ReservasPRO

Guia Completa de Codigo

Sistema de Reservas Empresarial
Desarrollado con JavaScript Vanilla + localStorage

Para estudiantes de segundo semestre de Programacion Web

Tecnologias	HTML5 | Tailwind CSS | JavaScript Vanilla | localStorage
Roles del Sistema	Administrador | Operador | Cliente
Nivel	Segundo Semestre - Programacion Web
Archivos	11 archivos JavaScript + HTML + CSS

Indice de Contenidos

1. index.html — Pagina Principal

Este es el unico archivo HTML del proyecto. En una SPA (Single Page Application), solo existe un HTML que carga toda la aplicacion. Las "paginas" no son archivos separados, sino contenido que JavaScript inyecta dinamicamente en el elemento main.

Concepto clave: SPA significa que el navegador carga el HTML una sola vez y JavaScript se encarga de cambiar lo que se muestra sin recargar la pagina completa.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <!-- viewport: hace que la pagina se vea bien en celulares -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>ReservasPRO</title>
  <!-- Tailwind CSS: framework de estilos cargado desde internet (CDN) -->
  <script src="https://cdn.tailwindcss.com"></script>
  <!-- Nuestros estilos personalizados -->
  <link rel="stylesheet" href="css/styles.css">
</head>
<body class="bg-gray-50 min-h-screen">

  <!--
    CONTENEDOR PRINCIPAL DE LA SPA
    Aqui es donde JavaScript inyecta todas las vistas dinamicamente.
    role="main" y aria-label son atributos de accesibilidad para
    lectores de pantalla (personas con discapacidad visual).
  -->
  <main id="main-content" role="main" aria-label="Contenido principal">
    <div class="flex items-center justify-center min-h-screen">
      <p class="text-gray-400">Cargando...</p>
    </div>
  </main>

  <!--
    type="module": permite usar import/export de ES6 en el navegador.
    Sin esto, los archivos JavaScript no pueden importarse entre si.
    src="js/app.js": este es el archivo que arranca toda la aplicacion.
  -->
  <script type="module" src="js/app.js"></script>

</body>
</html>
```

2. css/styles.css — Estilos Globales

Archivo de estilos complementario a Tailwind. Aqui definimos animaciones y reglas globales que Tailwind no cubre por defecto.

Concepto clave: Tailwind CSS es un framework de estilos "utility-first": en lugar de escribir CSS personalizado, usas clases predefinidas directamente en el HTML como bg-blue-600 o text-white.

```
/* styles.css */
/* Tailwind CSS se carga desde el CDN en el HTML.
   Este archivo contiene estilos personalizados adicionales. */

/* box-sizing: border-box hace que el padding y border
   NO aumenten el tamano de los elementos. Es una buena practica
   aplicarlo a todos los elementos (*). */
* {
  box-sizing: border-box;
}

/* Fuente base de toda la aplicacion */
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

/* Clase .vista: se aplica a cada contenedor de vista
   para que aparezca con una animacion suave */
.vista {
  animation: fadeIn 0.2s ease-in-out;
}

/* @keyframes define la animacion: el elemento empieza
   invisible (opacity: 0) y ligeramente desplazado hacia abajo,
   luego aparece en su posicion normal */
@keyframes fadeIn {
  from { opacity: 0; transform: translateY(8px); }
  to   { opacity: 1; transform: translateY(0); }
}
```

3. js/events.js — Bus de Eventos

Este archivo implementa el patron "Event Bus" o "Bus de Eventos". Permite que diferentes partes de la aplicacion se comuniquen entre si sin conocerse directamente. Por ejemplo, cuando se crea una reserva, el servicio de reservas no necesita saber que la vista existe; simplemente lanza un evento y cualquier componente que este escuchando reaccionara.

Concepto clave: Patron Event Bus: es como un canal de radio. Un emisor transmite en una frecuencia y todos los receptores sintonizados en esa frecuencia reciben el mensaje, sin que el emisor sepa quienes son los receptores.

```
// events.js — Bus de eventos centralizado de la aplicacion
// Permite comunicacion desacoplada entre modulos

const AppEvents = {
  // CONSTANTES DE EVENTOS: nombres de los eventos disponibles.
  // Usar constantes evita errores de escritura al referenciarlos.
  BOOKINGS_UPDATED: 'app:bookings:updated', // Se lanza cuando cambian las reservas
  USER_CHANGED:     'app:user:changed',     // Se lanza cuando cambia el usuario
  VIEW_CHANGE:      'app:view:change',      // Se lanza cuando se cambia de vista

  // dispatch(): EMITIR un evento con datos opcionales
  // eventName: nombre del evento (una de las constantes de arriba)
  // detail: objeto con datos adicionales que queremos enviar
  dispatch(eventName, detail = {}) {
    // CustomEvent es una clase nativa del navegador para crear eventos personalizados
    document.dispatchEvent(new CustomEvent(eventName, { detail }));
  },

  // on(): ESCUCHAR un evento (suscribirse)
  // eventName: nombre del evento a escuchar
  // callback: funcion que se ejecuta cuando llega el evento
  on(eventName, callback) {
    document.addEventListener(eventName, callback);
  },

  // off(): DEJAR DE ESCUCHAR un evento (desuscribirse)
  // Importante para evitar que funciones se ejecuten
  // cuando ya no son necesarias (memory leaks)
  off(eventName, callback) {
    document.removeEventListener(eventName, callback);
  }
};

export default AppEvents;
```

4. js/store.js — Estado Global

El Store es la "fuente de verdad unica" de la aplicacion. Centraliza todas las operaciones de lectura y escritura en localStorage. En lugar de que cada modulo acceda directamente a localStorage, todos pasan por el Store. Cuando se guarda algo, el Store notifica automaticamente a toda la app mediante eventos.

Concepto clave: localStorage: es un espacio de almacenamiento en el navegador que persiste incluso despues de cerrar la pagina. Guarda datos como texto (strings), por eso usamos JSON.stringify() para guardar objetos y JSON.parse() para leerlos.

Nota: localStorage tiene un limite de aproximadamente 5MB por dominio.

```
// store.js — Estado global y acceso a localStorage
import AppEvents from './events.js';

const Store = {
  // ■■ RESERVAS ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

  // setBookings(): GUARDAR reservas en localStorage
  // Luego de guardar, dispara un evento para que la UI se actualice
  setBookings(bookings) {
    // JSON.stringify convierte el array de objetos a texto para poder guardarlo
    localStorage.setItem('bookings', JSON.stringify(bookings));
    // Notifica a toda la app que las reservas cambiaron
    AppEvents.dispatch(AppEvents.BOOKINGS_UPDATED, { bookings });
  },

  // getBookings(): LEER reservas desde localStorage
  // Si no hay reservas guardadas, devuelve un array vacio []
  getBookings() {
    // JSON.parse convierte el texto guardado de vuelta a un array de objetos
    return JSON.parse(localStorage.getItem('bookings') || '[]');
  },

  // ■■ USUARIOS ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

  // setUsers(): GUARDAR lista de usuarios
  setUsers(users) {
    localStorage.setItem('users', JSON.stringify(users));
  },

  // getUsers(): LEER lista de usuarios
  getUsers() {
    return JSON.parse(localStorage.getItem('users') || '[]');
  },

  // ■■ SESION ACTUAL ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

  // setCurrentUser(): GUARDAR el usuario que inicio sesion
  setCurrentUser(user) {
    localStorage.setItem('currentUser', JSON.stringify(user));
    // Notifica que el usuario activo cambio
    AppEvents.dispatch(AppEvents.USER_CHANGED, { user });
  },

  // getCurrentUser(): OBTENER el usuario con sesion activa
  // Devuelve null si nadie ha iniciado sesion
  getCurrentUser() {
    return JSON.parse(localStorage.getItem('currentUser') || 'null');
  }
};
```

```
export default Store;
```

5. js/router.js — Navegacion y Proteccion de Rutas

El router controla que vista se muestra segun la URL y los permisos del usuario. Implementa el patron "Navigation Guard": antes de mostrar cualquier vista, verifica si el usuario tiene permiso para verla. Si no tiene permiso, lo redirige automaticamente.

Concepto clave: Navigation Guard: es como un guardia de seguridad en la entrada de cada seccion. Antes de dejarte entrar, verifica tu identificacion (rol). Si no tienes el nivel adecuado, te manda de vuelta.

