// ==== INÍCIO SEÇÃO - VARIÁVEIS GLOBAIS ====

let prayerTargets = [];

let archivedTargets = [];

let resolvedTargets = [];

// Removido: let localStorageKeyPrefix = ''; // Não mais necessário

let lastDisplayedTargets = [];

let currentPage = 1;

let currentArchivedPage = 1;

let currentResolvedPage = 1;

const targetsPerPage = 10;

let currentSearchTermMain = '';

let currentSearchTermArchived = '';

let currentSearchTermResolved = '';

let showDeadlineOnly = false;

// ==== FIM SEÇÃO - VARIÁVEIS GLOBAIS ====

// ==== INÍCIO SEÇÃO - AUTENTICAÇÃO (Firebase) ====

// Função para exibir informações do usuário e botão de logout

function displayUserInfo() {

const userInfoDiv = document.getElementById('user-info');

const user = firebase.auth().currentUser;

if (user) {

userInfoDiv.innerHTML = `

<span>${user.email}</span>

<button id="logout-button" class="btn">Sair</button>

`;

// Adiciona o listener ao botão de logout \*após\* ele ser criado

document.getElementById('logout-button').addEventListener('click', logout);

} else {

userInfoDiv.textContent = ''; // Limpa se não houver usuário

window.location.href = 'login/login.html'; //Redireciona se não houver usuário

}

}

// Função de logout

function logout() {

firebase.auth().signOut().then(() => {

// Logout bem-sucedido. Redireciona para a página de login.

window.location.href = 'login/login.html';

}).catch((error) => {

console.error("Erro ao fazer logout:", error);

alert("Erro ao fazer logout. Tente novamente.");

});

}

//Verifica o estado da autenticação

firebase.auth().onAuthStateChanged((user) => {

if (user) {

// User is signed in, see docs for a list of available properties

// https://firebase.google.com/docs/reference/js/firebase.User

displayUserInfo(); // Exibe informações do usuário

loadData();

} else {

// User is signed out

window.location.href = 'login/login.html';

}

});

// ==== FIM SEÇÃO - AUTENTICAÇÃO ====

// ==== INÍCIO SEÇÃO - FUNÇÕES UTILITÁRIAS ====

// Função para formatar data para o formato ISO (YYYY-MM-DD)

function formatDateToISO(date) {

const year = date.getFullYear();

const month = String(date.getMonth() + 1).padStart(2, '0');

const day = String(date.getDate()).padStart(2, '0');

return `${year}-${month}-${day}`;

}

// Função para formatar data para exibição (DD/MM/YYYY)

function formatDateForDisplay(dateString) {

// Verificar se dateString é um objeto Date e convertê-lo para string ISO, se necessário

if (dateString instanceof Date) {

dateString = formatDateToISO(dateString);

}

if (!dateString || dateString.includes('NaN')) return 'Data Inválida';

const date = new Date(dateString);

if (isNaN(date.getTime())) return 'Data Inválida';

const day = String(date.getDate()).padStart(2, '0');

const month = String(date.getMonth() + 1).padStart(2, '0');

const year = date.getFullYear();

return `${day}/${month}/${year}`;

}

// Função para calcular o tempo decorrido

function timeElapsed(date) {

const now = new Date();

const targetDate = new Date(date);

const elapsed = now - targetDate;

const days = Math.floor(elapsed / (1000 \* 60 \* 60 \* 24));

if (days < 7) {

return `${days} dia(s)`;

}

const weeks = Math.floor(days / 7);

return `${weeks} semana(s)`;

}

// Função para verificar se uma data está vencida em relação à data atual

function isDateExpired(dateString) {

const today = new Date();

today.setHours(0, 0, 0, 0); // Zerar horas, minutos, segundos e milissegundos

const date = new Date(dateString);

return date < today;

}

// Função para gerar um ID único

function generateUniqueId() {

return Date.now().toString(36) + Math.random().toString(36).substr(2);

}

// ==== FIM SEÇÃO - FUNÇÕES UTILITÁRIAS ====

// ==== INÍCIO SEÇÃO - FUNÇÕES AUXILIARES ====

// Função para reidratar os alvos (converter strings de data para objetos Date)

function rehydrateTargets(targets) {

return targets.map(target => {

// Converter strings de data para objetos Date

target.date = new Date(target.date);

if (target.archivedDate) {

target.archivedDate = new Date(target.archivedDate);

}

if (target.deadlineDate) {

target.deadlineDate = new Date(target.deadlineDate);

}

if (target.observations) {

target.observations.forEach(obs => {

obs.date = new Date(obs.date);

});

}

// Aqui você pode adicionar mais lógica para associar métodos,

// se necessário, por exemplo, usando uma classe ou um protótipo.

return target;

});

}

// Função para reassociar os eventos aos botões (CORRIGIDA)

function reattachEventListeners() {

// Reassociar eventos aos botões dos alvos principais

const targetDivs = document.querySelectorAll("#targetList .target");

targetDivs.forEach(targetDiv => {

const targetId = targetDiv.querySelector(".add-observation-form").dataset.targetId;

const resolvedButton = targetDiv.querySelector(".resolved");

resolvedButton.onclick = () => markAsResolved(targetId);

const archiveButton = targetDiv.querySelector(".archive");

archiveButton.onclick = () => archiveTarget(targetId);

const addObservationButton = targetDiv.querySelector(".add-observation");

addObservationButton.onclick = () => toggleAddObservation(targetId);

});

// Reassociar eventos aos botões de alvos ARQUIVADOS (se necessário)

const archivedTargetDivs = document.querySelectorAll("#archivedList .target");

archivedTargetDivs.forEach(targetDiv => {

// Aqui você pode adicionar lógica para associar eventos a botões

// específicos dentro dos alvos arquivados, se houver.

// Por exemplo, um botão para desarquivar.

});

// Reassociar eventos aos botões de alvos RESOLVIDOS (se necessário)

const resolvedTargetDivs = document.querySelectorAll("#resolvedList .target");

resolvedTargetDivs.forEach(targetDiv => {

// Aqui você pode adicionar lógica para associar eventos a botões

// específicos dentro dos alvos resolvidos, se houver.

});

}

// Função para atualizar a data do último backup

function atualizarDataUltimoBackup() {

const ultimoBackupDiv = document.getElementById('ultimoBackup');

const user = firebase.auth().currentUser; //Obtém o usuário logado

if (user) {

const dataUltimoBackupString = localStorage.getItem(user.uid + '\_ultimoBackup');

if (dataUltimoBackupString) {

const dataUltimoBackup = new Date(dataUltimoBackupString);

const dia = dataUltimoBackup.getDate().toString().padStart(2, '0');

const mes = (dataUltimoBackup.getMonth() + 1).toString().padStart(2, '0');

const ano = dataUltimoBackup.getFullYear();

const hora = dataUltimoBackup.getHours().toString().padStart(2, '0');

const minuto = dataUltimoBackup.getMinutes().toString().padStart(2, '0');

ultimoBackupDiv.textContent = `Último backup: ${dia}/${mes}/${ano} ${hora}:${minuto}`;

} else {

// Se não houver informação de backup para o login atual, oculta o painel

ultimoBackupDiv.textContent = '';

}

} else {

ultimoBackupDiv.textContent = ''; //Limpa se não houver usuário

}

}

// ==== FIM SEÇÃO - FUNÇÕES AUXILIARES ====

// ==== INÍCIO SEÇÃO - FUNÇÕES DE MENSAGEM DE CONFIRMAÇÃO ====

// Função para mostrar a mensagem de sucesso

function showSuccessMessage(messageId) {

const message = document.getElementById(messageId);

message.classList.add('show');

