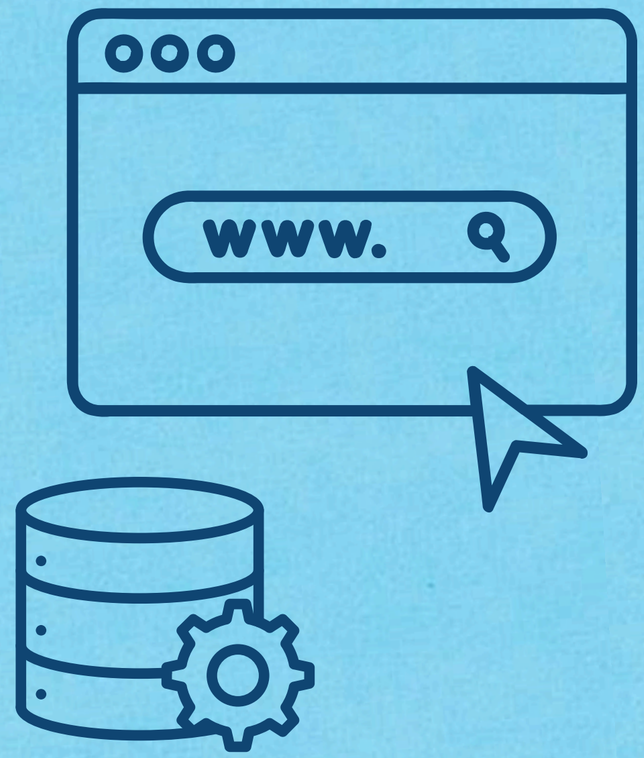
IIC2413

AYUDANTÍA 8

OBJETIVOS

- Entender cómo se conecta una web con PHP y sql
- Cómo hacer queries y transacciones para interactuar con una BDD a través de una interfaz web.
- Cargar Vistas, Stored Procedures, Triggers e Índices desde archivos .sql
- Aplicar sanitización de inputs para proteger la BDD



PARTE 1

CLIENTE - SERVIDOR

¿QUÉ ES UN SERVIDOR WEB?

Es un programa que recibe **peticiones HTTP** desde un navegador y devuelve una respuesta (puede ser HTML, una imagen, un JSON, el resultado de un programa).



MÉTODOS HTTP

GET

- Los datos van en la URL (son visibles)
- Es para **leer** información, no modificarla

Un ejemplo de URL sería:
https://ejemplo.com/ejemplo.php?persona=Pepito&edad=27

En PHP, se pueden acceder:
$_GET['persona'] → Pepito
$_GET[edad] → 27

POST

- Los datos van ocultos en el cuerpo de la petición (no se ven en la URL)
- Se usa para **enviar o modificar** datos (por ejemplo, al querer iniciar sesión o querer registrar algo nuevo)

En PHP, se pueden acceder:
$_POST['usuario']
$_POST['contraseña']

PARTE 2

HTML Y CSS

¿QUÉ SON?

- **HTML** define la **estructura** de la página: títulos, párrafos, imágenes, formularios, etc.
- Las etiquetas se escriben como: <tag>contenido</tag>
- Por otro lado, **CSS** le da **estilo** (colores, tamaños, etc).

En el código de ejemplo se muestra una estructura básica de un archivo HTML.

Página útil:
https://www.w3schools.com/tags/ref_byfunc.asp

- Ahí pueden ver otros tipos de tags que se utilizan en HTML

```html
<!--Archivo index.html-->
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Página random - Iniciar sesión</title>
    <link rel="stylesheet" href="stylesheets.css">
</head>
<body>
    <div class="container">
        <h1>Bienvenido a mi página</h1>

        <form action="validar_login.php" method="POST" class="formulario">
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>

            <label for="clave">Contraseña:</label>
            <input type="password" id="clave" name="clave" required>

            <button type="submit">Iniciar sesión</button>
        </form>

        <p>¿No tienes cuenta? <a href="registro.php">Regístrate aquí</a></p>
    </div>
</body>
</html>
```

PARTE 3

CONEXIÓN A LA BDD CON PHP

¿CÓMO SE CONECTA?