```
// router.js — Navegacion y proteccion de rutas por rol
import Store from './store.js';

// routeConfig: mapa declarativo de permisos por ruta
// Cada ruta define: que roles pueden acceder y a donde redirigir si no tiene acceso
const routeConfig = {
  'login':     { rolesPermitidos: ['*'],                          redireccionSinAcceso: null },
  'register':  { rolesPermitidos: ['*'],                          redireccionSinAcceso: null },
  'bookings':  { rolesPermitidos: ['cliente', 'operador', 'admin'], redireccionSinAcceso: 'login' },
  'manage':    { rolesPermitidos: ['operador', 'admin'],          redireccionSinAcceso: 'bookings' },
  'dashboard': { rolesPermitidos: ['admin'],                      redireccionSinAcceso: 'bookings' },
  'usuarios':  { rolesPermitidos: ['admin'],                      redireccionSinAcceso: 'bookings' },
};

// Donde va cada rol cuando inicia sesion exitosamente
const vistaDefaultPorRol = {
  admin:    'dashboard',
  operador: 'manage',
  cliente:  'bookings',
  guest:    'login'
};

// navigateTo(): funcion principal para cambiar de vista
// Antes de navegar, verifica si el usuario tiene permiso
export function navigateTo(vista) {
  const config = routeConfig[vista];

  // Early Return: si la ruta no existe en el mapa, ir al login
  if (!config) {
    console.warn(`Ruta "${vista}" no definida.`);
    return renderView('login');
  }

  const usuario          = Store.getCurrentUser();
  const rolActual        = usuario?.rol || 'guest'; // ?. evita error si usuario es null
  const esTodosPermitidos = config.rolesPermitidos.includes('*'); // '*' = todos
  const tieneAcceso      = esTodosPermitidos || config.rolesPermitidos.includes(rolActual);

  // Early Return: si la ruta requiere autenticacion y no hay sesion
  if (!esTodosPermitidos && !usuario) {
    return renderView('login');
  }

  // Early Return: si el rol del usuario no tiene acceso a esta ruta
  if (!tieneAcceso) {
    const vistaFallback = vistaDefaultPorRol[rolActual];
    console.warn(`Acceso denegado a "${vista}" para rol "${rolActual}".`);
    return renderView(vistaFallback);
  }

  renderView(vista); // Si pasa todos los guards, mostrar la vista
```

```
}

// renderView(): cambia el hash de la URL y dispara el evento de cambio de vista
function renderView(vista) {
  window.location.hash = vista; // Actualiza la URL: ej. http://localhost/#dashboard
  document.dispatchEvent(new CustomEvent('app:view:change', { detail: { vista } }));
}

// Escuchar cuando el usuario usa los botones atras/adelante del navegador
window.addEventListener('hashchange', () => {
  const vistaActual = window.location.hash.replace('#', '') || 'login';
  navigateTo(vistaActual);
});

// initRouter(): inicializar el router al cargar la aplicacion
export function initRouter() {
  const usuario   = Store.getCurrentUser();
  const rolActual = usuario?.rol || 'guest';
  const vistaInicial = usuario ? vistaDefaultPorRol[rolActual] : 'login';

  window.location.hash = ''; // Limpiar hash previo
  navigateTo(vistaInicial);
}
```

6. js/app.js — Orquestador Principal

Es el archivo que conecta todos los modulos. Importa todas las vistas, define que funcion de renderizado corresponde a cada ruta, y arranca el router cuando la pagina termina de cargar. Es el "director de orquesta" de la aplicacion.

Concepto clave: import/export ES6: permite dividir el codigo en multiples archivos y reutilizar funciones entre ellos. import trae funciones de otro archivo, export las hace disponibles para otros archivos.

```
// app.js — Punto de entrada y orquestador principal de la aplicacion
import AppEvents from './events.js';
import { initRouter, navigateTo } from './router.js';
import { renderAuthView } from './auth/auth.view.js';
import { renderBookingsView } from './bookings/booking.view.js';
import { renderManageView } from './bookings/manage.view.js';
import { calcularMetricas } from './dashboard/dashboard.service.js';
import { renderDashboard } from './dashboard/dashboard.view.js';

// vistas: mapa que conecta cada nombre de ruta con su funcion de renderizado
// Cuando el router dice "muestra la vista dashboard", app.js ejecuta renderDashboardView()
const vistas = {
  login:     () => renderAuthView('login'),
  register:  () => renderAuthView('register'),
  bookings:  () => renderBookingsView(),
  manage:    () => renderManageView(),
  usuarios:  () => renderUsuariosView(),
  dashboard: () => renderDashboardView()
};

// renderDashboardView(): construye y muestra el panel de administrador
function renderDashboardView() {
  const main = document.getElementById('main-content');
  // innerHTML: inyecta HTML directamente en el elemento.
  // Las template literals (backticks ``) permiten HTML multilinea con variables ${}.
  main.innerHTML = `
    <div class="max-w-5xl mx-auto px-4 py-8 vista">
      <div class="flex justify-between items-center mb-8">
        <h2 class="text-2xl font-bold text-gray-800">Panel Estadistico</h2>
        <div class="flex gap-2">
          <button id="btn-gestionar"
            class="text-sm text-blue-600 border border-blue-200 px-3 py-1.5 rounded-lg">
            Gestionar Reservas
          </button>
          <button id="btn-usuarios-dash"
            class="text-sm text-purple-600 border border-purple-200 px-3 py-1.5 rounded-lg">
            Gestionar Usuarios
          </button>
          <button id="btn-logout-dash"
            class="text-sm text-gray-400 border border-gray-200 px-3 py-1.5 rounded-lg">
            Cerrar sesion
          </button>
        </div>
      </div>
      <!-- Tarjetas de metricas: los IDs son usados por dashboard.view.js -->
      <div class="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4 gap-4 mb-8">
        <div class="bg-white rounded-2xl shadow-sm p-5">
          <p class="text-sm text-gray-400 mb-1">Total Reservas</p>
          <p id="stat-total" class="text-3xl font-bold text-blue-600">0</p>
        </div>
        <div class="bg-white rounded-2xl shadow-sm p-5">
          <p class="text-sm text-gray-400 mb-1">Pendientes</p>
```

```
          <p id="stat-pendientes" class="text-3xl font-bold text-yellow-500">0</p>
        </div>
        <div class="bg-white rounded-2xl shadow-sm p-5">
          <p class="text-sm text-gray-400 mb-1">Confirmadas</p>
          <p id="stat-confirmadas" class="text-3xl font-bold text-green-500">0</p>
        </div>
        <div class="bg-white rounded-2xl shadow-sm p-5">
          <p class="text-sm text-gray-400 mb-1">Canceladas</p>
          <p id="stat-canceladas" class="text-3xl font-bold text-red-500">0</p>
        </div>
      </div>
      <div class="bg-white rounded-2xl shadow-sm p-6">
        <h3 class="text-lg font-semibold text-gray-700 mb-4">Usuarios mas activos</h3>
        <ul id="lista-usuarios-activos" class="divide-y divide-gray-50"></ul>
      </div>
    </div>
  `;
  // addEventListener: agrega funciones que se ejecutan cuando el usuario hace clic
  document.getElementById('btn-logout-dash').addEventListener('click', () => {
    localStorage.removeItem('currentUser');
    navigateTo('login');
  });
  document.getElementById('btn-gestionar').addEventListener('click', () => navigateTo('bookings'));
  document.getElementById('btn-usuarios-dash').addEventListener('click', () => navigateTo('usuarios'));
  // Calcular metricas y renderizarlas en el DOM
  renderDashboard(calcularMetricas());
}

// renderUsuariosView(): construye la tabla de gestion de usuarios (solo admin)
function renderUsuariosView() {
  const main = document.getElementById('main-content');
  const usuarios = JSON.parse(localStorage.getItem('users') || '[]');
  const coloresRol = {
    admin: 'bg-purple-100 text-purple-700',
    operador: 'bg-blue-100 text-blue-700',
    cliente: 'bg-gray-100 text-gray-600'
  };
  main.innerHTML = `
    <div class="max-w-5xl mx-auto px-4 py-8 vista">
      <div class="flex justify-between items-center mb-8">
        <div>
          <h2 class="text-2xl font-bold text-gray-800">Gestion de Usuarios</h2>
          <p class="text-gray-400 text-sm mt-1">${usuarios.length} usuarios registrados</p>
        </div>
        <div class="flex gap-2">
          <button id="btn-volver-dashboard"
            class="text-sm text-blue-600 border border-blue-200 px-3 py-1.5 rounded-lg">
            Dashboard
          </button>
          <button id="btn-logout-usuarios"
            class="text-sm text-gray-400 border border-gray-200 px-3 py-1.5 rounded-lg">
            Cerrar sesion
          </button>
        </div>
      </div>
      <div class="bg-white rounded-2xl shadow-sm overflow-hidden">
        <table class="w-full text-sm">
          <thead class="bg-gray-50 border-b border-gray-100">
            <tr>
              <th class="text-left px-4 py-3 font-medium text-gray-500">Nombre</th>
              <th class="text-left px-4 py-3 font-medium text-gray-500">Email</th>
```