// Ocultar a mensagem após 3 segundos

setTimeout(() => {

message.classList.remove('show');

}, 3000);

}

// ==== FIM SEÇÃO - FUNÇÕES DE MENSAGEM DE CONFIRMAÇÃO ====

// ==== INÍCIO SEÇÃO - MANIPULAÇÃO DE DADOS ====

// Importar dados de arquivo JSON (Melhorada - Com Diagnóstico Aprimorado e Mensagem de Sucesso)

function importData(event) {

const file = event.target.files[0];

if (!file) {

console.log("Nenhum arquivo selecionado.");

return;

}

// Extrair data e hora do nome do arquivo

const filename = file.name;

const dateRegex = /^(.\*?)\_(\d{4})(\d{2})(\d{2})\_(\d{2})(\d{2})/;

const match = filename.match(dateRegex);

let fileDate = null;

let importedLogin = null; // Variável para armazenar o login extraído do nome do arquivo

if (match) {

importedLogin = match[1]; // Captura o login do nome do arquivo

const year = parseInt(match[2], 10);

const month = parseInt(match[3], 10) - 1; // Mês começa em 0 (janeiro)

const day = parseInt(match[4], 10);

const hour = parseInt(match[5], 10);

const minute = parseInt(match[6], 10);

fileDate = new Date(year, month, day, hour, minute);

//Armazena a data do último backup para o usuário importado

const user = firebase.auth().currentUser;

if(user){

localStorage.setItem(user.uid + '\_ultimoBackup', fileDate.toISOString());

atualizarDataUltimoBackup();

}

} else {

console.log("Nome do arquivo não corresponde ao formato esperado para extrair data e hora.");

}

const reader = new FileReader();

reader.onload = function (e) {

let importSuccess = false;

try {

console.log("Conteúdo bruto do arquivo:", e.target.result);

const importedData = JSON.parse(e.target.result);

console.log("Dados importados (parsed):", importedData);

const newPrayerTargets = importedData.prayerTargets || [];

const newArchivedTargets = importedData.archivedTargets || [];

console.log("Login importado:", importedLogin);

console.log("Novos alvos de oração importados:", newPrayerTargets);

console.log("Novos alvos arquivados importados:", newArchivedTargets);

const user = firebase.auth().currentUser; //Obtém o usuário logado

if (importedLogin !== user.email) { //Usa o e-mail, muito mais confiável

if (!confirm(`O login do arquivo importado (${importedLogin}) não corresponde ao login atual (${user.email}). Deseja continuar?`)) {

console.log("Importação cancelada devido à incompatibilidade de login.");

return;

}

}

const allTargets = [...prayerTargets, ...archivedTargets];

// Verificar duplicatas com base no título, data e ID

const isDuplicate = (target) => allTargets.some(item => item.id === target.id || (item.title === target.title && item.date === target.date));

newPrayerTargets.forEach(target => {

if (!isDuplicate(target)) {

// Gerar novo ID se necessário

if (!target.id) {

console.log(`Atribuindo novo ID para o alvo: ${target.title}`);

target.id = generateUniqueId();

} else if (allTargets.some(t => t.id === target.id)) {

console.log(`Conflito de ID detectado para o alvo: ${target.title}. Gerando novo ID.`);

target.id = generateUniqueId();

}

prayerTargets.push(target);

} else {

console.log(`Alvo duplicado detectado e ignorado: ${target.title}`);

}

});

newArchivedTargets.forEach(target => {

if (!isDuplicate(target)) {

// Gerar novo ID se necessário

if (!target.id) {

console.log(`Atribuindo novo ID para o alvo arquivado: ${target.title}`);

target.id = generateUniqueId();

} else if (allTargets.some(t => t.id === target.id)) {

console.log(`Conflito de ID detectado para o alvo arquivado: ${target.title}. Gerando novo ID.`);

target.id = generateUniqueId();

}

archivedTargets.push(target);

} else {

console.log(`Alvo arquivado duplicado detectado e ignorado: ${target.title}`);

}

});

// Reidratar os alvos (converter strings de data para objetos Date)

prayerTargets = rehydrateTargets(prayerTargets);

archivedTargets = rehydrateTargets(archivedTargets);

console.log("prayerTargets após reidratação:", prayerTargets);

console.log("archivedTargets após reidratação:", archivedTargets);

// Se chegou até aqui sem exceções, a importação foi bem-sucedida

importSuccess = true;

} catch (error) {

console.error("Erro detalhado ao importar dados:", error);

console.error("Stack trace do erro:", error.stack);

alert("Erro ao importar dados. Veja o console para mais detalhes.");

} finally {

updateStorage();

// Atualizar a UI e reassociar event listeners

renderTargets();

renderArchivedTargets();

renderResolvedTargets();

refreshDailyTargets();

// Reassociar os eventos (Importante para que os botões funcionem)

reattachEventListeners();

// Mostrar mensagem de sucesso somente se a importação foi bem-sucedida

if (importSuccess) {

showSuccessMessage('importSuccessMessage');

}

}

};

reader.onerror = function (error) {

console.error("Erro ao ler o arquivo:", error);

};

reader.readAsText(file);

}

// ==== INÍCIO SEÇÃO - INICIALIZAÇÃO E LOGIN ====

//Removidas as funções setLogin e loadData (não mais necessárias, pois o login é tratado no arquivo login.js)

//Função modificada para carregar os dados do usuário

function loadData() {

const user = firebase.auth().currentUser;

if (user) {

const uid = user.uid;

// Carrega os dados do localStorage, usando o UID do usuário como prefixo.

prayerTargets = JSON.parse(localStorage.getItem(uid + "\_prayerTargets")) || [];

archivedTargets = JSON.parse(localStorage.getItem(uid + "\_archivedTargets")) || [];

resolvedTargets = archivedTargets.filter((target) => target.resolved);

// Verificar prazos de validade vencidos

checkExpiredDeadlines();

// Renderizar os dados

renderTargets();

renderArchivedTargets();

renderResolvedTargets();

refreshDailyTargets();

// Mostrar as seções principais

document.querySelector(".daily-section").style.display = "block";

document.querySelector(".form-section").style.display = "block";

document.querySelector(".daily-buttons-container").style.display = "flex";

document.getElementById("mainPanel").style.display = "none";

atualizarDataUltimoBackup(); //Chama a função para atualizar a data

} else {

// Se nenhum usuário estiver logado, redireciona para a página de login.

window.location.href = 'login/login.html';

}

}

// Inicializar o login e carregar os dados ao carregar a página

window.onload = function () {

//A autenticação é verificada no firebase.auth().onAuthStateChanged

document.getElementById('searchMain').addEventListener('input', handleSearchMain);

document.getElementById('searchArchived').addEventListener('input', handleSearchArchived);

document.getElementById('searchResolved').addEventListener('input', handleSearchResolved);

document.getElementById('showDeadlineOnly').addEventListener('change', handleDeadlineFilterChange);

document.getElementById('showExpiredOnlyMain').addEventListener('change', handleExpiredOnlyMainChange);

// carregar a página

atualizarDataUltimoBackup();//Chama a função para atualizar a data

};

