación continua y la adaptación son esenciales.
- **Compromiso entre Usabilidad y Seguridad:**
  - A menudo, hay un equilibrio a considerar entre hacer que una aplicación sea fácil de usar y mantenerla segura. Es esencial encontrar un equilibrio adecuado.

Capas de Seguridad Web

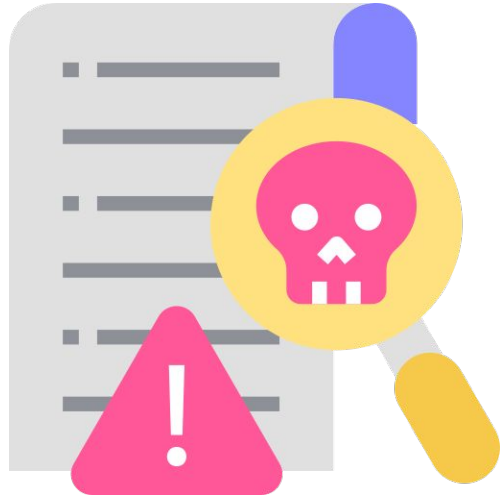
- **Capa de Red:**
  - Abarca medidas como firewalls y sistemas de detección/preventivos de intrusiones. Protege contra amenazas como ataques DDoS y escaneo de puertos.
- **Capa de Transporte:**
  - Se centra en la seguridad de las conexiones, implementando protocolos como TLS/SSL para cifrar y proteger el tráfico de datos entre cliente y servidor.
- **Capa de Aplicación:**
  - Aquí es donde entran en juego aspectos como la validación de entrada, la gestión de sesiones y las medidas contra vulnerabilidades como XSS o inyecciones SQL.
- **Capa de Datos:**
  - Protege la información almacenada, ya sea en bases de datos, archivos o sistemas de almacenamiento en la nube. Considera encriptación, controles de acceso y medidas contra inyecciones de código.
- **Capa de Autenticación y Autorización:**
  - Asegura que solo los usuarios permitidos accedan a recursos específicos y que lo hagan de manera correcta y segura. Incluye técnicas como 2FA, OAuth y gestión de tokens.
- **Capa de Monitorización y Respuesta:**
  - Se trata de detectar actividad anómala o maliciosa en tiempo real y responder adecuadamente, ya sea alertando al personal de seguridad o tomando medidas automáticas.

Principales Amenazas y Vulnerabilidades

- **Cross-Site Scripting (XSS):**
  - Ataques que inyectan scripts maliciosos en páginas web, afectando directamente a los usuarios del sitio.
- **Inyección SQL (SQLi):**
  - Exploita vulnerabilidades en la validación de entrada para ejecutar consultas SQL no deseadas en una base de datos.
- **Cross-Site Request Forgery (CSRF):**
  - Ataques que engañan al usuario para que ejecute acciones no deseadas en una aplicación web en la que está autenticado.
- **Ataques de Man-in-the-Middle (MitM):**
  - Interceptan y posiblemente alteran la comunicación entre dos partes sin que ninguna de ellas lo sepa.
- **Secuestro de Sesión:**
  - Ataques que roban o adivinan el ID de sesión de un usuario para usurpar su identidad en una aplicación web.
- **Ataques de Fuerza Bruta:**
  - Intentos repetidos de adivinar contraseñas u otros datos de autenticación.
- **Exposición de Datos Sensibles:**
  - Fallos de seguridad que permiten a los atacantes acceder a información confidencial, como datos personales o financieros.

Cross-Site Scripting (XSS)

- **Definición de XSS:**
  - Es un tipo de ataque que permite a los agresores inyectar scripts maliciosos en páginas web vistas por otros usuarios. Estos scripts pueden acceder a cualquier cookie, token de sesión o información sensible que el navegador use con ese sitio.
- **Tipos de XSS:**
  - Reflejado: El script se introduce a través de la URL o formularios y se refleja inmediatamente en la página.
  - Almacenado: El script malicioso se almacena en el servidor (por ejemplo, en una base de datos) y se muestra a otros usuarios.
  - DOM-based: El script se ejecuta a resultas de modificar el DOM en el navegador del usuario.
- **React y XSS:**
  - Por defecto, React escapa cualquier información que se inserta en el contenido de la página, lo que ayuda a prevenir ataques XSS. Sin embargo, es crucial evitar el uso de funciones que pueden introducir vulnerabilidades, como dangerouslySetInnerHTML.
- **Buenas Prácticas:**
  - Evita la inserción de HTML dinámico siempre que sea posible.
  - Utiliza bibliotecas y funciones que escapen automáticamente contenido, como las proporcionadas por React.
  - Mantente informado sobre las últimas vulnerabilidades y parches en React y otras bibliotecas que utilices.