```
              <th class="text-left px-4 py-3 font-medium text-gray-500">Rol</th>
              <th class="text-left px-4 py-3 font-medium text-gray-500">Estado</th>
              <th class="text-left px-4 py-3 font-medium text-gray-500">Acciones</th>
            </tr>
          </thead>
          <tbody>
            ${usuarios.map(u => `
              <tr class="border-b border-gray-50 hover:bg-gray-50">
                <td class="px-4 py-3 font-medium text-gray-800">${u.nombre}</td>
                <td class="px-4 py-3 text-gray-500">${u.email}</td>
                <td class="px-4 py-3">
                  <!-- onchange: ejecuta la funcion cuando el select cambia de valor -->
                  <select onchange="cambiarRolUsuario('${u.uid}', this.value)"
                    class="text-xs border border-gray-200 rounded-lg px-2 py-1">
                    <option value="cliente"  ${u.rol==='cliente'  ? 'selected':''}>cliente</option>
                    <option value="operador" ${u.rol==='operador' ? 'selected':''}>operador</option>
                    <option value="admin"    ${u.rol==='admin'    ? 'selected':''}>admin</option>
                  </select>
                </td>
                <td class="px-4 py-3">
                  <span class="text-xs px-2 py-0.5 rounded-full
                    ${u.activo ? 'bg-green-100 text-green-700' : 'bg-red-100 text-red-700'}">
                    ${u.activo ? 'Activo' : 'Inactivo'}
                  </span>
                </td>
                <td class="px-4 py-3">
                  <button onclick="toggleUsuario('${u.uid}', ${u.activo})"
                    class="text-xs px-3 py-1 rounded-lg border transition-colors
                    ${u.activo
                      ? 'text-red-500 border-red-200'
                      : 'text-green-500 border-green-200'}">
                    ${u.activo ? 'Desactivar' : 'Activar'}
                  </button>
                </td>
              </tr>
            `).join('')}
          </tbody>
        </table>
      </div>
    </div>
  `;
  document.getElementById('btn-volver-dashboard').addEventListener('click', () => navigateTo('dashboard'));
  document.getElementById('btn-logout-usuarios').addEventListener('click', () => {
    localStorage.removeItem('currentUser');
    navigateTo('login');
  });
}

// Funciones globales para botones generados dinamicamente
// window.nombreFuncion: hace la funcion accesible desde el HTML inline (onclick="...")
window.cambiarRolUsuario = function(uid, nuevoRol) {
  const usuarios = JSON.parse(localStorage.getItem('users') || '[]');
  const index = usuarios.findIndex(u => u.uid === uid);
  if (index === -1) return;
  usuarios[index].rol = nuevoRol;
  localStorage.setItem('users', JSON.stringify(usuarios));
};

window.toggleUsuario = function(uid, estadoActual) {
  const usuarios = JSON.parse(localStorage.getItem('users') || '[]');
  const index = usuarios.findIndex(u => u.uid === uid);
```

```
  if (index === -1) return;
  usuarios[index].activo = !estadoActual;
  localStorage.setItem('users', JSON.stringify(usuarios));
  navigateTo('usuarios'); // Recargar la vista para reflejar el cambio
};

// Escuchar el evento de cambio de vista y ejecutar la funcion correspondiente
AppEvents.on(AppEvents.VIEW_CHANGE, (e) => {
  const { vista } = e.detail;
  const renderFn = vistas[vista];
  if (renderFn) {
    renderFn(); // Ejecutar la funcion de renderizado
  } else {
    console.warn(`Vista no registrada: "${vista}"`);
    navigateTo('login');
  }
});

// DOMContentLoaded: esperar a que el HTML cargue completamente antes de iniciar
// Si no esperamos, el elemento #main-content podria no existir todavia
document.addEventListener('DOMContentLoaded', () => {
  console.log('ReservasPRO iniciando...');
  initRouter(); // Arrancar el router
});
```

7. js/auth/auth.service.js — Logica de Autenticacion

Contiene la logica de negocio del registro e inicio de sesion. No toca el DOM (la pantalla), solo trabaja con datos. Implementa el patron Early Return para manejar errores de forma limpia.

Concepto clave: Patron Early Return: en lugar de anidar multiples if-else, retornamos (salimos) inmediatamente cuando hay un error. Esto hace el codigo mas facil de leer porque el "camino feliz" (caso exitoso) queda siempre al final.

Nota: Seguridad: en este proyecto usamos btoa() para simular un hash de contrasena con fines educativos. En produccion real NUNCA se deben guardar contrasenas en localStorage. Se debe usar un servidor con bcrypt.

```
// auth.service.js — Logica de autenticacion y registro
import Store from '../store.js';
import AppEvents from '../events.js';

// simularHash(): convierte una contrasena en texto cifrado simulado
// btoa() codifica en Base64 (NO es seguro para produccion, solo educativo)
function simularHash(texto) {
  return btoa(texto + '_salt_empresa_2024');
}

// register(): registrar un nuevo usuario en el sistema
// Recibe: { nombre, email, password, rol }
// Devuelve: { exito: true/false, error?: string, usuario?: object }
export function register({ nombre, email, password, rol = 'cliente' }) {

  // Early Return 1: verificar que todos los campos esten completos
  const camposRequeridos = [nombre, email, password];
  const hayCamposVacios = camposRequeridos.some(campo => !campo || campo.trim() === '');
  if (hayCamposVacios) {
    return { exito: false, error: 'Todos los campos son obligatorios.' };
  }

  // Early Return 2: validar formato de email con expresion regular
  // /^[...]@[...]+\.[...]+$/ = patron que describe como debe verse un email
  const formatoEmailValido = /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);
  if (!formatoEmailValido) {
    return { exito: false, error: 'El formato del email no es valido.' };
  }

  // Early Return 3: la contrasena debe tener al menos 8 caracteres
  const longitudPasswordSuficiente = password.length >= 8;
  if (!longitudPasswordSuficiente) {
    return { exito: false, error: 'La contrasena debe tener al menos 8 caracteres.' };
  }

  const usuarios = Store.getUsers();

  // Early Return 4: verificar que el email no este ya registrado
  // .some() devuelve true si AL MENOS UN elemento cumple la condicion
  const emailYaRegistrado = usuarios.some(u => u.email === email.toLowerCase());
  if (emailYaRegistrado) {
    return { exito: false, error: 'Este email ya esta registrado.' };
  }

  // Si paso todas las validaciones, crear el nuevo usuario
  const nuevoUsuario = {
    uid: `usr_${Date.now()}`, // Date.now() genera un numero unico basado en el tiempo
    nombre: nombre.trim(),
    email: email.toLowerCase().trim(),
```

```
    passwordHash: simularHash(password), // Guardar hash, no la contrasena real
    rol,
    fechaRegistro: new Date().toISOString(),
    activo: true
  };

  // Spread operator (...): copia todos los elementos del array y agrega el nuevo
  Store.setUsers([...usuarios, nuevoUsuario]);

  // Devolver el usuario sin el hash de contrasena por seguridad
  return { exito: true, usuario: { ...nuevoUsuario, passwordHash: undefined } };
}

// login(): verificar credenciales e iniciar sesion
// Recibe: { email, password, rolEsperado }
// Devuelve: { exito: true/false, error?: string, usuario?: object }
export function login({ email, password, rolEsperado }) {

  // Early Return 1: campos vacios
  const hayCredencialesVacias = !email || !password;
  if (hayCredencialesVacias) {
    return { exito: false, error: 'Email y contrasena son requeridos.' };
  }

  const usuarios = Store.getUsers();

  // Early Return 2: email no registrado
  // .find() devuelve el primer elemento que cumple la condicion, o undefined
  const usuarioEncontrado = usuarios.find(u => u.email === email.toLowerCase());
  if (!usuarioEncontrado) {
    return { exito: false, error: 'Credenciales invalidas.' };
  }

  // Early Return 3: cuenta desactivada por el administrador
  const cuentaEstaActiva = usuarioEncontrado.activo === true;
  if (!cuentaEstaActiva) {
    return { exito: false, error: 'Esta cuenta ha sido desactivada.' };
  }

  // Early Return 4: el tipo de cuenta seleccionado no coincide con el rol real
  const rolCoincide = usuarioEncontrado.rol === rolEsperado;
  if (!rolCoincide) {
    return { exito: false, error: `Esta cuenta no es de tipo "${rolEsperado}".` };
  }

  // Early Return 5: contrasena incorrecta
  const hashIngresado = simularHash(password);
  const esPasswordValido = hashIngresado === usuarioEncontrado.passwordHash;
  if (!esPasswordValido) {
    return { exito: false, error: 'Credenciales invalidas.' };
  }

  // CAMINO FELIZ: todo correcto, guardar sesion
  const sesionUsuario = {
    uid:    usuarioEncontrado.uid,
    nombre: usuarioEncontrado.nombre,
    email:  usuarioEncontrado.email,
    rol:    usuarioEncontrado.rol
    // NO guardamos passwordHash en la sesion por seguridad
  };
```

```
  Store.setCurrentUser(sesionUsuario);
  return { exito: true, usuario: sesionUsuario };
}

// logout(): cerrar sesion eliminando el currentUser de localStorage
export function logout() {
  localStorage.removeItem('currentUser');
  AppEvents.dispatch(AppEvents.USER_CHANGED, { user: null });
}
```

8. js/auth/auth.view.js — Interfaz de Login y Registro

Maneja la interfaz de usuario del login y registro. Esta vista es reutilizable: la misma funcion renderiza tanto el formulario de login como el de registro, cambiando el contenido segun el parametro "modo".

Concepto clave: Separacion de responsabilidades: auth.service.js maneja la LOGICA (datos), auth.view.js maneja la INTERFAZ (pantalla). Nunca mezclar logica de negocio con manipulacion del DOM.