// ==== FIM SEÇÃO - INICIALIZAÇÃO E LOGIN ====

// ==== INÍCIO SEÇÃO - FUNÇÕES DE RENDERIZAÇÃO ====

// Renderizar alvos principais (Modificado para incluir botão "Editar Prazo")

function renderTargets() {

const targetList = document.getElementById("targetList");

targetList.innerHTML = "";

let filteredTargets = prayerTargets;

if (showDeadlineOnly) {

filteredTargets = filteredTargets.filter(t => t.hasDeadline && !isDateExpired(t.deadlineDate));

}

const showExpiredOnlyMain = document.getElementById("showExpiredOnlyMain").checked;

if (showExpiredOnlyMain) {

filteredTargets = filteredTargets.filter(t => t.hasDeadline && isDateExpired(t.deadlineDate));

}

filteredTargets = filterTargets(filteredTargets, currentSearchTermMain);

const startIndex = (currentPage - 1) \* targetsPerPage;

const endIndex = startIndex + targetsPerPage;

const targetsToDisplay = filteredTargets.slice(startIndex, endIndex);

targetsToDisplay.forEach((target) => {

const formattedDate = formatDateForDisplay(target.date);

const deadlineTag = target.hasDeadline ? `<span class="deadline-tag ${isDateExpired(target.deadlineDate) ? 'expired' : ''}">Prazo: ${formatDateForDisplay(target.deadlineDate)}</span>` : '';

const targetDiv = document.createElement("div");

targetDiv.classList.add("target");

targetDiv.innerHTML = `

<h3>${deadlineTag} ${target.title}</h3>

<p>${target.details}</p>

<p><strong>Data:</strong> ${formattedDate}</p>

<p><strong>Tempo Decorrido:</strong> ${timeElapsed(target.date)}</p>

<p><strong>Status:</strong> Pendente</p>

<button onclick="markAsResolved('${target.id}')" class="btn resolved">Marcar como Respondido</button>

<button onclick="archiveTarget('${target.id}')" class="btn archive">Arquivar</button>

<button onclick="toggleAddObservation('${target.id}')" class="btn add-observation">Adicionar Observação</button>

${target.hasDeadline ? `<button onclick="editDeadline('${target.id}')" class="btn edit-deadline">Editar Prazo</button>` : ''}

<div class="add-observation-form" data-target-id="${target.id}" style="display: none;">

<h4 class="target-title"></h4>

<textarea placeholder="Escreva aqui a nova observação"></textarea>

<input type="date" >

<button onclick="saveObservation('${target.id}')" class="btn">Salvar Observação</button>

</div>

<div class="observations-list">

${renderObservations(target.observations)}

</div>

`;

targetList.appendChild(targetDiv);

});

renderPagination('mainPanel', currentPage, filteredTargets);

}

// Renderizar alvos arquivados

function renderArchivedTargets() {

const archivedList = document.getElementById("archivedList");

archivedList.innerHTML = "";

const filteredTargets = filterTargets(archivedTargets, currentSearchTermArchived);

const startIndex = (currentArchivedPage - 1) \* targetsPerPage;

const endIndex = startIndex + targetsPerPage;

const targetsToDisplay = filteredTargets.slice(startIndex, endIndex);

targetsToDisplay.forEach((target) => {

const formattedDate = formatDateForDisplay(target.date);

const formattedArchivedDate = formatDateForDisplay(target.archivedDate);

const archivedDiv = document.createElement("div");

archivedDiv.classList.add("target");

archivedDiv.innerHTML = `

<h3>${target.title}</h3>

<p>${target.details}</p>

<p><strong>Data Original:</strong> ${formattedDate}</p>

<p><strong>Tempo Decorrido:</strong> ${timeElapsed(target.date)}</p>

<p><strong>Status:</strong> ${target.resolved ? "Respondido" : "Arquivado"}</p>

<p><strong>Data de Arquivo:</strong> ${formattedArchivedDate}</p>

`;

archivedList.appendChild(archivedDiv);

});

renderPagination('archivedPanel', currentArchivedPage, filteredTargets);

}

// Renderizar alvos respondidos

function renderResolvedTargets() {

const resolvedList = document.getElementById("resolvedList");

resolvedList.innerHTML = "";

const filteredTargets = filterTargets(resolvedTargets, currentSearchTermResolved);

const startIndex = (currentResolvedPage - 1) \* targetsPerPage;

const endIndex = startIndex + targetsPerPage;

const targetsToDisplay = filteredTargets.slice(startIndex, endIndex);

targetsToDisplay.forEach((target) => {

const formattedDate = formatDateForDisplay(target.date);

const resolvedDate = formatDateForDisplay(target.archivedDate);

const resolvedDiv = document.createElement("div");

resolvedDiv.classList.add("target", "resolved");

resolvedDiv.innerHTML = `

<h3>${target.title}</h3>

<p>${target.details}</p>

<p><strong>Data Original:</strong> ${formattedDate}</p>

<p><strong>Tempo Decorrido:</strong> ${timeElapsed(target.date)}</p>

<p><strong>Status:</strong> Respondido</p>

<p><strong>Data de Resolução:</strong> ${resolvedDate}</p>

`;

resolvedList.appendChild(resolvedDiv);

});

renderPagination('resolvedPanel', currentResolvedPage, filteredTargets);

}

function markAsResolvedDeadline(targetId) {

const targetIndex = prayerTargets.findIndex(target => target.id === targetId);

if (targetIndex === -1) {

console.error("Alvo não encontrado.");

return;

}

const formattedDate = formatDateToISO(new Date());

prayerTargets[targetIndex].resolved = true;

prayerTargets[targetIndex].archivedDate = formattedDate;

archivedTargets.push(prayerTargets[targetIndex]);

resolvedTargets.push(prayerTargets[targetIndex]);

prayerTargets.splice(targetIndex, 1);

updateStorage();

currentPage = 1;

renderTargets();

refreshDailyTargets();

exportData();

}

function archiveTargetDeadline(targetId) {

const targetIndex = prayerTargets.findIndex(target => target.id === targetId);

if (targetIndex === -1) {

console.error("Alvo não encontrado.");

return;

}

const formattedDate = formatDateToISO(new Date());

prayerTargets[targetIndex].archivedDate = formattedDate;

archivedTargets.push(prayerTargets[targetIndex]);

prayerTargets.splice(targetIndex, 1);

updateStorage();

currentPage = 1;

renderTargets();

refreshDailyTargets();

exportData();

}

// Alterna a exibição do formulário de adição de observação (prazo de validade) e exibe o título do alvo

function toggleAddObservationDeadline(targetId) {

const form = document.querySelector(`.add-observation-form[data-target-id="${targetId}"]`);

form.style.display = form.style.display === 'none' ? 'block' : 'none';

if (form.style.display === 'block') {

const target = prayerTargets.find(t => t.id === targetId);

form.querySelector('.target-title').textContent = `Adicionando observação para: ${target.title}`;

}

}

function saveObservationDeadline(targetId) {

const form = document.querySelector(`.add-observation-form[data-target-id="${targetId}"]`);

const textarea = form.querySelector('textarea');

const dateInput = form.querySelector('input[type="date"]');

const observationText = textarea.value.trim();

const observationDateValue = dateInput.value;

if (observationText !== "") {

let observationDate = observationDateValue ? observationDateValue : formatDateToISO(new Date(new Date().getTime() + new Date().getTimezoneOffset() \* 60000));

const targetIndex = prayerTargets.findIndex(t => t.id === targetId);

if (targetIndex === -1) {

console.error("Alvo não encontrado.");

return;

}

const newObservation = {

date: observationDate,

observation: observationText,

};

prayerTargets[targetIndex].observations.push(newObservation);