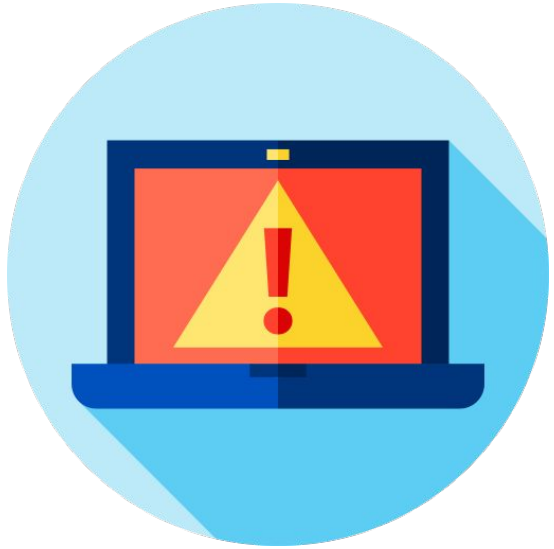


Cross-Site Request Forgery (CSRF)

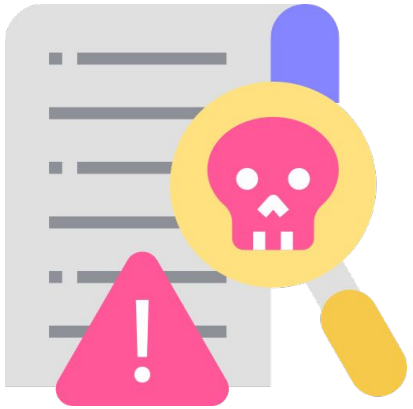
- **Definición de CSRF:**
  - Es un ataque que engaña a un usuario autenticado para que ejecute una acción no deseada en una aplicación web en la que está autenticado, sin su conocimiento ni consentimiento.
- **¿Cómo Funciona?:**
  - Un agresor podría engañar al usuario para que haga clic en un enlace o cargue una página que realiza una solicitud no deseada a una aplicación web en la que el usuario está autenticado, aprovechando las cookies de autenticación.
- **Impacto del Ataque:**
  - Puede llevar a acciones no autorizadas, como cambiar la contraseña de un usuario, modificar datos o realizar acciones en nombre del usuario sin su conocimiento.
- **Medidas de Prevención:**
  - **Token CSRF:** Utilizar un token que se verifica con cada solicitud, asegurando que las peticiones son intencionales.
  - **Comprobar el encabezado 'Referer':** Asegurarse de que las solicitudes provienen del sitio esperado.
  - **Requerir Autenticación:** Para acciones críticas, solicitar al usuario que se autentique de nuevo o confirme la acción.
  - **SameSite Cookie Attribute:** Configurar las cookies para que solo se envíen en solicitudes del mismo sitio, previniendo ataques entre sitios.

Inyección SQL

- **Definición de Inyección SQL:**
  - Es un tipo de ataque en el que el agresor puede ejecutar comandos SQL arbitrarios en una base de datos a través de una aplicación. Esto puede llevar a la manipulación, filtración o destrucción de datos.
- **Relación con el Frontend:**
  - Aunque es un ataque dirigido a bases de datos, se inicia a través del frontend. Los datos maliciosos se introducen a través de campos de entrada, como formularios, que no validan o desinfectan adecuadamente las entradas del usuario.
- **Impacto Potencial:**
  - Acceso no autorizado a datos sensibles, modificación o eliminación de registros, y en algunos casos, toma de control sobre el servidor de la base de datos.
- **Medidas de Prevención en el Frontend:**
  - **Validación de Entrada:** Todas las entradas del usuario deben ser validadas y desinfectadas antes de ser procesadas.
  - **Uso de Consultas Parametrizadas:** En lugar de construir consultas SQL mediante la concatenación de strings, se deben usar consultas parametrizadas para garantizar que las entradas se traten como datos y no como código.
  - **Mensajes de Error Genéricos:** No revelar detalles específicos de errores SQL en el frontend para evitar dar pistas a los atacantes.
  - **Limitar Privilegios:** Asegurarse de que las cuentas de base de datos utilizadas por las aplicaciones web tengan los mínimos privilegios necesarios.