```
// auth.view.js — Interfaz de login y registro de usuarios
import { login, register } from './auth.service.js';
import { navigateTo } from '../router.js';

// renderAuthView(): construye y muestra el formulario de login o registro
// modo: 'login' o 'register' — determina que formulario se muestra
export function renderAuthView(modo = 'login') {
  const main = document.getElementById('main-content');

  // innerHTML: inyecta el HTML del formulario en el contenedor principal
  // Las template literals permiten HTML con condiciones usando operador ternario: condicion ? siTrue : siFalse
  main.innerHTML = `
    <div class="min-h-screen flex items-center justify-center bg-gray-50 px-4">
      <div class="bg-white rounded-2xl shadow-sm p-8 w-full max-w-md vista">

        <h1 class="text-2xl font-bold text-blue-700 mb-1 text-center">ReservasPRO</h1>
        <p class="text-gray-400 text-sm text-center mb-6">
          ${modo === 'login' ? 'Selecciona tu tipo de cuenta e inicia sesion' : 'Crea tu cuenta'}
        </p>

        <!-- Contenedor de alertas: oculto por defecto, se muestra con JavaScript -->
        <div id="alerta-auth" role="alert" aria-live="polite"
          class="hidden mb-4 p-3 bg-red-50 border border-red-200 rounded-lg text-red-700 text-sm">
        </div>
        <div id="exito-auth" role="status" aria-live="polite"
          class="hidden mb-4 p-3 bg-green-50 border border-green-200 rounded-lg text-green-700 text-sm">
        </div>

        <form id="form-auth" novalidate class="flex flex-col gap-4">

          <!-- Selector de tipo de cuenta: aparece tanto en login como en registro -->
          <!-- Los radio buttons permiten seleccionar solo UNA opcion del grupo -->
          <div class="flex flex-col gap-2">
            <label class="text-sm font-medium text-gray-600">
              ${modo === 'login' ? 'Iniciar sesion como' : 'Tipo de cuenta'}
            </label>
            <div class="grid grid-cols-3 gap-2">
              <label class="cursor-pointer">
                <!-- peer: clase de Tailwind para estilar el hermano siguiente segun el estado -->
                <input type="radio" name="rol" value="cliente" class="hidden peer" checked>
                <div class="peer-checked:bg-blue-600 peer-checked:text-white
                            border border-gray-200 rounded-xl p-3 text-center transition-all">
                  <p class="text-xl mb-1">usuario</p>
                  <p class="text-xs font-medium">Cliente</p>
                  <p class="text-xs text-gray-400 mt-0.5">Mis reservas</p>
                </div>
              </label>
              <label class="cursor-pointer">
                <input type="radio" name="rol" value="operador" class="hidden peer">
                <div class="peer-checked:bg-blue-600 peer-checked:text-white
                            border border-gray-200 rounded-xl p-3 text-center transition-all">
                  <p class="text-xl mb-1">tools</p>
```

```
                <p class="text-xs font-medium">Operador</p>
                <p class="text-xs text-gray-400 mt-0.5">Gestionar agenda</p>
              </div>
            </label>
            <label class="cursor-pointer">
              <input type="radio" name="rol" value="admin" class="hidden peer">
              <div class="peer-checked:bg-blue-600 peer-checked:text-white
                          border border-gray-200 rounded-xl p-3 text-center transition-all">
                <p class="text-xl mb-1">settings</p>
                <p class="text-xs font-medium">Admin</p>
                <p class="text-xs text-gray-400 mt-0.5">Acceso total</p>
              </div>
            </label>
          </div>
        </div>

        <!-- Campo nombre: solo aparece en modo registro -->
        ${modo === 'register' ? `
        <div class="flex flex-col gap-1">
          <label for="input-nombre" class="text-sm font-medium text-gray-600">Nombre completo</label>
          <input type="text" id="input-nombre" placeholder="Tu nombre completo"
            class="border border-gray-200 rounded-lg px-3 py-2 text-sm focus:ring-2 focus:ring-blue-400 focus:outline-none"
            aria-required="true">
        </div>
        ` : ''}

        <div class="flex flex-col gap-1">
          <label for="input-email" class="text-sm font-medium text-gray-600">Email</label>
          <input type="email" id="input-email" placeholder="tu@email.com"
            class="border border-gray-200 rounded-lg px-3 py-2 text-sm focus:ring-2 focus:ring-blue-400 focus:outline-none"
            aria-required="true">
        </div>

        <div class="flex flex-col gap-1">
          <label for="input-password" class="text-sm font-medium text-gray-600">Contrasena</label>
          <input type="password" id="input-password"
            placeholder="${modo === 'login' ? 'Tu contrasena' : 'Minimo 8 caracteres'}"
            class="border border-gray-200 rounded-lg px-3 py-2 text-sm focus:ring-2 focus:ring-blue-400 focus:outline-none"
            aria-required="true">
        </div>

        <button type="submit"
          class="bg-blue-600 hover:bg-blue-700 text-white font-medium py-2.5 rounded-lg transition-colors text-sm mt-1">
          ${modo === 'login' ? 'Iniciar sesion' : 'Crear cuenta'}
        </button>
      </form>

      <p class="text-center text-sm text-gray-400 mt-6">
        ${modo === 'login'
          ? `No tienes cuenta? <button id="btn-cambiar-modo" class="text-blue-600 font-medium">Registrate</button>`
          : `Ya tienes cuenta? <button id="btn-cambiar-modo" class="text-blue-600 font-medium">Inicia sesion</button>`
        }
      </p>
    </div>
  </div>
`;

// Agregar listener al formulario para manejar el submit
document.getElementById('form-auth').addEventListener('submit', (e) => {
  e.preventDefault(); // Evitar que el formulario recargue la pagina
  manejarSubmitAuth(modo);
```

```
  });

  // Listener para cambiar entre login y registro
  document.getElementById('btn-cambiar-modo').addEventListener('click', () => {
    renderAuthView(modo === 'login' ? 'register' : 'login');
  });
}

// manejarSubmitAuth(): procesa el envio del formulario de login o registro
function manejarSubmitAuth(modo) {
  const alerta  = document.getElementById('alerta-auth');
  const exito   = document.getElementById('exito-auth');

  // Leer los valores de los campos del formulario
  const email    = document.getElementById('input-email')?.value;
  const password = document.getElementById('input-password')?.value;
  const nombre   = document.getElementById('input-nombre')?.value;

  // querySelector: busca el radio button que este seleccionado (checked)
  const rolSeleccionado = document.querySelector('input[name="rol"]:checked')?.value || 'cliente';

  let resultado;
  if (modo === 'login') {
    resultado = login({ email, password, rolEsperado: rolSeleccionado });
  } else {
    resultado = register({ nombre, email, password, rol: rolSeleccionado });
  }

  // Si hubo error, mostrar el mensaje y salir
  if (!resultado.exito) {
    exito.classList.add('hidden');
    alerta.textContent = resultado.error;
    alerta.classList.remove('hidden'); // Quitar clase hidden para mostrar la alerta
    return;
  }

  alerta.classList.add('hidden');

  // Si fue registro exitoso, mostrar mensaje y redirigir al login
  if (modo === 'register') {
    exito.textContent = `Cuenta creada como ${rolSeleccionado}. Ahora inicia sesion.`;
    exito.classList.remove('hidden');
    setTimeout(() => renderAuthView('login'), 1500); // Esperar 1.5s y redirigir
    return;
  }

  // Si fue login exitoso, navegar segun el rol del usuario
  const vistaDefaultPorRol = {
    admin: 'dashboard', operador: 'manage', cliente: 'bookings'
  };
  const vistaDestino = vistaDefaultPorRol[resultado.usuario.rol] || 'bookings';
  navigateTo(vistaDestino);
}
```

9. js/bookings/booking.validators.js — Validaciones

Contiene funciones puras de validacion. Una funcion pura recibe datos, los verifica y devuelve true o false, sin modificar nada externo. Al separarlas del servicio, pueden reutilizarse y testearse de forma independiente.

Concepto clave: Funcion pura: siempre que recibe los mismos parametros, devuelve el mismo resultado. No tiene efectos secundarios (no modifica variables externas, no hace peticiones, no toca el DOM).