- **PDO** (PHP Data Objects) es una librería que permite a PHP hablar con la base de datos. Le mandas un query, lo ejecuta en PostgreSQL, y te devuelve los resultados como un array de PHP.
- Se crea un objeto PDO pasándole host, dbname, usuario y clave.
- Se usa try/catch para manejar errores de conexión.
- Esta función se reutiliza en todos los archivos PHP del proyecto.

```php
<?php
0 references
function conectar_db() {
    $host = 'localhost'; // Cambiar si se quiere usar el servidor remoto
    $dbname = nombre_bdd; // Nombre de base de datos
    $user = usuario; // Nombre de usuario
    $password = contraseña; // Número de alumno

    try {
        $db = new PDO("pgsql:host=$host;dbname=$dbname", $user, $password);
        $db -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        return $db;
    } catch (PDOException $e) {
        echo "Error de conexión: " . $e->getMessage();
        exit();
    }
}
?>
```

QUERIES SEGURAS

- Es importante que las queries y transacciones que hagan sean seguras (es decir, que eviten “SQL injection”).
- Para eso utilizamos **prepare** (como dice el nombre, “prepara” la query) y **bindParam** (une los parámetros pedidos con los del input)

```php
<?php
include 'conectar_bdd.php';
$db = conectar_db();

$email = $_POST['email'];
$clave = $_POST['clave'];

// 1. Preparar el query con placeholders
$query = "SELECT id_persona, nombre_completo FROM usuario
          WHERE email = :email AND clave = :clave";
$stmt = $db->prepare($query);

// 2. Vincular los valores reales (PDO los sanitiza)
$stmt->bindParam(':email', $email);
$stmt->bindParam(':clave', $clave);

// 3. Ejecutar
$stmt->execute();

// 4. Obtener resultados
$usuario = $stmt->fetch(PDO::FETCH_ASSOC);

if ($usuario) {
    echo "Bienvenido " . $usuario['nombre_completo'];
} else {
    echo "Credenciales inválidas";
}
?>
```

QUERIES SEGURAS

Para las transacciones hacemos lo mismo en su respectivo flujo:

- Se usa **beginTransaction()** y **endTransaction()**
- Usamos **prepare** y **bindParam**
- Se usan bloques **try** y **catch** para el manejo de errores
- De acuerdo a estos bloques, la transacción se confirma (**commit()**) o se deshace (**rollBack()**).

```php
<?php
// Ejemplo: Insertar un usuario
include 'conectar_bdd.php';
$db = conectar_db();

try {
    // 1. Inicia transacción
    $db->beginTransaction();

    // 2. Insertar al la tabla Persona
    $query = "INSERT INTO Persona (run, nombre, email)
    VALUES (:run, :nombre, :email)";
    $stmt = $db->prepare($query);
    $stmt->bindParam(':run', $run);
    $stmt->bindParam(':nombre', $nombre);
    $stmt->bindParam(':email', $email);
    $stmt->execute();

    // 3. Si todo está bien, se confirma
    $db->commit();
    echo "Persona registrada correctamente";

} catch (PDOException $e) {
    // Si falla, deshacer
    $db->rollBack();
    echo "Error: " . $e->getMessage();
}
?>
```

PARTE 4

EJERCICIO

SISTEMA DE BIBLIOTECA SIMPLE

¿Qué hace?

- Usuarios inician sesión
- Ven libros disponibles
- Pueden pedir libros prestados
- Disponibilidad de libros se actualiza automáticamente

SISTEMA DE BIBLIOTECA SIMPLE

Demo







¿CÓMO SE HACE?

schema.sql

- DROP para eliminar tablas existentes y partir de cero cuando se corre shema de nuevo

- Usuarios: guarda credenciales de usuarios
- Libros: contiene catálogo y disponibilidad
- Prestamos: relaciona usuarios + libros

- Insertamos datos de prueba

```sql
DROP TABLE IF EXISTS prestamos;
DROP TABLE IF EXISTS libros;
DROP TABLE IF EXISTS usuarios;

CREATE TABLE usuarios (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(50) NOT NULL
);

CREATE TABLE libros (
    id SERIAL PRIMARY KEY,
    titulo VARCHAR(100) NOT NULL,
    autor VARCHAR(100) NOT NULL,
    disponible BOOLEAN DEFAULT TRUE
);

CREATE TABLE prestamos (
    id SERIAL PRIMARY KEY,
    id_usuario INT REFERENCES usuarios(id),
    id_libro INT REFERENCES libros(id),
    fecha DATE DEFAULT CURRENT_DATE
);

INSERT INTO usuarios (nombre, password) VALUES
    ('ana', '1234'),
    ('pedro', 'abcd');

INSERT INTO libros (titulo, autor) VALUES
    ('Don Quijote', 'Cervantes'),
    ('Cien años de soledad', 'García Márquez'),
    ('El Principito', 'Saint-Exupéry');
```