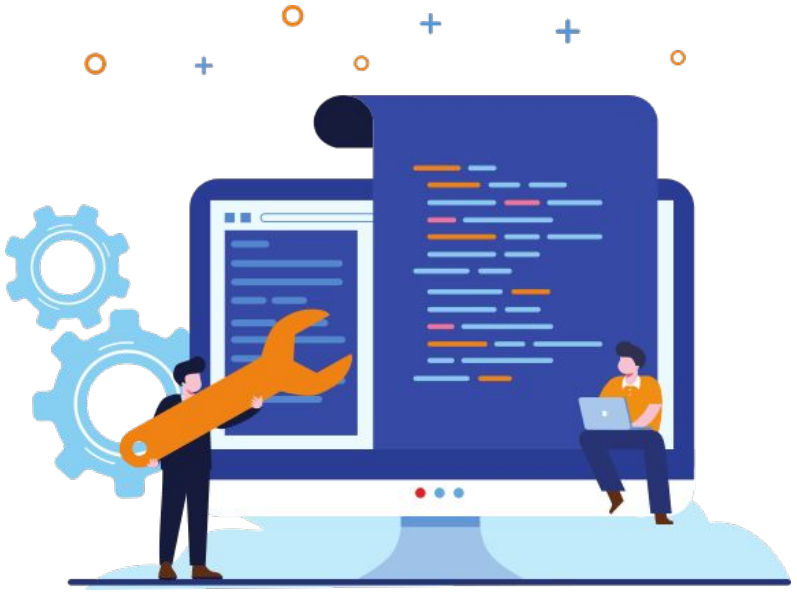
Ataques de Man-in-the-Middle



- **Definición de Man-in-the-Middle (MitM):**
  - Es un tipo de ataque donde el agresor intercepta y, a menudo, altera la comunicación entre dos partes sin que ninguna de ellas lo sepa.
- **¿Cómo ocurre?:**
  - Puede suceder en múltiples niveles, desde redes Wi-Fi no seguras hasta ataques en el nivel de transporte, como el secuestro de sesión TLS.
- **Impacto del Ataque:**
  - Robo de datos sensibles, interceptación de credenciales de inicio de sesión, alteración de mensajes transmitidos y posible inyección de malware.
- **Relevancia para el Frontend:**
  - Aunque es un ataque de red, afecta directamente al frontend, ya que los datos enviados o recibidos por el cliente pueden ser capturados o alterados.
- **Medidas de Prevención:**
  - **HTTPS:** Siempre usar HTTPS para asegurar la comunicación entre el cliente y el servidor.
  - **HSTS (Strict Transport Security):** Una cabecera que fuerza a los navegadores a usar conexiones HTTPS.
  - **Verificar Certificados:** Asegurarse de que el cliente verifique la validez de los certificados para prevenir ataques de intermediarios.
  - **VPNs y Redes Seguras:** Evitar el uso de redes públicas no seguras al interactuar con aplicaciones sensibles.

Buenas Prácticas de Seguridad en el Desarrollo Web

- **Educación Continua:**
  - Mantente informado sobre las últimas vulnerabilidades, ataques y soluciones en seguridad web. La formación constante es esencial.
- **Desarrollo Basado en Principios:**
  - Asegúrate de seguir principios básicos como el principio de menor privilegio, validación de entrada y salida, y defensa en profundidad.
- **Revisión de Código:**
  - Realizar revisiones de código regulares con un enfoque en la seguridad para identificar y rectificar posibles vulnerabilidades.
- **Uso de Herramientas de Análisis:**
  - Utilizar herramientas de análisis estático y dinámico para identificar vulnerabilidades en el código y en tiempo de ejecución, respectivamente.
- **Testeo de Seguridad:**
  - Además de las pruebas unitarias y de integración, realizar pruebas de penetración y de seguridad específicas para evaluar la robustez de la aplicación.
- **Manejo de Dependencias:**
  - Mantener todas las dependencias y librerías actualizadas, y estar atentos a las vulnerabilidades reportadas en estas.
- **Plan de Respuesta a Incidentes:**
  - Tener un plan establecido para responder rápidamente a cualquier brecha o vulnerabilidad detectada, minimizando así el impacto.



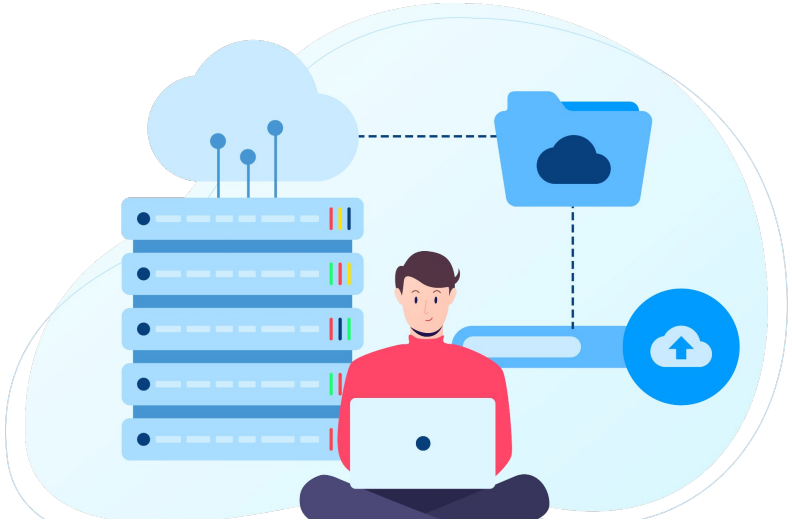
Validación de Datos

- **Importancia de la Validación:**
  - La entrada no validada es una de las principales causas de vulnerabilidades en las aplicaciones web. Cada entrada debe ser tratada como potencialmente peligrosa.
- **Tipos de Validación:**
  - Validación del Lado del Cliente: Realizada en el navegador del usuario, puede mejorar la experiencia del usuario pero no es suficiente por sí sola.
  - Validación del Lado del Servidor: Esencial para la seguridad, ya que verifica los datos antes de que sean procesados o almacenados.
- **Listas Blancas vs Listas Negras:**
  - Es más seguro especificar qué está permitido (lista blanca) que intentar enumerar todas las posibles entradas maliciosas (lista negra).
- **Desinfección de Datos:**
  - Además de validar, es importante desinfectar la entrada, eliminando o neutralizando cualquier dato que pueda ser perjudicial.
- **Errores de Validación:**
  - Al informar al usuario sobre errores de validación, evita dar detalles que puedan ayudar a un atacante. Proporciona mensajes de error genéricos y útiles.
- **Consideraciones Especiales:**
  - Las aplicaciones con características como cargas de archivos, búsquedas o cualquier funcionalidad que acepte entrada del usuario deben tener consideraciones especiales para validar y desinfectar correctamente.