// Removido: localStorage.setItem(localStorageKeyPrefix + "prayerTargets", JSON.stringify(prayerTargets));

updateStorage(); //Chama a função de atualização, que agora usa o UID

renderTargets();

textarea.value = "";

dateInput.value = "";

form.style.display = "none";

exportData();

} else {

alert("Por favor, insira o texto da observação.");

}

}

// Função para renderizar a paginação

function renderPagination(panelId, page, targets) {

const totalPages = Math.ceil(targets.length / targetsPerPage);

let paginationDiv = document.getElementById("pagination-" + panelId);

if (!paginationDiv) {

paginationDiv = document.createElement("div");

paginationDiv.id = "pagination-" + panelId;

document.getElementById(panelId).appendChild(paginationDiv);

}

paginationDiv.innerHTML = "";

if (totalPages <= 1) {

paginationDiv.style.display = 'none';

return;

}

paginationDiv.style.display = 'flex';

paginationDiv.style.justifyContent = 'center';

paginationDiv.style.margin = '10px 0';

for (let i = 1; i <= totalPages; i++) {

const pageLink = document.createElement("a");

pageLink.href = "#";

pageLink.textContent = i;

pageLink.classList.add("page-link");

if (i === page) {

pageLink.classList.add('active');

}

pageLink.addEventListener("click", (event) => {

event.preventDefault();

if (panelId === 'mainPanel') {

currentPage = i;

renderTargets();

} else if (panelId === 'archivedPanel') {

currentArchivedPage = i;

renderArchivedTargets();

} else if (panelId === 'resolvedPanel') {

currentResolvedPage = i;

renderResolvedTargets();

}

});

paginationDiv.appendChild(pageLink);

}

}

// Renderizar as observações de um alvo

function renderObservations(observations) {

if (!observations || observations.length === 0) return '';

let observationsHTML = '<h4>Observações:</h4>';

observations.forEach(obs => {

observationsHTML += `<p><strong>${formatDateForDisplay(obs.date)}:</strong> ${obs.observation}</p>`;

});

return observationsHTML;

}

// Função para alternar a exibição do formulário de adição de observação

function toggleAddObservation(targetId) {

const form = document.querySelector(`.add-observation-form[data-target-id="${targetId}"]`);

form.style.display = form.style.display === 'none' ? 'block' : 'none';

if (form.style.display === 'block') {

const target = prayerTargets.find(t => t.id === targetId);

form.querySelector('.target-title').textContent = `Adicionando observação para: ${target.title}`;

}

}

// Função para salvar a observação

function saveObservation(targetId) {

const form = document.querySelector(`.add-observation-form[data-target-id="${targetId}"]`);

const textarea = form.querySelector('textarea');

const dateInput = form.querySelector('input[type="date"]');

const observationText = textarea.value.trim();

const observationDateValue = dateInput.value;

if (observationText !== "") {

let observationDate = observationDateValue ? observationDateValue : formatDateToISO(new Date(new Date().getTime() + new Date().getTimezoneOffset() \* 60000));

const targetIndex = prayerTargets.findIndex(t => t.id === targetId);

if (targetIndex === -1) {

console.error("Alvo não encontrado.");

return;

}

const newObservation = {

date: observationDate,

observation: observationText,

};

prayerTargets[targetIndex].observations.push(newObservation);

updateStorage(); //Chama a função de atualização

renderTargets();

textarea.value = "";

dateInput.value = "";

form.style.display = "none";

exportData();

} else {

alert("Por favor, insira o texto da observação.");

}

}

// Função para lidar com a mudança no filtro de alvos por prazo

function handleDeadlineFilterChange() {

showDeadlineOnly = document.getElementById("showDeadlineOnly").checked;

currentPage = 1;

renderTargets();

}

// Função para lidar com a mudança no filtro de alvos por prazo (filtro novo)

function handleExpiredOnlyMainChange() {

currentPage = 1;

renderTargets();

}

// ==== FIM SEÇÃO - FUNÇÕES DE RENDERIZAÇÃO ====

// ==== INÍCIO SEÇÃO - MANIPULAÇÃO DE DADOS ====

// Adicionar alvo

document.getElementById('hasDeadline').addEventListener('change', function() {

const deadlineContainer = document.getElementById('deadlineContainer');

deadlineContainer.style.display = this.checked ? 'block' : 'none';

});

const form = document.getElementById("prayerForm");

form.addEventListener("submit", (e) => {

e.preventDefault();

const hasDeadline = document.getElementById("hasDeadline").checked;

const deadlineDate = hasDeadline ? formatDateToISO(new Date(document.getElementById("deadlineDate").value + "T00:00:00")) : null;

const newTarget = {

id: generateUniqueId(),

title: document.getElementById("title").value,

details: document.getElementById("details").value,

date: formatDateToISO(new Date(document.getElementById("date").value + "T00:00:00")),

resolved: false,

observations: [],

hasDeadline: hasDeadline,

deadlineDate: deadlineDate

};

prayerTargets.push(newTarget);

updateStorage(); //Chama a função de atualização

currentPage = 1;

renderTargets();

form.reset();

refreshDailyTargets();

exportData();

});

// Marcar como Respondido

function markAsResolved(targetId) {

const targetIndex = prayerTargets.findIndex(target => target.id === targetId);

if (targetIndex === -1) {

console.error("Alvo não encontrado.");

return;

}

const formattedDate = formatDateToISO(new Date());

prayerTargets[targetIndex].resolved = true;

prayerTargets[targetIndex].archivedDate = formattedDate;

archivedTargets.push(prayerTargets[targetIndex]);

resolvedTargets.push(prayerTargets[targetIndex]);

prayerTargets.splice(targetIndex, 1);

updateStorage(); //Chama a função de atualização

currentPage = 1;

renderTargets();

refreshDailyTargets();

exportData();

}

// Arquivar Alvo

function archiveTarget(targetId) {

const targetIndex = prayerTargets.findIndex(target => target.id === targetId);

if (targetIndex === -1) {

console.error("Alvo não encontrado.");

return;

}

const formattedDate = formatDateToISO(new Date());

prayerTargets[targetIndex].archivedDate = formattedDate;

archivedTargets.push(prayerTargets[targetIndex]);

prayerTargets.splice(targetIndex, 1);

updateStorage(); //Chama a função de atualização

currentPage = 1;

renderTargets();

refreshDailyTargets();

exportData();

}

// Atualizar LocalStorage. Agora usa o UID do usuário como prefixo.

function updateStorage() {

const user = firebase.auth().currentUser;

if (user) {

const uid = user.uid;

localStorage.setItem(uid + "\_prayerTargets", JSON.stringify(prayerTargets));

localStorage.setItem(uid + "\_archivedTargets", JSON.stringify(archivedTargets));

resolvedTargets = archivedTargets.filter((target) => target.resolved);

}

}

// Exportar dados para arquivo JSON

function exportData() {

const user = firebase.auth().currentUser; //Obtem o usuário logado

if(!user){

console.error("Nenhum usuário logado. Não é possível exportar.");

return;

}

const now = new Date();

const year = now.getFullYear();

const month = String(now.getMonth() + 1).padStart(2, '0');

const day = String(now.getDate()).padStart(2, '0');

const hours = String(now.getHours()).padStart(2, '0');

const minutes = String(now.getMinutes()).padStart(2, '0');

const filename = `${user.email}\_${year}${month}${day}\_${hours}${minutes}.json`; //Usa o email do usuário

const data = { login: user.email, prayerTargets, archivedTargets }; //Usa o e-mail do usuário

const dataStr = JSON.stringify(data, null, 2);

const blob = new Blob([dataStr], { type: "application/json" });

const url = URL.createObjectURL(blob);

const a = document.createElement("a");

a.href = url;

a.download = filename;

a.click();