¿CÓMO SE HACE?

libros_disponibles.sql

- Creamos vista para libros disponibles
- Ventajas:
  - Simplifica consultas
  - Encapsula lógica
  - Reutilizable

```sql
DROP VIEW  IF EXISTS libros_disponibles;

CREATE VIEW libros_disponibles AS
    SELECT
        l.id,
        l.titulo,
        l.autor
    FROM libros l
    WHERE l.disponible = TRUE;
```

¿CÓMO SE HACE?

actualizar_disponibilidad.sql

- **Objetivo:** marcar un libro como no disponible cuando se crea un préstamo
- Stored Procedure (fn_actualizar_disponibilidad): contiene lógica que cambia un libro a no disponible cuando se registra un préstamo
- Trigger (trg_actualizar_disponibilidad): detecta automáticamente un INSERT en la tabla ‘prestamos’ y ejecuta la función que actualiza la disponibilidad del libro

```sql
DROP FUNCTION IF EXISTS fn_actualizar_disponibilidad;

CREATE OR REPLACE FUNCTION fn_actualizar_disponibilidad()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE libros
    SET disponible = FALSE
    WHERE id = NEW.id_libro;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trg_actualizar_disponibilidad ON prestamos;

CREATE TRIGGER trg_actualizar_disponibilidad
AFTER INSERT ON prestamos
FOR EACH ROW
EXECUTE FUNCTION fn_actualizar_disponibilidad();
```

¿CÓMO SE HACE?

db.php

- Credenciales para conectarse a la base de datos
- conectarBD() se usa en el resto del código
- Siempre es igual

- **OJO:** si se usa en local hay que crear credenciales en psql. Si se usa en el servidor hay que poner las credenciales de cada uno

```php
<?php
function conectarBD() {
    $host = 'localhost'; // stonebraker.ing.uc.cl
    $dbname = 'biblioteca'; // <usuario_uc>.e3
    $usuario = 'usuario'; // <usuario_uc>.e3
    $clave = 'clave'; // n° alumno

    try {
        $db = new PDO("pgsql:host=$host;dbname=$dbname", $usuario, $clave);
        $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        return $db;
    } catch (PDOException $e) {
        echo "Error de conexión: " . $e->getMessage();
        exit();
    }
}
?>
```

¿CÓMO SE HACE?

index.php

- index.php siempre será la página por defecto
- session_start() inicia o recupera sesión actual y permite acceder a variable $_SESSION
- if (isset($_SESSION[‘id_usuario’]) {} verifica si usuario ya inició sesión. Si ya se autenticó lo redirige a libros.php
- <?php if (isset($_GET[‘error’])): ?> para manejo de errores por URL (como ‘Contraseña inválida’ o ‘Debes iniciar sesión primero’)
- Formulario html recibe campos ingresados y al apretar botón se envían a validar_login.php
- Notar que pueden incluir código php en el html usando <?php ?>

```php
<?php
session_start();

if (isset($_SESSION['id_usuario'])) {
    header('Location: libros.php');
    exit();
}
?>
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Biblioteca – Login</title>
</head>
<body>
    <h1>Iniciar sesión</h1>

    <?php if (isset($_GET['error'])): ?>
        <p style="color:red"><?= htmlspecialchars($_GET['error']) ?></p>
    <?php endif; ?>

    <form action="validar_login.php" method="POST">
        <label for="nombre">Usuario:</label>
        <input type="text" id="nombre" name="nombre" required><br><br>

        <label for="password">Contraseña:</label>
        <input type="password" id="password" name="password" required><br><br>

        <button type="submit">Entrar</button>
    </form>
</body>
</html>
```

¿CÓMO SE HACE?