Manejo Seguro de Contraseñas

- **Importancia del Manejo Seguro:**
  - Las contraseñas son a menudo el único obstáculo entre un atacante y datos sensibles. Un manejo inadecuado puede llevar a brechas de seguridad catastróficas.
- **Nunca Almacenes Contraseñas en Texto Plano:**
  - Almacenar contraseñas en su forma original es una práctica riesgosa y altamente desaconsejada.
- **Uso de Hash:**
  - Convertir contraseñas en un valor hash antes de almacenarlas. Asegurarse de usar algoritmos de hash seguros y actualizados como bcrypt o Argon2.
- **Salting de Contraseñas:**
  - Añadir un valor único (salt) a la contraseña antes de hacer el hash. Esto asegura que incluso si dos usuarios tienen la misma contraseña, sus hashes serán diferentes.
- **Ataques de Fuerza Bruta y Diccionario:**
  - Utilizar medidas como límites de intentos, CAPTCHAs y tiempos de espera para proteger contra estos ataques.
- **Verificación de Contraseñas en el Servidor:**
  - Siempre verifica contraseñas en el lado del servidor, no confíes en las validaciones del cliente.
- **Actualizaciones y Mantenimiento:**
  - A medida que los algoritmos de hash se vuelven obsoletos o se descubren vulnerabilidades, es esencial actualizar y migrar a métodos más seguros.

Actualizaciones y Dependencias

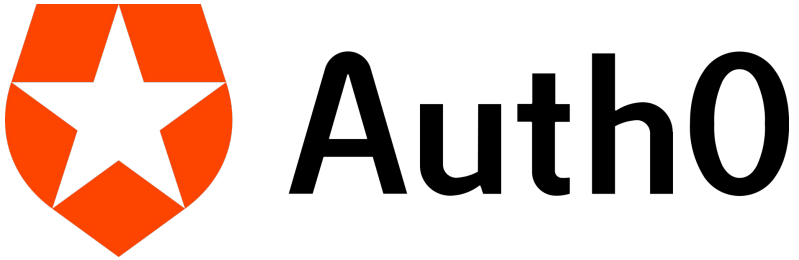


- **Importancia de las Actualizaciones:**
  - Las vulnerabilidades son descubiertas constantemente en software y librerías. Mantener todo actualizado es esencial para cerrar posibles brechas de seguridad.
- **Gestión de Dependencias:**
  - Utiliza herramientas como npm o yarn para rastrear y gestionar tus dependencias. Estas herramientas pueden informarte de vulnerabilidades conocidas.
- **Auditorías de Seguridad:**
  - Herramientas como npm audit pueden identificar y sugerir correcciones para vulnerabilidades en tus dependencias.
- **Versiones Obsoletas:**
  - Evita usar versiones obsoletas o ya no mantenidas de librerías y frameworks. Estas pueden tener vulnerabilidades no corregidas.
- **Revisión Regular:**
  - Realiza revisiones regulares de tus dependencias y actualiza cuando sea necesario, no solo cuando hay una vulnerabilidad conocida.
- **Código Propio:**
  - No solo las dependencias externas pueden ser un riesgo. Mantén también tu propio código actualizado y corrige cualquier vulnerabilidad que descubras o de la que te informen.
- **Pruebas Después de Actualizar:**
  - Asegúrate de realizar pruebas exhaustivas después de cualquier actualización para garantizar que todo funciona como se espera y que no se introducen nuevos problemas.

Autenticación y Autorización

- **Diferencia Clave:**
  - **Autenticación:** Verificar la identidad de un usuario (¿Quién eres?).
  - **Autorización:** Determinar los derechos o permisos de un usuario autenticado (¿Qué puedes hacer?).
- **Importancia:**
  - Es esencial garantizar que solo los usuarios adecuados tengan acceso a recursos específicos y que puedan realizar solo acciones permitidas.
- **Métodos de Autenticación:**
  - Contraseña tradicional, autenticación de dos factores (2FA), autenticación biométrica, entre otros.
- **Gestión de Sesiones:**
  - Una vez que un usuario está autenticado, es esencial gestionar su sesión de manera segura, utilizando tokens, cookies seguras y técnicas de renovación de sesión.
- **Control de Acceso Basado en Roles (RBAC):**
  - Define roles (como “usuario”, “administrador”, etc.) y asigna permisos específicos a esos roles. Luego, asigna roles a los usuarios para determinar lo que pueden y no pueden hacer.
- **Principio de Menor Privilegio:**
  - Otorga a los usuarios y aplicaciones solo los permisos que necesitan para realizar una tarea específica y nada más.
- **Auditoría y Registro:**
  - Mantén registros detallados de todas las actividades de autenticación y autorización para rastrear cualquier comportamiento anómalo o malicioso.