URL.revokeObjectURL(url);

// Atualizar a data e hora do último backup no localStorage

localStorage.setItem(user.uid + '\_ultimoBackup', now.toISOString()); //Armazena com UID

atualizarDataUltimoBackup();

}

// Resetar todos os dados

function resetData() {

const user = firebase.auth().currentUser; //Obtem o usuário

if (confirm("Tem certeza de que deseja limpar a página? Esta ação não pode ser desfeita.")) {

if (confirm("Você gostaria de exportar os alvos antes de limpar?")) {

exportData();

}

prayerTargets = [];

archivedTargets = [];

resolvedTargets = [];

updateStorage(); //Chama a função de atualização

renderTargets();

renderArchivedTargets();

renderResolvedTargets();

refreshDailyTargets();

if(user){

// Limpar a informação de último backup para o usuário atual

localStorage.removeItem(user.uid + '\_ultimoBackup');

atualizarDataUltimoBackup();

}

alert("Todos os alvos foram resetados.");

}

}

// ==== FIM SEÇÃO - MANIPULAÇÃO DE DADOS ====

// ==== INÍCIO SEÇÃO - RENDERIZAÇÃO ====

// Renderizar alvos arquivados (Repetido, mas com o botão de excluir)

function renderArchivedTargets() {

const archivedList = document.getElementById("archivedList");

archivedList.innerHTML = "";

const filteredTargets = filterTargets(archivedTargets, currentSearchTermArchived);

const startIndex = (currentArchivedPage - 1) \* targetsPerPage;

const endIndex = startIndex + targetsPerPage;

const targetsToDisplay = filteredTargets.slice(startIndex, endIndex);

targetsToDisplay.forEach((target) => {

const formattedDate = formatDateForDisplay(target.date);

const formattedArchivedDate = formatDateForDisplay(target.archivedDate);

const archivedDiv = document.createElement("div");

archivedDiv.classList.add("target");

archivedDiv.innerHTML = `

<h3>${target.title}</h3>

<p>${target.details}</p>

<p><strong>Data Original:</strong> ${formattedDate}</p>

<p><strong>Tempo Decorrido:</strong> ${timeElapsed(target.date)}</p>

<p><strong>Status:</strong> ${target.resolved ? "Respondido" : "Arquivado"}</p>

<p><strong>Data de Arquivo:</strong> ${formattedArchivedDate}</p>

<button onclick="deleteArchivedTarget('${target.id}')" class="btn delete">Excluir</button>

`;

archivedList.appendChild(archivedDiv);

});

renderPagination('archivedPanel', currentArchivedPage, filteredTargets);

}

// ==== FIM SEÇÃO - RENDERIZAÇÃO ====

// ==== INÍCIO SEÇÃO - EXCLUSÃO ====

// Função para excluir permanentemente um alvo arquivado

function deleteArchivedTarget(targetId) {

if (confirm("Tem certeza de que deseja excluir este alvo permanentemente? Esta ação não pode ser desfeita.")) {

const targetIndex = archivedTargets.findIndex(target => target.id === targetId);

if (targetIndex === -1) {

console.error("Alvo não encontrado.");

return;

}

archivedTargets.splice(targetIndex, 1);

updateStorage(); //Chama a função de atualização

// Atualizar a lista de alvos resolvidos

resolvedTargets = archivedTargets.filter(target => target.resolved);

// Renderizar novamente os alvos arquivados e os resolvidos

renderArchivedTargets();

renderResolvedTargets();

// Mostrar mensagem de sucesso

showSuccessMessage('deleteSuccessMessage');

}

}

// ==== FIM SEÇÃO - EXCLUSÃO ====

// ==== INÍCIO SEÇÃO - EVENT LISTENERS ====

//Removido: Listener do botão setLoginButton (não mais necessário)

const exportDataButton = document.getElementById("exportData");

exportDataButton.addEventListener("click", exportData);

const importDataInput = document.getElementById("importData");

importDataInput.addEventListener("change", importData);

const resetDataButton = document.getElementById("resetData");

resetDataButton.addEventListener("click", resetData);

document.getElementById('viewAllTargetsButton').addEventListener('click', () => {

mainPanel.style.display = "block";

dailySection.style.display = "none";

archivedPanel.style.display = "none";

resolvedPanel.style.display = "none";

viewArchivedButton.style.display = "inline-block";

viewResolvedButton.style.display = "inline-block";

backToMainButton.style.display = "inline-block";

showDeadlineOnly = false;

document.getElementById("showDeadlineOnly").checked = false;

renderTargets();

});

const viewArchivedButton = document.getElementById("viewArchivedButton");

const viewResolvedButton = document.getElementById("viewResolvedButton");

const backToMainButton = document.getElementById("backToMainButton");

const mainPanel = document.getElementById("mainPanel");

const dailySection = document.getElementById("dailySection");

const archivedPanel = document.getElementById("archivedPanel");

const resolvedPanel = document.getElementById("resolvedPanel");

viewArchivedButton.addEventListener("click", () => {

mainPanel.style.display = "none";

dailySection.style.display = "none";

archivedPanel.style.display = "block";

resolvedPanel.style.display = "none";

viewArchivedButton.style.display = "none";

viewResolvedButton.style.display = "inline-block";

backToMainButton.style.display = "inline-block";

currentArchivedPage = 1;

renderArchivedTargets();

});

viewResolvedButton.addEventListener("click", () => {

mainPanel.style.display = "none";

dailySection.style.display = "none";

archivedPanel.style.display = "none";

resolvedPanel.style.display = "block";

viewArchivedButton.style.display = "inline-block";

viewResolvedButton.style.display = "none";

backToMainButton.style.display = "inline-block";

currentResolvedPage = 1;

renderResolvedTargets();

});

backToMainButton.addEventListener("click", () => {

mainPanel.style.display = "none";

dailySection.style.display = "block";

archivedPanel.style.display = "none";

resolvedPanel.style.display = "none";

viewArchivedButton.style.display = "inline-block";

viewResolvedButton.style.display = "inline-block";

backToMainButton.style.display = "none";

hideTargets();

currentPage = 1;

});

const copyDailyButton = document.getElementById("copyDaily");

copyDailyButton.addEventListener("click", function () {

const dailyTargetsElement = document.getElementById("dailyTargets");

if (!dailyTargetsElement) {

alert("Não foi possível encontrar os alvos diários para copiar.");

return;

}

const dailyTargetsText = Array.from(dailyTargetsElement.children).map(div => {

const title = div.querySelector('h3')?.textContent || '';

const details = div.querySelector('p:nth-of-type(1)')?.textContent || '';

const timeElapsed = div.querySelector('p:nth-of-type(2)')?.textContent || '';

const observations = Array.from(div.querySelectorAll('p'))

.slice(2)

.map(p => p.textContent)

.join('\n');

let result = `${title}\n${details}\n${timeElapsed}`;

if (observations) {

result += `\nObservações:\n${observations}`;

}

return result;

}).join('\n\n---\n\n');

navigator.clipboard.writeText(dailyTargetsText).then(() => {

alert('Alvos diários copiados para a área de transferência!');

}, (err) => {

console.error('Erro ao copiar texto: ', err);

alert('Não foi possível copiar os alvos diários, por favor tente novamente.');

});