validar_login.php

- require_once ‘db.php’ para poder usar conectarBD()
- Se obtienen los datos envíados por el formulario del login (nombre y password).
- $db = conectarBD() crea conexión PDO
- $query busca un usuario con el mismo nombre y contraseña que se proporcione
- Se prepara la query, se hace bind de parámetros y luego se ejecuta
- $usuario = $stmt->fetch(PDO::FETCH_ASSOC) obtiene fila encontrada. Devuelve datos si se encuentra usuario, false si no
- Login exitoso guarda usuario en sesión y redirige, sino vuelve al login con mensaje de error

```php
<?php
session_start();

if (isset($_SESSION['id_usuario'])) {
    header('Location: libros.php');
    exit();
}

require_once 'db.php';

$nombre = $_POST['nombre']   ?? '';
$password = $_POST['password'] ?? '';

$db = conectarBD();

$query = "SELECT id, nombre
          FROM usuarios
          WHERE nombre = :nombre AND password = :password";
$stmt = $db->prepare($query);
$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':password', $password);
$stmt->execute();

$usuario = $stmt->fetch(PDO::FETCH_ASSOC);

if ($usuario) {
    $_SESSION['id_usuario'] = $usuario['id'];
    $_SESSION['nombre'] = $usuario['nombre'];

    header('Location: libros.php');
    exit();
} else {
    header('Location: index.php?error=Usuario o contraseña incorrectos');
    exit();
}
?>
```

¿CÓMO SE HACE?

libros.php

- Verificación de autenticación y conexión a base de datos igual que antes
- Query usa VIEW creada en libros_disponibles.sql
- Se prepara query, ejecuta y se hace fetchAll para obtener todos los libros de la vista

```php
<?php
session_start();

if (!isset($_SESSION['id_usuario'])) {
    header('Location: index.php?error=Debes iniciar sesión primero');
    exit();
}

require_once 'db.php';

$db = conectarBD();

$query = "SELECT * FROM libros_disponibles";
$stmt = $db->prepare($query);
$stmt->execute();

$libros = $stmt->fetchAll(PDO::FETCH_ASSOC);
?>
```

¿CÓMO SE HACE?

libros.php

- HTML para mostrar todos los libros
- Notar el uso de variables y código php usando <?php ?>
- <?php foreach ($libros as $libro): ?> muestra todos los libros obtenidos de la query hecha, junto con su información
- <form action="prestar.php" method="POST"> envía solicitud de préstamo a prestar.php

*Código anterior php y HTML van en un mismo archivo libros.php

```php
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Biblioteca – Libros</title>
</head>
<body>
    <h1>Bienvenido, <?= htmlspecialchars($_SESSION['nombre']) ?></h1>
    <a href="logout.php">Cerrar sesión</a>

    <h2>Libros disponibles</h2>

    <?php if (isset($_GET['success'])): ?>
        <p style="color:green"><?= htmlspecialchars($_GET['success']) ?></p>
    <?php endif; ?>

    <?php if (isset($_GET['error'])): ?>
        <p style="color:red"><?= htmlspecialchars($_GET['error']) ?></p>
    <?php endif; ?>

    <?php if (empty($libros)): ?>
        <p>No hay libros disponibles.</p>
    <?php else: ?>
        <?php foreach ($libros as $libro): ?>
            <div style="border:1px solid #ccc; padding:10px; margin-bottom:10px;">
                <strong><?= htmlspecialchars($libro['titulo']) ?></strong>
                – <?= htmlspecialchars($libro['autor']) ?>

                <form action="prestar.php" method="POST">
                    <input type="hidden" name="id_libro" value="<?= $libro['id'] ?>">
                    <button type="submit">Pedir prestado</button>
                </form>
            </div>
        <?php endforeach; ?>
    <?php endif; ?>
</body>
</html>
```

¿CÓMO SE HACE?

prestar.php

- Verificación de autenticación y conexión a base de datos igual que siempre
- Primero se hace un SELECT a libros con el id_libro enviado para verificar que efectivamente exista y aún está disponible
- Verifica si no existe el libro o si ya no está disponible y muestra mensajes de error, sino sigue con la lógica

```php
<?php
session_start();

if (!isset($_SESSION['id_usuario'])) {
    header('Location: index.php?error=Debes iniciar sesión primero');
    exit();
}

require_once 'db.php';

$id_libro = $_POST['id_libro'] ?? '';
$id_usuario = $_SESSION['id_usuario'];

$db = conectarBD();

$query = "SELECT id, titulo, disponible
          FROM libros
          WHERE id = :id_libro";
$stmt = $db->prepare($query);
$stmt->bindParam(':id_libro', $id_libro);
$stmt->execute();

$libro = $stmt->fetch(PDO::FETCH_ASSOC);

if (!$libro) {
    header('Location: libros.php?error=El libro no existe');
    exit();
}

if (!$libro['disponible']) {
    header('Location: libros.php?error=Ese libro ya no está disponible');
    exit();
}
```