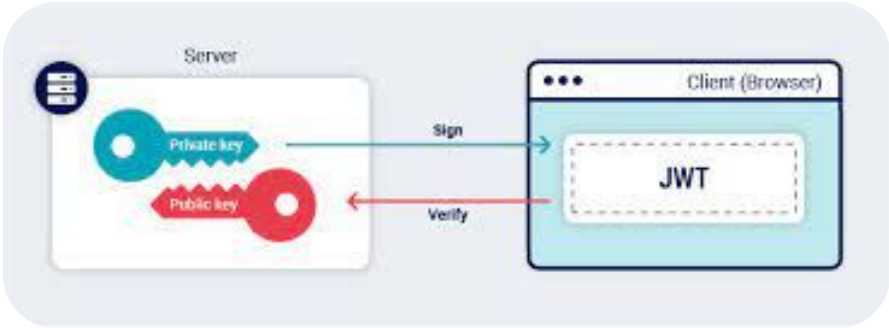
Configurar Auth0 para la gestión de usuarios de aplicación



- **¿Qué es Auth0?:**
  - Una plataforma universal de identidad que ofrece soluciones para la autenticación, autorización y gestión de usuarios.
- **Ventajas de usar Auth0:**
  - Configuración rápida y sencilla.
  - Integración con múltiples proveedores de identidad (como Google, Facebook).
  - Soporte para autenticación de dos factores (2FA).
  - Herramientas de monitorización y análisis de seguridad.
- **Pasos básicos para la configuración:**
  - Crear una cuenta en Auth0.
  - Definir una nueva “Aplicación” en el dashboard de Auth0.
  - Configurar las URL de callback, logout y origen permitidas.
  - Integrar Auth0 en tu aplicación utilizando las SDKs proporcionadas.
  - Personalizar flujos y pantallas de autenticación según las necesidades.
- **Gestión de Usuarios:**
  - Utiliza el dashboard de Auth0 para gestionar usuarios, asignar roles y definir permisos.
  - Posibilidad de conectar Auth0 con bases de datos externas o sistemas de gestión de identidad.
- **Seguridad y Cumplimiento:**
  - Auth0 cumple con estándares y regulaciones de la industria, ofreciendo características como la rotación de claves, el hashing de contraseñas y auditorías de seguridad.

Tokens y su Almacenamiento Seguro

- **¿Qué es un Token?:**
  - Es una cadena de caracteres que representa una afirmación o reclamo sobre un sujeto (como un usuario). Se utiliza para mantener a un usuario autenticado o para transmitir información entre partes de manera segura.
- **Tipos de Tokens más Comunes:**
  - **JWT (JSON Web Token):** Una representación compacta y autocontenida de una afirmación que se puede transferir entre dos partes.
  - **OAuth Tokens:** Utilizados en el protocolo OAuth para autorización.
- **Ventajas del Uso de Tokens:**
  - No requieren consulta a la base de datos para validar cada solicitud.
  - Flexibilidad para ser usados entre diferentes aplicaciones o servicios.
  - Puede agregar información adicional (payload) que puede ser decodificada del lado del cliente.
- **¿Dónde guardar un Token?:**
  - **Cookies:** Una opción común, pero es vulnerable a ataques CSRF.
  - **LocalStorage:** Es fácil de usar, pero vulnerable a ataques XSS.
  - **SessionStorage:** Similar a LocalStorage, pero limitado a una sesión.
  - **En memoria:** Almacenar el token en una variable de estado en aplicaciones SPA. Menos persistente, pero más seguro contra ataques.
- Recomendación:
  - La elección depende de la naturaleza de la aplicación y las amenazas consideradas. En aplicaciones modernas de React, almacenar tokens en memoria combinado con otras medidas de seguridad es a menudo una buena práctica.