});

document.getElementById('generateViewButton').addEventListener('click', generateViewHTML);

document.getElementById('viewDaily').addEventListener('click', generateDailyViewHTML);

document.getElementById("viewResolvedViewButton").addEventListener("click", () => {

dateRangeModal.style.display = "block";

startDateInput.value = '';

endDateInput.value = '';

});

const dateRangeModal = document.getElementById("dateRangeModal");

const closeDateRangeModalButton = document.getElementById("closeDateRangeModal");

const generateResolvedViewButton = document.getElementById("generateResolvedView");

const cancelDateRangeButton = document.getElementById("cancelDateRange");

const startDateInput = document.getElementById("startDate");

const endDateInput = document.getElementById("endDate");

closeDateRangeModalButton.addEventListener("click", () => {

dateRangeModal.style.display = "none";

});

generateResolvedViewButton.addEventListener("click", () => {

const startDate = startDateInput.value;

const endDate = endDateInput.value;

const today = new Date();

const formattedToday = formatDateToISO(today);

const adjustedEndDate = endDate || formattedToday;

generateResolvedViewHTML(startDate, adjustedEndDate);

dateRangeModal.style.display = "none";

});

cancelDateRangeButton.addEventListener("click", () => {

dateRangeModal.style.display = "none";

});

// ==== FIM SEÇÃO - EVENT LISTENERS ====

// ==== INÍCIO SEÇÃO - GERAÇÃO DE VISUALIZAÇÃO (HTML) ====

// Função para gerar o HTML com os alvos ativos

function generateViewHTML() {

// Captura o versículo atual exibido na página

const verseElement = document.getElementById('dailyVerses');

const currentVerse = verseElement ? verseElement.textContent : 'Versículo não encontrado.';

let htmlContent = `<!DOCTYPE html>

<html lang="pt-BR">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Alvos de Oração</title>

<style>

body {

font-family: 'Playfair Display', serif;

margin: 10px;

padding: 10px;

background-color: #f9f9f9;

color: #333;

font-size: 16px;

}

h1 {

text-align: center;

color: #333;

margin-bottom: 20px;

font-size: 2.5em;

}

h2 {

color: #555;

font-size: 1.75em;

margin-bottom: 10px;

}

div {

margin-bottom: 20px;

padding: 15px;

border: 1px solid #ddd;

border-radius: 5px;

background-color: #fff;

box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

}

p {

margin: 5px 0;

}

hr {

margin-top: 30px;

margin-bottom: 30px;

border: 0;

border-top: 1px solid #ddd;

}

.verse-container {

font-style: italic;

text-align: center;

margin-bottom: 20px;

color: #555;

}

.deadline-tag {

background-color: #ffcc00;

color: #333;

padding: 5px 10px;

border-radius: 5px;

margin-left: 10px;

font-size: 0.8em;

}

.expired {

background-color: #ff6666;

color: #fff;

}

@media (max-width: 768px) {

body {

font-size: 14px;

}

h1 {

font-size: 2em;

}

h2 {

font-size: 1.5em;

}

}

</style>

</head>

<body>

<h1>Alvos de Oração</h1>

<div class="verse-container">${currentVerse}</div> <!-- Inserindo o versículo aqui -->`;

if (prayerTargets.length === 0) {

htmlContent += '<p>Nenhum alvo de oração cadastrado.</p>';

} else {

prayerTargets.forEach(target => {

const formattedDate = formatDateForDisplay(target.date);

const time = timeElapsed(target.date);

const deadlineTag = target.hasDeadline ? `<span class="deadline-tag ${isDateExpired(target.deadlineDate) ? 'expired' : ''}">Prazo: ${formatDateForDisplay(target.deadlineDate)}</span>` : '';

htmlContent += `

<div>

<h2>${deadlineTag} ${target.title}</h2>

<p>${target.details}</p>

<p><strong>Data de Cadastro:</strong> ${formattedDate}</p>

<p><strong>Tempo Decorrido:</strong> ${time}</p>

`;

if (target.observations && target.observations.length > 0) {

htmlContent += `<h3>Observações:</h3>`;

target.observations.forEach(obs => {

htmlContent += `<p><strong>${formatDateForDisplay(obs.date)}:</strong> ${obs.observation}</p>`;

});

}

htmlContent += '</div><hr>';

});

}

htmlContent += `</body></html>`;

const blob = new Blob([htmlContent], { type: 'text/html' });

const url = URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

const now = new Date();

const filename = `Alvos de oração geral até o dia ${String(now.getDate()).padStart(2, '0')}-${String(now.getMonth() + 1).padStart(2, '0')}-${now.getFullYear()}.html`;

a.download = filename;

a.click();

URL.revokeObjectURL(url);

}

// Função para gerar o HTML com os alvos do dia

function generateDailyViewHTML() {

// Captura o versículo atual exibido na página

const verseElement = document.getElementById('dailyVerses');

const currentVerse = verseElement ? verseElement.textContent : 'Versículo não encontrado.';

let htmlContent = `<!DOCTYPE html>

<html lang="pt-BR">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Alvos de Oração do Dia</title>

<style>

body {

font-family: 'Playfair Display', serif;

margin: 10px;

padding: 10px;

background-color: #f9f9f9;

color: #333;

font-size: 16px;

}

h1 {

text-align: center;

color: #333;

margin-bottom: 20px;

font-size: 2.5em;

}

h2 {

color: #555;

font-size: 1.75em;

margin-bottom: 10px;

display: inline-block; /\* Para alinhar com a tag de prazo \*/

}

div {

margin-bottom: 20px;

padding: 15px;

border: 1px solid #ddd;

border-radius: 5px;

background-color: #fff;

box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

}

p {

margin: 5px 0;

}

hr {

margin-top: 30px;

margin-bottom: 30px;

border: 0;

border-top: 1px solid #ddd;

}

.verse-container {

font-style: italic;

text-align: center;

margin-bottom: 20px;

color: #555;

}

.deadline-tag {

background-color: #ffcc00;

color: #333;

padding: 5px 10px;

border-radius: 5px;

margin-right: 10px; /\* Espaço à direita da tag \*/

font-size: 0.8em;

}

.title-container {

display: flex; /\* Para alinhar título e tag na mesma linha \*/

align-items: center; /\* Para alinhar verticalmente \*/

justify-content: flex-start; /\* Alinha os itens à esquerda \*/

}

@media (max-width: 768px) {

body {

font-size: 14px;

}

h1 {

font-size: 2em;

}

h2 {

font-size: 1.5em;

}

}

</style>

</head>

<body>

<h1>Alvos de Oração do Dia</h1>

<div class="verse-container">${currentVerse}</div>`;

const dailyTargetsElement = document.getElementById("dailyTargets");

if (!dailyTargetsElement || dailyTargetsElement.children.length === 0) {

htmlContent += '<p>Nenhum alvo de oração do dia disponível.</p>';

} else {

Array.from(dailyTargetsElement.children).forEach(div => {

// Captura a tag de prazo separadamente

const deadlineTag = div.querySelector('.deadline-tag')?.outerHTML || '';

// Captura o título

const titleElement = div.querySelector('h3');

let title = titleElement ? titleElement.textContent.trim() : '';

const details = div.querySelector('p:nth-of-type(1)')?.textContent || '';

const timeElapsed = div.querySelector('p:nth-of-type(2)')?.textContent || '';

const observations = Array.from(div.querySelectorAll('h4 + p'))

.map(p => p.textContent)