```
// booking.validators.js — Funciones puras de validacion de reservas

// esFechaFutura(): verifica que la fecha y hora de la reserva sean en el futuro
// fecha: string en formato 'YYYY-MM-DD' (ej: '2026-03-15')
// hora: string en formato 'HH:MM' (ej: '14:30')
// Devuelve: true si es futura, false si ya paso
export function esFechaFutura(fecha, hora) {
  // Concatenamos fecha y hora para crear un objeto Date completo
  const fechaHoraReserva = new Date(`${fecha}T${hora}:00`);
  // new Date() sin argumentos = ahora mismo
  // Si la reserva es mayor que ahora, es futura
  return fechaHoraReserva > new Date();
}

// hayConflictoDeHorario(): verifica si ya existe una reserva en el mismo horario
// reservasExistentes: array con todas las reservas del sistema
// fecha, horaInicio, horaFin: datos de la nueva reserva que se quiere crear
// recurso: sala o espacio que se quiere reservar (ej: 'Sala A')
// Devuelve: true si hay conflicto, false si el horario esta libre
export function hayConflictoDeHorario(reservasExistentes, fecha, horaInicio, horaFin, recurso) {
  // .some() verifica si AL MENOS UNA reserva existente entra en conflicto
  return reservasExistentes.some(reserva => {
    // Condicion 1: debe ser el mismo recurso (misma sala)
    const mismoRecurso = reserva.recurso === recurso;
    // Condicion 2: debe ser la misma fecha
    const mismaFecha   = reserva.fecha === fecha;
    // Condicion 3: la reserva existente debe estar activa (no cancelada)
    const estadoActivo = ['pendiente', 'confirmada'].includes(reserva.estado);

    // Si alguna condicion basica falla, no hay conflicto con esta reserva
    if (!mismoRecurso || !mismaFecha || !estadoActivo) return false;

    // Algoritmo de deteccion de solapamiento de intervalos:
    // Dos rangos horarios se SOLAPAN si:
    //   el nuevo inicia ANTES de que el existente termine
    //   Y el nuevo termina DESPUES de que el existente inicie
    // Es mas facil pensar al reves: NO se solapan si uno termina antes de que el otro empiece
    const iniciaNueva  = horaInicio < reserva.horaFin;
    const terminaNueva = horaFin    > reserva.horaInicio;
    return iniciaNueva && terminaNueva;
  });
}
```

10. js/bookings/booking.service.js — CRUD de Reservas

Implementa las 4 operaciones del CRUD: Create (crear), Read (leer), Update (actualizar) y Delete (eliminar) de reservas. Tambien controla el ciclo de vida de los estados con un mapa de transiciones permitidas.

Concepto clave: CRUD: las 4 operaciones basicas de cualquier sistema de datos. Create=Crear, Read=Leer, Update=Actualizar, Delete=Eliminar. Todo sistema de informacion se basa en estas operaciones.

```
// booking.service.js — CRUD completo de reservas
import Store from '../store.js';
import { esFechaFutura, hayConflictoDeHorario } from './booking.validators.js';

// createBooking(): CREATE — crear una nueva reserva
// Aplica validaciones antes de guardar
export function createBooking({ fecha, horaInicio, horaFin, descripcion, recurso }) {
  const usuario = Store.getCurrentUser();

  // Early Returns: salir inmediatamente si hay un error
  if (!usuario)
    return { exito: false, error: 'Debes iniciar sesion para reservar.' };

  const camposCompletos = fecha && horaInicio && horaFin && recurso;
  if (!camposCompletos)
    return { exito: false, error: 'Completa todos los campos requeridos.' };

  const horaFinEsPosterior = horaFin > horaInicio;
  if (!horaFinEsPosterior)
    return { exito: false, error: 'La hora de fin debe ser posterior a la de inicio.' };

  const reservaEsFutura = esFechaFutura(fecha, horaInicio);
  if (!reservaEsFutura)
    return { exito: false, error: 'No puedes reservar en una fecha u hora pasada.' };

  const reservasExistentes = Store.getBookings();
  const existeConflicto = hayConflictoDeHorario(reservasExistentes, fecha, horaInicio, horaFin, recurso);
  if (existeConflicto)
    return { exito: false, error: `El recurso "${recurso}" ya esta reservado en ese horario.` };

  // CAMINO FELIZ: crear la reserva
  const nuevaReserva = {
    id: `res_${Date.now()}`,  // ID unico basado en timestamp
    uid: usuario.uid,          // A quien pertenece la reserva
    fecha, horaInicio, horaFin, recurso,
    estado: 'pendiente',       // Estado inicial siempre es pendiente
    descripcion: descripcion?.trim() || '',
    historialEstados: [
      { estado: 'pendiente', fecha: new Date().toISOString(), modificadoPor: usuario.uid }
    ],
    creadaEn: new Date().toISOString(),
    actualizadaEn: new Date().toISOString()
  };

  // Spread: copiar array existente y agregar la nueva reserva al final
  Store.setBookings([...reservasExistentes, nuevaReserva]);
  return { exito: true, reserva: nuevaReserva };
}

// updateBookingStatus(): UPDATE — cambiar el estado de una reserva
// Solo operadores y admins pueden ejecutar esto
// Las transiciones de estado siguen reglas estrictas de negocio
```

```
export function updateBookingStatus({ reservaId, nuevoEstado, nuevaFecha, nuevaHoraInicio, nuevaHoraFin }) {
  const usuario = Store.getCurrentUser();

  const tienePermisoDeGestion = ['operador', 'admin'].includes(usuario?.rol);
  if (!tienePermisoDeGestion)
    return { exito: false, error: 'No tienes permisos para cambiar estados.' };

  const reservas = Store.getBookings();
  const reservaActual = reservas.find(r => r.id === reservaId);
  if (!reservaActual)
    return { exito: false, error: 'Reserva no encontrada.' };

  // Mapa de transiciones: define las transiciones de estado permitidas
  // Una reserva cancelada NO puede confirmarse directamente, debe reprogramarse primero
  const transicionesPermitidas = {
    pendiente:    ['confirmada', 'cancelada'],
    confirmada:   ['cancelada'],
    cancelada:    ['reprogramada'],  // Debe reprogramarse antes de poder confirmarse
    reprogramada: ['confirmada', 'cancelada']
  };

  const estadoActual = reservaActual.estado;
  const transicionEsValida = transicionesPermitidas[estadoActual]?.includes(nuevoEstado);
  if (!transicionEsValida)
    return { exito: false, error: `No se puede pasar de "${estadoActual}" a "${nuevoEstado}".` };

  // Si va a reprogramarse, necesita nueva fecha y horario
  const requiereNuevaFecha = nuevoEstado === 'reprogramada';
  if (requiereNuevaFecha && (!nuevaFecha || !nuevaHoraInicio || !nuevaHoraFin))
    return { exito: false, error: 'Para reprogramar debes proporcionar nueva fecha y horario.' };

  // Spread operator: copiar todos los campos de la reserva y sobreescribir los que cambian
  const reservaActualizada = {
    ...reservaActual,
    estado: nuevoEstado,
    ...(requiereNuevaFecha && { fecha: nuevaFecha, horaInicio: nuevaHoraInicio, horaFin: nuevaHoraFin }),
    actualizadaEn: new Date().toISOString(),
    historialEstados: [
      ...reservaActual.historialEstados,
      { estado: nuevoEstado, fecha: new Date().toISOString(), modificadoPor: usuario.uid }
    ]
  };

  // .map(): recorre el array y reemplaza solo la reserva modificada
  const reservasActualizadas = reservas.map(r => r.id === reservaId ? reservaActualizada : r);
  Store.setBookings(reservasActualizadas);
  return { exito: true, reserva: reservaActualizada };
}

// deleteBooking(): DELETE — eliminar una reserva permanentemente
// Solo el administrador puede eliminar reservas
export function deleteBooking(reservaId) {
  const usuario = Store.getCurrentUser();

  const esAdmin = usuario?.rol === 'admin';
  if (!esAdmin)
    return { exito: false, error: 'Solo el administrador puede eliminar reservas.' };

  const reservas = Store.getBookings();
  const reservaExiste = reservas.some(r => r.id === reservaId);
  if (!reservaExiste)
```

```
    return { exito: false, error: 'La reserva no existe.' };

  // .filter(): devuelve un NUEVO array excluyendo la reserva a eliminar
  // No modifica el array original (inmutabilidad)
  const reservasFiltradas = reservas.filter(r => r.id !== reservaId);
  Store.setBookings(reservasFiltradas); // Guardar y notificar cambio
  return { exito: true, mensaje: `Reserva ${reservaId} eliminada.` };
}

// getMyBookings(): READ — obtener solo las reservas del usuario actual (cliente)
export function getMyBookings() {
  const usuario = Store.getCurrentUser();
  if (!usuario) return [];
  const todasLasReservas = Store.getBookings();
  // .filter() para mostrar solo las reservas que pertenecen al usuario actual
  const misReservas = todasLasReservas.filter(r => r.uid === usuario.uid);
  // .sort() para ordenar de mas reciente a mas antigua
  return misReservas.sort((a, b) => new Date(b.creadaEn) - new Date(a.creadaEn));
}

// getAllBookings(): READ — obtener TODAS las reservas (admin y operador)
export function getAllBookings() {
  const reservas = Store.getBookings();
  return reservas.sort((a, b) => new Date(b.creadaEn) - new Date(a.creadaEn));
}
```

11. js/bookings/booking.view.js — Interfaz de Reservas

Vista principal de reservas. Se adapta segun el rol: los clientes ven un formulario para crear reservas y su historial; los administradores ven todas las reservas con opciones de gestion (cambiar estado, eliminar). Incluye filtros por estado y un modal para cambiar estados.

Concepto clave: Vista adaptativa: la misma funcion renderBookingsView() genera interfaces diferentes segun el rol del usuario logueado. Esto evita duplicar codigo para cada rol.

Nota: Las funciones asignadas a window.nombreFuncion son accesibles desde el HTML generado dinamicamente (onclick="nombreFuncion()"). Es necesario porque el HTML se crea despues de cargar el script.