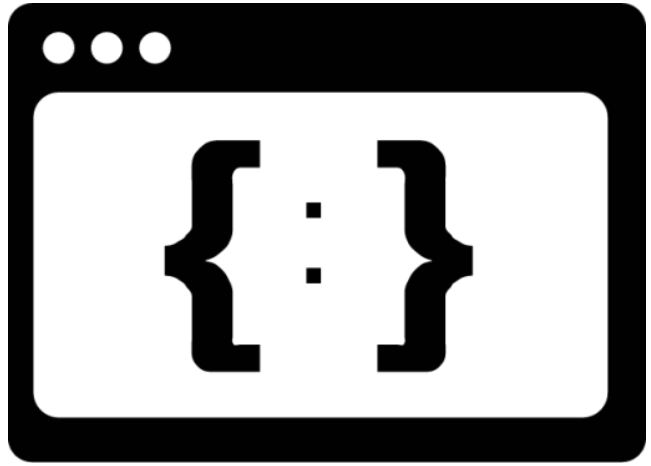


HTTPS y Seguridad de la Comunicación

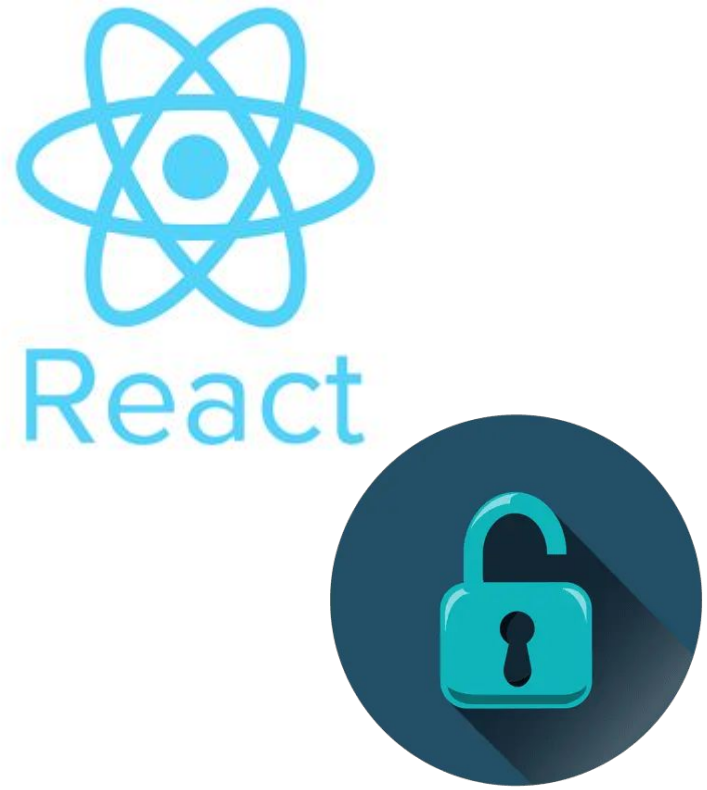
- **¿Qué es HTTPS?:**
  - Protocolo de Transferencia de Hipertexto Seguro. Es una versión segura de HTTP, que utiliza SSL/TLS para cifrar la comunicación entre el cliente y el servidor.
- **Importancia del HTTPS:**
  - **Confidencialidad:** Asegura que la información intercambiada no puede ser leída por terceros.
  - **Integridad:** Garantiza que los datos no sean alterados durante la transmisión.
  - **Autenticación:** Confirma que te estás comunicando con el sitio web legítimo y no con un impostor.
- **Riesgos de No Usar HTTPS:**
  - Exposición de datos sensibles.
  - Ataques Man-in-the-Middle.
  - Pérdida de confianza del usuario.
- **Consideraciones al Implementar HTTPS:**
  - Obtener un certificado de una Autoridad Certificadora (CA) de confianza.
  - Configurar el servidor para usar cifrado fuerte y protocolos seguros.
  - Usar HSTS (Strict Transport Security) para asegurar que los navegadores solo se conecten mediante HTTPS.
- **¿Solo para Sitios con Información Sensible?:**
  - Antes se creía que HTTPS era necesario solo para sitios con transacciones financieras o datos sensibles. Hoy, la tendencia es que todos los sitios deben usar HTTPS para proteger la integridad de la web.

Headers de Seguridad

- **Importancia de los Headers de Seguridad:**
  - Los headers HTTP pueden ser configurados para mejorar la seguridad de tu aplicación, previniendo ciertos tipos de ataques o restricciones en ciertos comportamientos del navegador.
- **Content Security Policy (CSP):**
  - Define qué recursos pueden ser cargados por una página, previniendo ataques XSS y la inyección de contenido malicioso.
- **Strict-Transport-Security (HSTS):**
  - Asegura que los navegadores solo se conecten al sitio usando HTTPS, previniendo ataques de interceptación y downgrade de protocolo.
- **X-Frame-Options:**
  - Previene que tu sitio sea incorporado en un iframe, protegiendo contra ataques de clickjacking.
- **X-Content-Type-Options:**
  - Detiene ataques basados en el “mime sniffing”, forzando al navegador a usar el tipo MIME declarado por el servidor.
- **X-XSS-Protection:**
  - Activa la protección XSS incorporada en algunos navegadores.
- **Referrer-Policy:**
  - Controla cuánta información del referente (página origen) es incluida en las solicitudes.
- **Feature-Policy:**
  - Permite a los sitios controlar qué características y APIs pueden ser usadas en el navegador.
- **Conclusión:**
  - Utilizar headers de seguridad adecuados es esencial para fortalecer la seguridad de tu aplicación web, previniendo múltiples vectores de ataque conocidos.