.join('\n');

htmlContent += `

<div>

<div class="title-container">

${deadlineTag} <h2>${title}</h2>

</div>

<p>${details}</p>

<p>${timeElapsed}</p>

`;

if (observations) {

htmlContent += `<h4>Observações:</h4><p>${observations}</p>`;

}

htmlContent += `</div><hr>`;

});

}

htmlContent += `</body></html>`;

const blob = new Blob([htmlContent], { type: 'text/html' });

const url = URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

const now = new Date();

const filename = `Alvos de oração do dia ${String(now.getDate()).padStart(2, '0')}-${String(now.getMonth() + 1).padStart(2, '0')}-${now.getFullYear()}.html`;

a.download = filename;

a.click();

URL.revokeObjectURL(url);

}

// Função para gerar o HTML com os alvos respondidos, filtrando por data de resolução

function generateResolvedViewHTML(startDate, endDate) {

// Converte as datas de início e fim para objetos Date

const startDateObj = startDate ? new Date(startDate) : null;

const endDateObj = endDate ? new Date(endDate) : null;

// Filtra os alvos respondidos dentro do intervalo de datas especificado

const filteredResolvedTargets = resolvedTargets.filter(target => {

if (!target.resolved || !target.archivedDate) return false;

// Converte a data de resolução (archivedDate) para um objeto Date

const resolvedDateObj = new Date(target.archivedDate);

// Verifica se a data de resolução está dentro do intervalo

if (startDateObj && resolvedDateObj < startDateObj) return false;

if (endDateObj) {

endDateObj.setHours(23, 59, 59); // Ajusta para o final do dia

if (resolvedDateObj > endDateObj) return false;

}

return true;

});

// Monta o HTML para exibição

let htmlContent = `<!DOCTYPE html>

<html lang="pt-BR">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Alvos Respondidos</title>

<style>

body {

font-family: 'Playfair Display', serif;

margin: 10px;

padding: 10px;

background-color: #f9f9f9;

color: #333;

font-size: 16px;

}

h1 {

text-align: center;

color: #333;

margin-bottom: 20px;

font-size: 2.5em;

}

h2 {

color: #555;

font-size: 1.75em;

margin-bottom: 10px;

}

div {

margin-bottom: 20px;

padding: 15px;

border: 1px solid #ddd;

border-radius: 5px;

background-color: #fff;

box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

}

p {

margin: 5px 0;

}

hr {

margin-top: 30px;

margin-bottom: 30px;

border: 0;

border-top: 1px solid #ddd;

}

@media (max-width: 768px) {

body {

font-size: 14px;

}

h1 {

font-size: 2em;

}

h2 {

font-size: 1.5em;

}

}

</style>

</head>

<body>

<h1>Alvos Respondidos</h1>`;

if (filteredResolvedTargets.length === 0) {

htmlContent += '<p>Nenhum alvo respondido encontrado para o período selecionado.</p>';

} else {

filteredResolvedTargets.forEach(target => {

const formattedDate = formatDateForDisplay(target.date);

const formattedArchivedDate = formatDateForDisplay(target.archivedDate); // Formata a data de resolução

htmlContent += `

<div>

<h2>${target.title}</h2>

<p>${target.details}</p>

<p><strong>Data Original:</strong> ${formattedDate}</p>

<p><strong>Data de Resolução:</strong> ${formattedArchivedDate}</p>

</div><hr>

`;

});

}

htmlContent += `</body></html>`;

// Cria um link para download do HTML gerado

const blob = new Blob([htmlContent], { type: 'text/html' });

const url = URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

const now = new Date();

const filename = `Alvos Respondidos - ${String(now.getDate()).padStart(2, '0')}-${String(now.getMonth() + 1).padStart(2, '0')}-${now.getFullYear()}.html`;

a.download = filename;

a.click();

URL.revokeObjectURL(url);

}

// ==== FIM SEÇÃO - GERAÇÃO DE VISUALIZAÇÃO (HTML) ====

// ==== INÍCIO SEÇÃO - FUNÇÕES DE BUSCA ====

function filterTargets(targets, searchTerm) {

if (!searchTerm) return targets;

const lowerCaseSearchTerm = searchTerm.toLowerCase();

return targets.filter(target =>

target.title.toLowerCase().includes(lowerCaseSearchTerm) ||

target.details.toLowerCase().includes(lowerCaseSearchTerm) ||

(target.observations && target.observations.some(obs => obs.observation.toLowerCase().includes(lowerCaseSearchTerm)))

);

}

function handleSearchMain(event) {

currentSearchTermMain = event.target.value;

currentPage = 1;

renderTargets();

}

function handleSearchArchived(event) {

currentSearchTermArchived = event.target.value;

currentArchivedPage = 1;

renderArchivedTargets();

}

function handleSearchResolved(event) {

currentSearchTermResolved = event.target.value;

currentResolvedPage = 1;

renderResolvedTargets();

}

// ==== INÍCIO SEÇÃO - VERSÍCULOS BÍBLICOS ====

const verses = [

"Mateus 7:7-8: “Peçam, e será dado a vocês; busquem, e encontrarão; batam, e a porta será aberta a vocês. Pois todo o que pede recebe; o que busca encontra; e àquele que bate, a porta será aberta.”",

"Marcos 11:24: \"Portanto, eu digo a vocês, tudo o que pedirem em oração, creiam que já o receberam, e será de vocês.\"",

"João 14:13-14: “E eu farei o que vocês pedirem em meu nome, para que o Pai seja glorificado no Filho. O que vocês pedirem em meu nome, eu farei.”",

"Filipenses 4:6-7: “Não se preocupem com nada, mas em todas as situações, pela oração e petição, com ação de graças, apresentem seus pedidos a Deus. E a paz de Deus, que excede todo o entendimento, guardará os seus corações e as suas mentes em Cristo Jesus.”",

"1 Tessalonicenses 5:16-18: “Alegrem-se sempre, orem continuamente, deem graças em todas as circunstâncias; pois esta é a vontade de Deus para vocês em Cristo Jesus.”",

"Tiago 5:13-16: “Há alguém entre vocês que está em apuros? Que ele ore. Há alguém feliz? Que ele cante louvores. Há alguém entre vocês que está doente? Que ele chame os presbíteros da igreja para orar por ele e ungi-lo com óleo em nome do Senhor. E a oração oferecida com fé fará o doente ficar bom; o Senhor o levantará. Se ele pecou, ele será perdoado. Portanto, confessem seus pecados uns aos outros e orem uns pelos outros para que vocês possam ser curados. A oração de um justo é poderosa e eficaz.”",

"1 João 5:14-15: “Esta é a confiança que temos ao nos aproximarmos de Deus: que se pedirmos qualquer coisa de acordo com a sua vontade, ele nos ouve. E se sabemos que ele nos ouve — tudo o que pedimos — sabemos que temos o que lhe pedimos.”",

"Efésios 6:18: \"Orem no Espírito em todas as ocasiões com todo tipo de orações e pedidos. Com isso em mente, estejam alertas e sempre continuem a orar por todo o povo do Senhor.\"",

"1 Timóteo 2:1-2: \"Eu exorto, então, antes de tudo, que petições, orações, intercessões e ações de graças sejam feitas para todos os povos, para reis e todos aqueles em autoridade, para que possamos viver vidas pacíficas e tranquilas em toda a piedade e santidade.\"",

"2 Crônicas 7:14: “Se o meu povo, que se chama pelo meu nome, se humilhar, e orar, e buscar a minha face, e se desviar dos seus maus caminhos, então ouvirei dos céus, perdoarei os seus pecados, e sararei a sua terra.”",