```
// booking.view.js — Interfaz de reservas para cliente y administrador
import { createBooking, getMyBookings, getAllBookings,
         deleteBooking, updateBookingStatus } from './booking.service.js';
import Store from '../store.js';
import AppEvents from '../events.js';

// renderBookingsView(): construye la vista de reservas segun el rol del usuario
export function renderBookingsView() {
  const main    = document.getElementById('main-content');
  const usuario = Store.getCurrentUser();
  const esAdmin    = usuario?.rol === 'admin';
  const esOperador = usuario?.rol === 'operador';

  // El HTML se adapta segun el rol: admin ve todas las reservas y botones de gestion,
  // el cliente solo ve su formulario y su historial
  main.innerHTML = `... (HTML adaptativo segun rol) ...`;

  // Renderizar lista de reservas inicial
  renderListaReservas('todos');
  agregarListeners();
}

// renderListaReservas(): genera las tarjetas de reservas en el DOM
// filtro: 'todos', 'pendiente', 'confirmada', 'cancelada', 'reprogramada'
function renderListaReservas(filtro = 'todos') {
  const usuario    = Store.getCurrentUser();
  const esAdmin    = usuario?.rol === 'admin';
  const esOperador = usuario?.rol === 'operador';

  // Seleccionar que reservas mostrar segun el rol
  let reservas = esAdmin || esOperador ? getAllBookings() : getMyBookings();

  // Aplicar filtro de estado si no es 'todos'
  if (filtro !== 'todos') {
    reservas = reservas.filter(r => r.estado === filtro);
  }

  // Generar HTML de cada reserva con sus botones de accion correspondientes
  // Los botones se muestran u ocultan segun el rol y el estado de la reserva
}

// Funciones globales para botones en HTML dinamico:

// abrirModalEstado(): muestra el modal para cambiar estado de una reserva
window.abrirModalEstado = function(reservaId, estadoActual) {
  // Calcular las transiciones permitidas desde el estado actual
  // Ej: desde 'pendiente' solo puede ir a 'confirmada' o 'cancelada'
  const transicionesPermitidas = {
```

```
    pendiente: ['confirmada', 'cancelada'],
    confirmada: ['cancelada'],
    cancelada: ['reprogramada'],
    reprogramada: ['confirmada', 'cancelada']
  };
  // Poblar el select del modal con las opciones validas
  const opciones = transicionesPermitidas[estadoActual] || [];
  document.getElementById('select-nuevo-estado').innerHTML =
    opciones.map(op => `<option value="${op}">${op}</option>`).join('');
  document.getElementById('modal-estado').classList.remove('hidden');
  window._reservaEnEdicion = reservaId; // Guardar ID en variable global temporal
};

// confirmarCambioEstado(): ejecuta el cambio de estado al confirmar en el modal
window.confirmarCambioEstado = function() {
  const resultado = updateBookingStatus({
    reservaId: window._reservaEnEdicion,
    nuevoEstado: document.getElementById('select-nuevo-estado').value,
    nuevaFecha: document.getElementById('nueva-fecha').value,
    nuevaHoraInicio: document.getElementById('nueva-hora-inicio').value,
    nuevaHoraFin: document.getElementById('nueva-hora-fin').value,
  });
  if (!resultado.exito) { /* mostrar error */ return; }
  cerrarModal();
};

// cancelarReservaCliente(): permite al cliente cancelar su propia reserva
window.cancelarReservaCliente = function(reservaId) {
  if (!confirm('Deseas cancelar esta reserva?')) return;
  updateBookingStatus({ reservaId, nuevoEstado: 'cancelada' });
};

// eliminarReserva(): solo admin — elimina permanentemente una reserva
window.eliminarReserva = function(reservaId) {
  if (!confirm('Seguro que deseas eliminar esta reserva?')) return;
  deleteBooking(reservaId);
};
```

12. js/bookings/manage.view.js — Agenda del Operador

Vista exclusiva del operador. Incluye un filtro de fecha para ver la agenda del dia, filtros por estado y un resumen estadistico del dia seleccionado. El operador puede confirmar, cancelar o reprogramar reservas desde esta vista.

Concepto clave: Agenda diaria: el filtro de fecha permite al operador ver unicamente las reservas de un dia especifico, facilitando la gestion del trabajo diario sin distracciones de otras fechas.

```
// manage.view.js — Vista de agenda y gestion para el operador
import { getAllBookings, updateBookingStatus } from './booking.service.js';
import Store from '../store.js';
import AppEvents from '../events.js';

// renderManageView(): construye la interfaz del operador
export function renderManageView() {
  const main    = document.getElementById('main-content');
  const usuario = Store.getCurrentUser();

  // Obtener la fecha de hoy en formato YYYY-MM-DD para precargar el filtro
  const hoy = new Date().toISOString().split('T')[0];

  main.innerHTML = `
    <div class="max-w-5xl mx-auto px-4 py-8 vista">

      <!-- Header con info del operador -->
      <div class="flex justify-between items-center mb-8">
        <div>
          <h2 class="text-2xl font-bold text-gray-800">Agenda de Reservas</h2>
          <p class="text-gray-400 text-sm mt-1">Bienvenido, ${usuario?.nombre}
            <span class="bg-blue-100 text-blue-700 text-xs px-2 rounded-full ml-1">operador</span>
          </p>
        </div>
        <button id="btn-logout-op" class="text-sm text-gray-400 border border-gray-200 px-3 py-1.5 rounded-lg">
          Cerrar sesion
        </button>
      </div>

      <!-- Selector de fecha: preseleccionado en hoy -->
      <div class="bg-white rounded-2xl shadow-sm p-4 mb-6 flex gap-3 items-center">
        <label for="filtro-fecha" class="text-sm font-medium text-gray-600">Filtrar por fecha:</label>
        <input type="date" id="filtro-fecha" value="${hoy}"
          class="border border-gray-200 rounded-lg px-3 py-2 text-sm focus:ring-2 focus:ring-blue-400 focus:outline-none">
        <!-- Botones de acceso rapido -->
        <button onclick="verAgendaHoy()" class="text-xs bg-blue-50 text-blue-600 px-3 py-2 rounded-lg">
          Ver hoy
        </button>
        <button onclick="verTodas()" class="text-xs bg-gray-50 text-gray-600 px-3 py-2 rounded-lg">
          Ver todas
        </button>
      </div>

      <!-- Filtros por estado -->
      <div class="flex gap-2 mb-4 flex-wrap">
        <button onclick="filtrarEstadoOp('todos')" id="op-filtro-todos"
          class="op-filtro-btn bg-gray-800 text-white text-xs px-3 py-1.5 rounded-full">Todos</button>
        <button onclick="filtrarEstadoOp('pendiente')" id="op-filtro-pendiente"
          class="op-filtro-btn bg-white text-gray-600 border text-xs px-3 py-1.5 rounded-full">Pendientes</button>
        <button onclick="filtrarEstadoOp('confirmada')" id="op-filtro-confirmada"
          class="op-filtro-btn bg-white text-gray-600 border text-xs px-3 py-1.5 rounded-full">Confirmadas</button>
        <button onclick="filtrarEstadoOp('cancelada')" id="op-filtro-cancelada"
```

```
          class="op-filtro-btn bg-white text-gray-600 border text-xs px-3 py-1.5 rounded-full">Canceladas</button>
        <button onclick="filtrarEstadoOp('reprogramada')" id="op-filtro-reprogramada"
          class="op-filtro-btn bg-white text-gray-600 border text-xs px-3 py-1.5 rounded-full">Reprogramadas</button>
      </div>

      <!-- Resumen del dia: se actualiza segun el filtro de fecha -->
      <div class="grid grid-cols-2 sm:grid-cols-4 gap-3 mb-6" id="resumen-dia"></div>

      <!-- Lista de reservas filtradas -->
      <div id="contenedor-reservas-op" aria-live="polite"></div>
    </div>

    <!-- Modal de gestion de estado (igual al del admin) -->
    <div id="modal-op" class="hidden fixed inset-0 bg-black bg-opacity-40 flex items-center justify-center z-50">
      ... (mismo modal que en booking.view.js) ...
    </div>
  `;

  // Listener del filtro de fecha: re-renderiza cuando cambia la fecha
  document.getElementById('filtro-fecha').addEventListener('change', () => renderReservasOp());

  // Render inicial con la agenda de hoy
  renderReservasOp();
}

// renderResumenDia(): calcula y muestra el resumen estadistico del dia
function renderResumenDia(reservas) {
  // .filter() para contar reservas por estado
  const total       = reservas.length;
  const pendientes  = reservas.filter(r => r.estado === 'pendiente').length;
  const confirmadas = reservas.filter(r => r.estado === 'confirmada').length;
  const canceladas  = reservas.filter(r => r.estado === 'cancelada').length;

  document.getElementById('resumen-dia').innerHTML = `
    <div class="bg-white rounded-xl shadow-sm p-4 text-center">
      <p class="text-xs text-gray-400 mb-1">Total</p>
      <p class="text-2xl font-bold text-gray-700">${total}</p>
    </div>
    ... (tarjetas para pendientes, confirmadas, canceladas)
  `;
}

// renderReservasOp(): filtra y muestra las reservas segun fecha y estado
function renderReservasOp() {
  const fechaSeleccionada = document.getElementById('filtro-fecha')?.value || '';
  let reservas = getAllBookings();