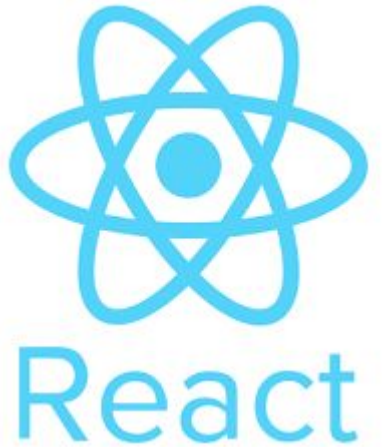
Seguridad en React



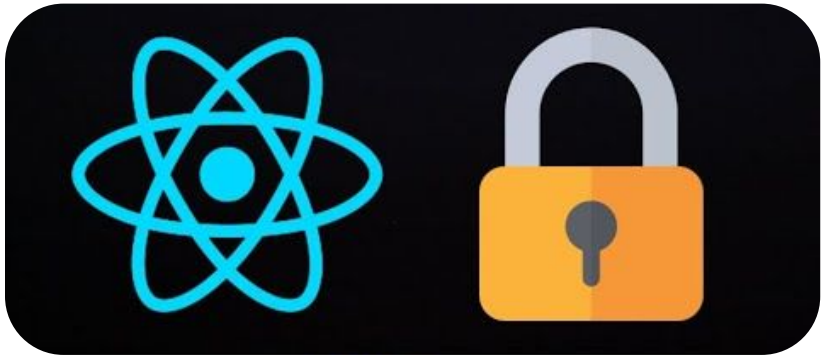
- **Protección contra XSS:**
  - React, por defecto, escapa todo el contenido antes de renderizarlo, lo que ayuda a prevenir ataques XSS.
  - Sin embargo, hay que tener cuidado al usar dangerouslySetInnerHTML, y siempre desinfectar el contenido.
- **Uso Seguro de setState:**
  - No utilices entradas de usuario directamente con setState sin validaciones, ya que puede abrir puertas a comportamientos inesperados.
- **Manejo Seguro de Eventos:**
  - Asegúrate de no pasar datos de usuario como argumentos en handlers de eventos sin las validaciones adecuadas.
- **Componentes de Terceros:**
  - Al usar componentes de terceros, asegúrate de que provienen de fuentes confiables y están bien mantenidos. Revisa si tienen vulnerabilidades conocidas.
- **Actualizaciones y Dependencias:**
  - Al igual que con cualquier otra herramienta, mantener React y sus librerías asociadas actualizadas es crucial para la seguridad.
- **Control de Acceso en el Frontend:**
  - Aunque las verificaciones de autorización deberían hacerse en el backend, es bueno tener controles de acceso en el frontend para mejorar la experiencia del usuario y añadir una capa extra de seguridad.
- **Headers y Solicitudes:**
  - Al hacer solicitudes a APIs, asegúrate de configurar tus headers correctamente y no exponer información sensible

Uso Seguro de setState

- **¿Qué es setState?:**
  - Es una función de React utilizada para actualizar el estado local de un componente. El nuevo estado se fusiona con el actual.
- **Inyección de Estado:**
  - Es similar a la inyección SQL en bases de datos. Si no se valida y desinfecta correctamente la entrada del usuario, este podría modificar el estado de un componente de formas no previstas.
- **Valida Antes de setState:**
  - Siempre valida cualquier dato antes de pasarlo a setState, especialmente si proviene de entradas de usuario o fuentes no confiables.
- **No Dependes Directamente de la Entrada del Usuario:**
  - Evita usar la entrada del usuario directamente como argumento para setState. Procesa y valida los datos primero.
- **Sé Cauteloso con Datos Dinámicos:**
  - Si utilizas datos dinámicos (por ejemplo, de una API) para establecer el estado, asegúrate de que estos datos sean confiables y no contengan scripts maliciosos.
- **Uso de Funciones en setState:**
  - Cuando el nuevo estado depende del anterior, utiliza la forma de función de setState para garantizar la actualización correcta del estado.
- **Conclusión:**
  - Aunque React proporciona muchas protecciones por defecto, el uso seguro de setState requiere atención y comprensión. Asegurarte de que estás utilizando esta función correctamente es esencial para la integridad y seguridad de tu aplicación.



Herramientas de Seguridad para React



Plugins y librerías para fortalecer tu aplicación

- **DOMPurify:**
  - Continúa siendo una de las principales librerías para desinfectar y prevenir ataques XSS en aplicaciones web, incluidas las hechas con React.
- **ESLint y sus plugins:**
  - eslint-plugin-react: Proporciona reglas específicas de linting para React.
  - eslint-plugin-security: Identifica patrones potencialmente inseguros en el código.
- **React-query & SWR:**
  - Son librerías modernas para la gestión de datos en React que proporcionan patrones seguros para la manipulación y obtención de datos.
- **Helmet:**
  - Aunque originalmente estaba diseñado para Express.js, su contraparte react-helmet sigue siendo relevante para establecer metatags de seguridad y otros encabezados HTML en aplicaciones React.
- **Redux Toolkit:**
  - Es la recomendación oficial para la lógica de negocio en aplicaciones React-Redux. Proporciona patrones seguros y optimizados para el manejo del estado.
- **npm audit o yarn audit:**
  - Herramientas esenciales que deberían ser utilizadas regularmente para identificar y corregir vulnerabilidades en las dependencias del proyecto.
- **Content Security Policy (CSP):**
  - Aunque no es una herramienta en sí, establecer una CSP es crucial para prevenir varios tipos de ataques, como XSS. Herramientas como react-csp pueden ayudar a integrar CSP en aplicaciones React.