¿CÓMO SE HACE?

prestar.php

- Después se hace un SELECT en prestamos con el id_usuario e id_libro para verificar si el usuario ya tiene ese libro prestado o no

- Si ya existe el prestamo muestra el error, si no sigue con la lógica

```php
$query = "SELECT id
          FROM prestamos
          WHERE id_usuario = :id_usuario AND id_libro = :id_libro";
$stmt = $db->prepare($query);
$stmt->bindParam(':id_usuario', $id_usuario);
$stmt->bindParam(':id_libro', $id_libro);
$stmt->execute();

$prestamo_existente = $stmt->fetch(PDO::FETCH_ASSOC);

if ($prestamo_existente) {
    header('Location: libros.php?error=Ya tienes ese libro prestado');
    exit();
}
```

¿CÓMO SE HACE?

prestar.php

- Una vez nos aseguramos que el libro existe, aún está disponible y el usuario no ha hecho el prestamo de ese libro hacemos la transacción
- Se inicia transacción, query inserta el préstamo en la tabla, se ejecuta y se hace commit
- Después del INSERT, postgres detecta el cambio y se activa el Trigger. Trigger llama automáticamente al Stored Procedure y se actualiza disponibilidad en tabla libros
- Si transacción falla se hace rollback

```php
try {
    $db->beginTransaction();

    $query = "INSERT INTO prestamos (id_usuario, id_libro)
              VALUES (:id_usuario, :id_libro)";
    $stmt = $db->prepare($query);
    $stmt->bindParam(':id_usuario', $id_usuario);
    $stmt->bindParam(':id_libro', $id_libro);
    $stmt->execute();

    $db->commit();

    header('Location: libros.php?success=Préstamo realizado correctamente');
    exit();
} catch (Exception $e) {
    $db->rollBack();
    header('Location: libros.php?error=Error al realizar préstamo');
    exit();
}
?>
```

¿CÓMO SE HACE?

logout.php

- Se elimina la sesión actual y se redirige a index.php para un nuevo inicio de sesión

```php
<?php
session_start();
session_destroy();

header('Location: index.php');
exit();
?>
```

PARTE 5

CONSEJOS ENTREGA 3

CONSEJOS

Comandos útiles en local

*Deben tener instalado PostgreSQL

1) Entrar a psql:

```
psql
```

2) Crear base de datos y usuario (en terminal psql):

```
franciscamatte=# CREATE DATABASE nombre_database;
CREATE USER mi_usuario WITH PASSWORD 'mi_clave';
GRANT ALL PRIVILEGES ON DATABASE biblioteca TO mi_usuario;
```

3) Ejecutar archivos .sql:

```
psql -U mi_usuario -d nombre_database -f /ruta/al/archivo/schema.sql
```

```
psql -U mi_usuario -d nombre_database -f /ruta/al/archivo/SP_y_Triggers.sql
```

```
psql -U mi_usuario -d nombre_database -f /ruta/al/archivo/Vistas.sql
```

CONSEJOS

Comandos útiles en local

4) Actualizar credenciales en archivo db.php

5) Correr proyecto PHP

```
php -S localhost:8000
```

6) Subir entrega al servidor y probarla

Importante: no olvidar cambiar credenciales de db.php en entrega final en el servidor

CONSEJOS

Comandos útiles en servidor

*PostgreSQL ya viene instalado

1) Crear carpeta /Sites en la raíz

2) Subir carpeta E3 a /Sites

3) Ejecutar archivos .sql

```
psql -U mi_usuario -d nombre_database -f /ruta/al/archivo/schema.sql
```

```
psql -U mi_usuario -d nombre_database -f /ruta/al/archivo/SP_y_Triggers.sql
```

```
psql -U mi_usuario -d nombre_database -f /ruta/al/archivo/Vistas.sql
```

4) Pagina en

https://stonebraker.ing.puc.cl/<usuario_UC>.e3/E3/ruta/a/index.php

¡GRACIAS!