  // Si hay fecha seleccionada, filtrar por esa fecha
  if (fechaSeleccionada) {
    reservas = reservas.filter(r => r.fecha === fechaSeleccionada);
  }

  renderResumenDia(reservas); // Actualizar tarjetas de resumen

  // Aplicar filtro de estado adicional
  const filtroEstado = window._filtroEstadoOp || 'todos';
  if (filtroEstado !== 'todos') {
    reservas = reservas.filter(r => r.estado === filtroEstado);
  }

  // Generar HTML de cada reserva con boton "Gestionar"
```

```
}

// Funciones globales de la vista del operador
window.filtrarEstadoOp = function(estado) {
  window._filtroEstadoOp = estado;
  renderReservasOp();
};
window.verAgendaHoy = function() {
  document.getElementById('filtro-fecha').value = new Date().toISOString().split('T')[0];
  renderReservasOp();
};
window.verTodas = function() {
  document.getElementById('filtro-fecha').value = '';
  renderReservasOp();
};
```

13. js/dashboard/dashboard.service.js — Calculo de Metricas

Procesa los datos de reservas y genera un objeto con metricas listas para mostrar. Esta completamente desacoplado del DOM: solo recibe y devuelve datos, sin tocar la pantalla. Esto permite reutilizar la logica para diferentes formatos de presentacion.

Concepto clave: Desacoplamiento: separar el calculo de datos (service) del renderizado (view) permite testear la logica de metricas sin necesitar un navegador, y reutilizar los mismos datos en diferentes representaciones visuales.

```
// dashboard.service.js — Calculo de metricas para el panel estadistico
import Store from '../store.js';

// calcularMetricas(): procesa todas las reservas y devuelve un objeto con estadisticas
// No toca el DOM. Solo trabaja con datos y devuelve un objeto con las metricas.
// Devuelve: { totalReservas, conteoPorEstado, usuariosMasActivos }
export function calcularMetricas() {
  const reservas = Store.getBookings();
  const usuarios = Store.getUsers();

  // Total de reservas: simplemente la longitud del array
  const totalReservas = reservas.length;

  // conteoPorEstado: objeto que cuenta cuantas reservas hay por cada estado
  // .reduce(): recorre el array acumulando un resultado
  // Resultado esperado: { pendiente: 3, confirmada: 1, cancelada: 1 }
  const conteoPorEstado = reservas.reduce((acumulador, reserva) => {
    // Si el estado ya existe en el acumulador, sumar 1; si no, inicializar en 1
    acumulador[reserva.estado] = (acumulador[reserva.estado] || 0) + 1;
    return acumulador;
  }, {}); // {} = valor inicial del acumulador (objeto vacio)

  // reservasPorUsuario: contar cuantas reservas tiene cada usuario por su uid
  // Resultado esperado: { 'usr_001': 5, 'usr_002': 2, 'usr_003': 8 }
  const reservasPorUsuario = reservas.reduce((acumulador, reserva) => {
    acumulador[reserva.uid] = (acumulador[reserva.uid] || 0) + 1;
    return acumulador;
  }, {});

  // usuariosMasActivos: top 5 usuarios con mas reservas
  const usuariosMasActivos = Object.entries(reservasPorUsuario)
    // Object.entries(): convierte el objeto en array de pares [uid, cantidad]
    // Ejemplo: [['usr_001', 5], ['usr_002', 2], ['usr_003', 8]]
    .sort(([, cantA], [, cantB]) => cantB - cantA) // Ordenar de mayor a menor
    .slice(0, 5) // Tomar solo los primeros 5
    .map(([uid, cantidad]) => {
      // Buscar el nombre del usuario por su uid
      const datosUsuario = usuarios.find(u => u.uid === uid);
      return {
        uid,
        nombre: datosUsuario?.nombre || 'Usuario eliminado',
        cantidad
      };
    });

  return { totalReservas, conteoPorEstado, usuariosMasActivos };
}
```

14. js/dashboard/dashboard.view.js — Renderizado del Panel

Recibe el objeto de metricas calculado por dashboard.service.js y lo muestra en el DOM. Al separar el renderizado del calculo, si queremos cambiar como se ven las estadisticas, solo tocamos este archivo sin afectar la logica de calculo.

```
// dashboard.view.js — Renderizado del panel estadistico
// Esta funcion solo se encarga de mostrar datos en pantalla.
// NO calcula nada, NO accede a localStorage directamente.

// renderDashboard(): recibe un objeto con metricas y las muestra en el DOM
// metricas: objeto devuelto por calcularMetricas() de dashboard.service.js
export function renderDashboard(metricas) {
  const { totalReservas, conteoPorEstado, usuariosMasActivos } = metricas;

  // Destructuring: extraer propiedades del objeto en variables individuales
  // Es equivalente a:
  // const totalReservas = metricas.totalReservas;
  // const conteoPorEstado = metricas.conteoPorEstado;
  // const usuariosMasActivos = metricas.usuariosMasActivos;

  // Obtener referencias a los elementos del DOM por su ID
  const elTotal        = document.getElementById('stat-total');
  const elPendientes   = document.getElementById('stat-pendientes');
  const elConfirmadas  = document.getElementById('stat-confirmadas');
  const elCanceladas   = document.getElementById('stat-canceladas');
  const elLista        = document.getElementById('lista-usuarios-activos');

  // Actualizar el contenido de cada elemento si existe
  // El operador ?. (optional chaining) evita errores si el elemento no existe en el DOM
  if (elTotal)       elTotal.textContent       = totalReservas;
  if (elPendientes)  elPendientes.textContent  = conteoPorEstado.pendiente  || 0;
  if (elConfirmadas) elConfirmadas.textContent = conteoPorEstado.confirmada || 0;
  if (elCanceladas)  elCanceladas.textContent  = conteoPorEstado.cancelada  || 0;

  if (elLista) {
    if (usuariosMasActivos.length === 0) {
      elLista.innerHTML = `<li class="text-gray-400 text-sm py-3">No hay datos aun.</li>`;
      return; // Early Return: no hay nada mas que hacer
    }

    // .map(): transformar cada usuario en una fila de la lista HTML
    // .join(''): unir todos los strings del array en uno solo (sin separadores)
    elLista.innerHTML = usuariosMasActivos.map(({ nombre, cantidad }) => `
      <li class="flex justify-between items-center py-3">
        <span class="text-gray-700 font-medium">${nombre}</span>
        <span class="bg-blue-100 text-blue-700 text-xs px-2.5 py-1 rounded-full">
          ${cantidad} reserva${cantidad !== 1 ? 's' : ''}
        </span>
      </li>
    `).join('');
  }
}
```

15. js/utils/storage.utils.js — Utilidades de localStorage

Funciones utilitarias para gestionar el espacio de localStorage de forma segura. Incluye verificacion de cuota, limpieza de datos antiguos y escritura segura con manejo de errores.

> **Nota:** localStorage tiene un limite de 5MB por dominio. Si se supera, el navegador lanza un QuotaExceededError que puede hacer que la aplicacion falle silenciosamente. Estas utilidades previenen ese problema.

```
// storage.utils.js — Utilidades para gestion segura de localStorage
const LIMITE_STORAGE_BYTES = 4.5 * 1024 * 1024; // 4.5MB (margen de seguridad)
const ANTIGUEDAD_MAXIMA_MS = 365 * 24 * 60 * 60 * 1000; // 1 anio en milisegundos

// calcularUsoActualBytes(): calcula cuanto espacio ocupa localStorage actualmente
// Recorre todas las claves y suma el tamano de cada valor
export function calcularUsoActualBytes() {
  let totalBytes = 0;
  for (const clave in localStorage) {
    if (localStorage.hasOwnProperty(clave)) {
      // Cada caracter en JavaScript ocupa 2 bytes (UTF-16)
      // Por eso multiplicamos por 2
      totalBytes += (localStorage.getItem(clave).length + clave.length) * 2;
    }
  }
  return totalBytes;
}

// hayEspacioSuficiente(): verifica si hay espacio para guardar nuevos datos
// datosNuevos: el objeto que se quiere guardar
// Devuelve: true si hay espacio, false si esta lleno
export function hayEspacioSuficiente(datosNuevos) {
  const bytesActuales = calcularUsoActualBytes();
  // Calcular cuanto ocuparian los nuevos datos
  const bytesNuevos = JSON.stringify(datosNuevos).length * 2;
  return (bytesActuales + bytesNuevos) < LIMITE_STORAGE_BYTES;
}

// limpiarReservasAntiguas(): elimina reservas de mas de un anio que no esten activas
// Protege el espacio de localStorage de acumulacion de datos innecesarios
export function limpiarReservasAntiguas() {
  const reservas = JSON.parse(localStorage.getItem('bookings') || '[]');
  const ahora = Date.now(); // Timestamp actual en milisegundos

  const reservasFiltradas = reservas.filter(reserva => {
    const fechaCreacion = new Date(reserva.creadaEn).getTime();
    const esReciente    = (ahora - fechaCreacion) < ANTIGUEDAD_MAXIMA_MS;
    const estaActiva    = ['pendiente', 'confirmada'].includes(reserva.estado);
    // Conservar: reservas recientes O reservas activas (sin importar su edad)
    return esReciente || estaActiva;
  });

  const seEliminaronRegistros = reservasFiltradas.length < reservas.length;
  if (seEliminaronRegistros) {
    localStorage.setItem('bookings', JSON.stringify(reservasFiltradas));
    console.info(`Limpieza: ${reservas.length - reservasFiltradas.length} reservas antiguas eliminadas.`);
  }

  return reservasFiltradas.length;
}

// setItemSeguro(): guardar en localStorage con verificacion de espacio previa
```