Bueno, Vamo a Codea!!!!

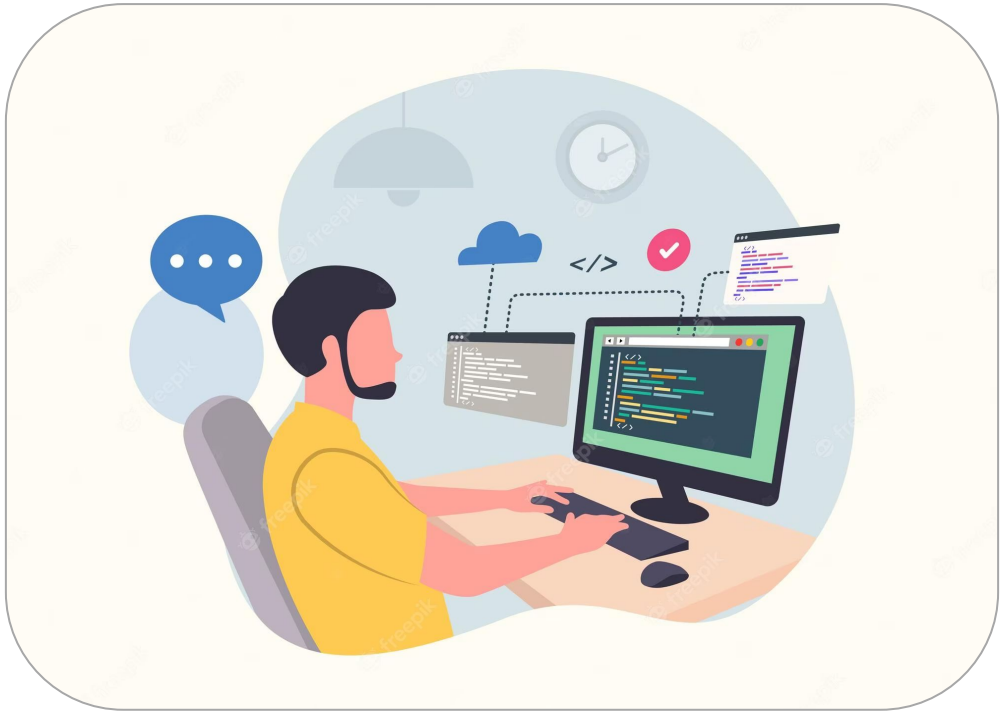
El objetivo de esta actividad es implementar un sistema de autenticación y autorización en una aplicación React utilizando Auth0. Los usuarios deberán poder iniciar sesión y acceder a sus perfiles.

Auth0

React

Actividad 8: Portal Perfil Usuario

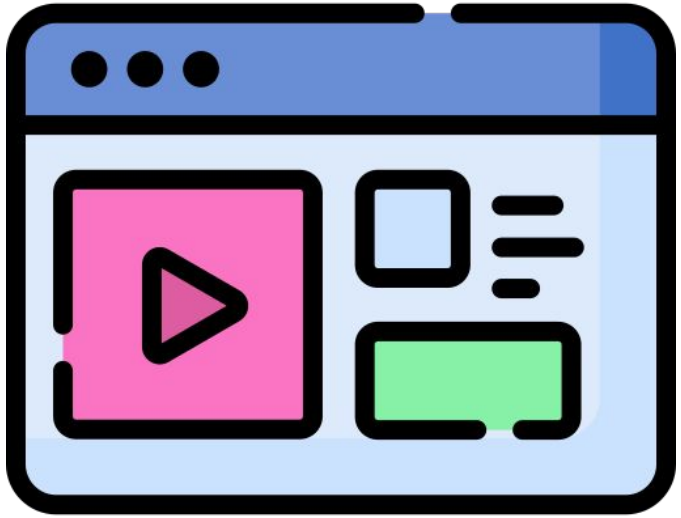
- Seguir las instrucciones de la actividad publicada en la UVE.



Próxima Clase

Integración con servicios de terceros

- Integración con servicios de terceros
- Uso de APIs
- Autenticación en servicios de terceros
- Ejemplos de integración con servicios populares
- Manejo de errores y excepciones



MUCHAS GRACIAS

ANDÉN Centro de Innovación y Emprendimientos Tecnológicos

SECRETARÍA DE EXTENSIÓN UNIVERSITARIA UTN - FRC SEU

UTN Facultad Regional Córdoba

Agencia CÓRDOBA JOVEN

CÓRDOBA entre todos