"Salmos 34:17: “Os justos clamam, o Senhor os ouve, e os livra de todas as suas angústias.”",

"Jeremias 33:3: “Clama a mim, e responder-te-ei, e anunciar-te-ei coisas grandes e firmes que não sabes.”",

"Salmos 145:18-19: “Perto está o Senhor de todos os que o invocam, de todos os que o invocam em verdade. Ele cumprirá o desejo dos que o temem; ouvirá o seu clamor, e os salvará.”",

"Daniel 9:18: “Inclina, ó Deus meu, os ouvidos, e ouve; abre os olhos, e olha para a nossa desolação, e para a cidade que é chamada pelo teu nome; porque não lançamos as nossas súplicas perante a tua face confiados em nossas justiças, mas em tuas muitas misericórdias.”",

"Provérbios 15:29: “O Senhor está longe dos perversos, mas ouve a oração dos justos.”",

"1 Reis 18:37: “Responde-me, Senhor, responde-me, para que este povo saiba que tu, Senhor, és Deus, e que tu fizeste o coração deles voltar para ti.”",

"Isaías 65:24: “E será que antes que clamem, eu responderei; estando eles ainda falando, eu os ouvirei.”"

];

function displayRandomVerse() {

const randomIndex = Math.floor(Math.random() \* verses.length);

const verseElement = document.getElementById('dailyVerses');

verseElement.textContent = verses[randomIndex];

}

// ==== FIM SEÇÃO - VERSÍCULOS BÍBLICOS ====

// ==== INÍCIO SEÇÃO - FUNCIONALIDADE DO BOTÃO "OREI!" ====

function addPrayButtonFunctionality(dailyDiv, targetIndex) {

const prayButton = document.createElement("button");

prayButton.textContent = "Orei!";

prayButton.classList.add("pray-button");

prayButton.onclick = () => {

dailyDiv.remove();

checkIfAllPrayersDone();

};

dailyDiv.insertBefore(prayButton, dailyDiv.firstChild);

}

function checkIfAllPrayersDone() {

const dailyTargets = document.getElementById("dailyTargets");

if (dailyTargets.children.length === 0) {

displayCompletionPopup();

}

}

function displayCompletionPopup() {

const popup = document.getElementById('completionPopup');

popup.style.display = 'block';

}

// Adicionando o event listener para fechar o popup

document.getElementById('closePopup').addEventListener('click', () => {

document.getElementById('completionPopup').style.display = 'none';

});

// ==== FIM SEÇÃO - FUNCIONALIDADE DO BOTÃO "OREI!" ====

// Atualizar os alvos diários

function refreshDailyTargets() {

const dailyTargets = document.getElementById("dailyTargets");

dailyTargets.innerHTML = "";

const dailyTargetsCount = Math.min(prayerTargets.length, 10); // Mostrar até 10 alvos

// Filtrar alvos que não foram exibidos recentemente

let availableTargets = prayerTargets.filter(target => !lastDisplayedTargets.includes(target));

// Se todos os alvos foram exibidos, reseta o histórico

if (availableTargets.length === 0) {

lastDisplayedTargets = [];

availableTargets = prayerTargets.slice(); // Cria uma cópia da array para evitar modificação direta

}

// Seleciona aleatoriamente os alvos

const shuffledTargets = availableTargets.sort(() => 0.5 - Math.random());

const selectedTargets = shuffledTargets.slice(0, dailyTargetsCount);

// Atualizar o histórico de exibição

lastDisplayedTargets = [...lastDisplayedTargets, ...selectedTargets].slice(-prayerTargets.length);

selectedTargets.forEach((target, index) => {

const dailyDiv = document.createElement("div");

dailyDiv.classList.add("target");

// Construindo o HTML para incluir título, detalhes e tempo decorrido, sem a tag de prazo no título

const deadlineTag = target.hasDeadline ? `<span class="deadline-tag ${isDateExpired(target.deadlineDate) ? 'expired' : ''}">Prazo: ${formatDateForDisplay(target.deadlineDate)}</span>` : '';

let contentHTML = `

<h3>${deadlineTag} ${target.title}</h3>

<p>${target.details}</p> <!-- Inclui os detalhes (observações originais) -->

<p><strong>Tempo Decorrido:</strong> ${timeElapsed(target.date)}</p>

`;

// Adicionando observações, se existirem

if (target.observations && target.observations.length > 0) {

contentHTML += `<h4>Observações:</h4>`;

target.observations.forEach(obs => {

contentHTML += `<p><strong>${formatDateForDisplay(obs.date)}:</strong> ${obs.observation}</p>`;

});

}

dailyDiv.innerHTML = contentHTML;

dailyTargets.appendChild(dailyDiv);

// Adicionar funcionalidade ao botão "Orei!"

addPrayButtonFunctionality(dailyDiv, index);

});

// Exibir versículo aleatório

displayRandomVerse();

}

// ==== FIM SEÇÃO - FUNÇÕES DE BUSCA ====

// ==== INÍCIO SEÇÃO - EDITAR PRAZO DE VALIDADE ====

function editDeadline(targetId) {

const target = prayerTargets.find(t => t.id === targetId);

if (!target) {

console.error("Alvo não encontrado.");

return;

}

// Obter a data atual do prazo (se houver)

const currentDeadline = target.deadlineDate ? formatDateForDisplay(target.deadlineDate) : '';

// Usar um prompt para obter a nova data de prazo de validade

const newDeadline = prompt("Insira a nova data de prazo de validade (DD/MM/YYYY):", currentDeadline);

// Se o usuário cancelar ou inserir uma data inválida, a função é encerrada

if (newDeadline === null) return;

// Validar a nova data

if (!isValidDate(newDeadline)) {

alert("Data inválida. Por favor, use o formato DD/MM/YYYY.");

return;

}

// Converter a nova data para o formato ISO (YYYY-MM-DD)

const newDeadlineISO = convertToISO(newDeadline);

// Atualizar o prazo de validade do alvo

target.deadlineDate = newDeadlineISO;

// Atualizar o localStorage

updateStorage();

// Renderizar novamente os alvos

renderTargets();

alert(`Prazo de validade do alvo "${target.title}" atualizado para ${newDeadline}.`);

exportData();

}

function isValidDate(dateString) {

const parts = dateString.split('/');

if (parts.length !== 3) return false;

const day = parseInt(parts[0], 10);

const month = parseInt(parts[1], 10);

const year = parseInt(parts[2], 10);

if (isNaN(day) || isNaN(month) || isNaN(year)) return false;

if (month < 1 || month > 12) return false;

const daysInMonth = new Date(year, month, 0).getDate();

if (day < 1 || day > daysInMonth) return false;

return true;

}

function convertToISO(dateString) {

const parts = dateString.split('/');

return `${parts[2]}-${parts[1].padStart(2, '0')}-${parts[0].padStart(2, '0')}`;

}

// ==== FIM SEÇÃO - EDITAR PRAZO DE VALIDADE ====

function hideTargets(){

const targetList = document.getElementById("targetList");

targetList.innerHTML = "";

}

// Função para verificar e alertar sobre prazos de validade vencidos

function checkExpiredDeadlines() {

const expiredTargets = prayerTargets.filter(target => target.hasDeadline && isDateExpired(target.deadlineDate));

if (expiredTargets.length > 0) {

let message = 'Os seguintes alvos estão com prazo de validade vencido:\n';

expiredTargets.forEach(target => {

message += `- ${target.title}\n`;

});

alert(message);

}

}