```
// Si no hay espacio, intenta limpiar datos antiguos antes de fallar
// clave: nombre con que se guarda (ej: 'bookings')
// valor: el objeto a guardar
export function setItemSeguro(clave, valor) {
  const hayEspacio = hayEspacioSuficiente(valor);

  if (!hayEspacio) {
    // Intentar liberar espacio limpiando reservas antiguas
    limpiarReservasAntiguas();

    const hayEspacioTrasCleaning = hayEspacioSuficiente(valor);
    if (!hayEspacioTrasCleaning) {
      console.error('Storage lleno. No se pudo guardar:', clave);
      return { exito: false, error: 'Almacenamiento local lleno.' };
    }
  }

  try {
    // try-catch: intentar ejecutar el codigo y capturar errores
    // Si localStorage lanza QuotaExceededError, lo capturamos en el catch
    localStorage.setItem(clave, JSON.stringify(valor));
    return { exito: true };
  } catch (error) {
    console.error('Error al escribir en localStorage:', error);
    return { exito: false, error: 'Error de almacenamiento: ' + error.message };
  }
}
```

Glosario de Conceptos Clave

SPA	Single Page Application. Aplicacion web que carga una sola pagina HTML y actualiza el contenido dinamicamente con JavaScript sin recargar el navegador.
localStorage	Almacenamiento del navegador que persiste datos incluso al cerrar la pagina. Limite: ~5MB. Solo guarda strings, por eso usamos JSON.stringify/parse.
JSON	JavaScript Object Notation. Formato de texto para representar objetos y arrays. Ejemplo: {"nombre": "Ana", "rol": "admin"}.
import/export	Sistema de modulos de ES6. export hace una funcion disponible para otros archivos, import la trae al archivo actual.
Arrow Function	Forma corta de escribir funciones: const suma = (a, b) => a + b; Es equivalente a function suma(a, b) { return a + b; }.
Spread (...)	Operador que "expande" un array u objeto. [...arr, nuevoItem] crea un nuevo array con todos los elementos mas el nuevo.
Optional Chaining (?.)	Accede a propiedades de un objeto de forma segura. usuario?.rol devuelve undefined si usuario es null, en lugar de lanzar un error.
.find()	Metodo de array que devuelve el PRIMER elemento que cumple la condicion, o undefined si ninguno la cumple.
.filter()	Metodo de array que devuelve un NUEVO array con todos los elementos que cumplen la condicion. No modifica el original.
.map()	Metodo de array que transforma cada elemento y devuelve un NUEVO array con los resultados de la transformacion.
.reduce()	Metodo de array que acumula todos los elementos en un solo valor (puede ser un numero, objeto, string, etc.).
.some()	Devuelve true si AL MENOS UN elemento del array cumple la condicion.
Early Return	Patron de programacion que sale de una funcion inmediatamente cuando hay un error, evitando ifs anidados y haciendo el codigo mas legible.
CustomEvent	Clase nativa del navegador para crear eventos personalizados. Permite la comunicacion entre modulos sin acoplarlos.
Template Literal	Strings con backticks (`) que permiten HTML multilinea e interpolacion de variables con ${variable}.
CRUD	Create, Read, Update, Delete. Las 4 operaciones basicas de cualquier sistema de gestion de datos.
Rol	Nivel de permisos de un usuario. En este sistema: admin (acceso total), operador (gestiona estados), cliente (crea sus reservas).
Hash URL	La parte de la URL despues del #. En una SPA se usa para simular navegacion: http://localhost/#dashboard indica que se esta en el dashboard.

DOM	Document Object Model. Representacion del HTML en memoria que JavaScript puede modificar. document.getElementById() accede a elementos del DOM.
addEventListener	Metodo para escuchar eventos del usuario (clicks, cambios en inputs). Ejecuta una funcion cuando ocurre el evento.

Flujo de la Aplicacion y Estructura de Archivos

Como interactuan los archivos entre si:

```
FLUJO COMPLETO DE LA APLICACION
================================

1. El navegador carga index.html
   ■■■ Carga Tailwind CSS (CDN)
   ■■■ Carga js/app.js (type="module")

2. app.js se inicializa
   ■■■ Importa todos los modulos (events, router, vistas, servicios)
   ■■■ Registra el listener de AppEvents.VIEW_CHANGE
   ■■■ Llama a initRouter()

3. initRouter() (router.js)
   ■■■ Lee currentUser de localStorage (via Store)
   ■■■ Determina la vista inicial segun el rol
   ■■■ Llama a navigateTo(vistaInicial)

4. navigateTo(vista) (router.js)
   ■■■■ Verifica permisos del usuario (Navigation Guard)
   ■■■■ Si tiene permiso: dispara evento app:view:change
   ■■■■ Si NO tiene permiso: redirige a vista apropiada

5. app.js recibe el evento app:view:change
   ■■■ Busca la funcion en el objeto "vistas"
   ■■■ Ejecuta la funcion de renderizado correspondiente
   ■■■ La funcion inyecta HTML en #main-content

6. El usuario interactua (formularios, botones)
   ■■■■ Los eventos llaman a funciones del service (ej: createBooking)
   ■■■■ El service actualiza localStorage via Store
   ■■■■ Store dispara BOOKINGS_UPDATED
   ■■■■ La vista escucha el evento y se re-renderiza automaticamente

DIAGRAMA DE TRANSICIONES DE ESTADO
===================================
         [Creacion]
              |
              v
        [ PENDIENTE ] -----> [ CANCELADA ]
              |                    |
              v                    v
        [ CONFIRMADA ] -----> [ CANCELADA ]
                                   |
                                   v
                          [ REPROGRAMADA ] --> [ CONFIRMADA ]
                                           --> [ CANCELADA ]

ESTRUCTURA DE CARPETAS
=========================
reservas-spa/
■■■ index.html              <- Unico HTML (SPA)
■■■■ css/
■   ■■■ styles.css          <- Estilos globales
■■■ js/
    ■■■ app.js              <- Orquestador principal
```

```
    ├── events.js           <- Bus de eventos
    ├── store.js            <- Estado global / localStorage
    ├── router.js           <- Navegacion y permisos
    ├── auth/
    │   ├── auth.service.js <- Logica de autenticacion
    │   └── auth.view.js    <- Interfaz de login/registro
    ├── bookings/
    │   ├── booking.validators.js <- Validaciones puras
    │   ├── booking.service.js    <- CRUD de reservas
    │   ├── booking.view.js       <- Vista cliente/admin
    │   └── manage.view.js        <- Vista operador
    ├── dashboard/
    │   ├── dashboard.service.js  <- Calculo de metricas
    │   └── dashboard.view.js     <- Renderizado del panel
    └── utils/
        └── storage.utils.js      <- Utilidades localStorage

MODELO DE DATOS EN localStorage
================================
localStorage["users"]    -> Array de objetos Usuario
localStorage["bookings"] -> Array de objetos Reserva
localStorage["currentUser"] -> Objeto Usuario (sesion activa)

Estructura Usuario:
{ uid, nombre, email, passwordHash, rol, fechaRegistro, activo }

Estructura Reserva:
{ id, uid, fecha, horaInicio, horaFin, estado, descripcion,
  recurso, historialEstados[], creadaEn, actualizadaEn }
```

Checklist de Funcionalidades Implementadas

TECNOLOGIAS

- [OK] HTML5 semantico con atributos ARIA de accesibilidad
- [OK] Tailwind CSS para estilos responsive y modernos
- [OK] JavaScript Vanilla con modulos ES6 (import/export)
- [OK] Persistencia completa en localStorage

ROL ADMINISTRADOR

- [OK] Visualizar todas las reservas del sistema
- [OK] Cambiar estado de cualquier reserva
- [OK] Eliminar reservas permanentemente
- [OK] Panel estadistico con metricas en tiempo real
- [OK] Gestionar usuarios (cambiar rol, activar/desactivar)

ROL OPERADOR

- [OK] Agenda diaria con filtro de fecha
- [OK] Confirmar reservas pendientes
- [OK] Reprogramar reservas canceladas
- [OK] Filtrar reservas por estado
- [OK] Resumen estadistico del dia

ROL CLIENTE

- [OK] Registrarse con seleccion de tipo de cuenta
- [OK] Iniciar sesion con validacion de rol
- [OK] Crear reservas con validacion de fechas

[OK]	Validacion de solapamiento de horarios
[OK]	Cancelar reservas propias
[OK]	Consultar historial personal

CALIDAD DE CODIGO

[OK]	Patron Early Return en todas las validaciones
[OK]	Variables descriptivas en condiciones complejas
[OK]	Separacion de logica (service) y presentacion (view)
[OK]	Navigation Guard centralizado por roles
[OK]	Custom Events para comunicacion desacoplada
[OK]	Manejo de errores con try-catch en storage
