// Import Vitest testing utilities

import { describe, it, expect, vi, beforeEach } from "vitest";

// Import storage interface and types

import { IStorage } from "../../storage";

import type { Schedule } from "@shared/schema";

// Import timezone utilities and date-fns utilities

import { toZonedTime } from "date-fns-tz";

import { addDays, parseISO, getDay, format, addMinutes } from "date-fns";

// --- Mocking Setup ---

import {

calculateAvailabilitySlots as actualCalculateAvailabilitySlots,

type AvailabilitySlot,

type AvailabilityOptions

} from './availability';

// Mock the dependency function fetchRelevantAppointmentsForDay

vi.mock('./availability', async (importActual) => {

const originalModule = await importActual() as typeof import('./availability');

return {

...originalModule,

fetchRelevantAppointmentsForDay: vi.fn(), // Mock implementation provided via beforeEach/test setup

};

});

// Import the mocked function AFTER vi.mock

import { fetchRelevantAppointmentsForDay } from './availability';

const mockedFetchRelevantAppointmentsForDay = fetchRelevantAppointmentsForDay as vi.Mock;

// --- End Mocking Setup ---

// \*\*\* FIXED: Mock the Drizzle DB instance realistically \*\*\*

const mockQueryResult = {

// Define mocks for all chainable methods used in fetchRelevantAppointmentsForDay

// Make them return 'this' (the mockQueryResult object) to allow chaining

select: vi.fn().mockReturnThis(),

from: vi.fn().mockReturnThis(),

leftJoin: vi.fn().mockReturnThis(), // Called multiple times, return this is fine

where: vi.fn().mockReturnThis(),

// Mock the final execution method(s) Drizzle might use

// Make the object returned by where() thenable/executable

// This implementation resolves with whatever the standalone mock function is set to return

then: vi.fn().mockImplementation(async (resolve) => resolve(await mockedFetchRelevantAppointmentsForDay())),

execute: vi.fn().mockImplementation(async () => await mockedFetchRelevantAppointmentsForDay()),

};

const mockDb = {

// .select() is the start of the chain, returning the chainable object

select: vi.fn(() => mockQueryResult),

// Add other top-level DB methods if needed (e.g., transactions)

$transaction: vi.fn().mockImplementation(async (cb) => await cb(mockDb))

};

// --- Test Fixtures & Helpers ---

function createMockStorage(): IStorage & { \_setFacility: Function, \_setAppointmentType: Function } {

const facilitiesMap = new Map();

const appointmentTypesMap = new Map();

const storageMock: Partial<IStorage> = {

getFacility: vi.fn().mockImplementation(async (facilityId, tenantId) => facilitiesMap.get(`${facilityId}\_${tenantId ?? 'any'}`)),

getAppointmentType: vi.fn().mockImplementation(async (appointmentTypeId) => appointmentTypesMap.get(appointmentTypeId)),

// Add ALL other IStorage method stubs...

getUsers: vi.fn().mockResolvedValue([]), getUser: vi.fn().mockResolvedValue(null),

createUser: vi.fn().mockResolvedValue({ id: 1 }), updateUser: vi.fn().mockResolvedValue({ id: 1 }),

deleteUser: vi.fn().mockResolvedValue(true), getUserByUsername: vi.fn().mockResolvedValue(null),

getUsersByOrganizationId: vi.fn().mockResolvedValue([]), getTenants: vi.fn().mockResolvedValue([]),

getTenantById: vi.fn().mockResolvedValue(null), createTenant: vi.fn().mockResolvedValue({ id: 1 }),

updateTenant: vi.fn().mockResolvedValue({ id: 1 }), deleteTenant: vi.fn().mockResolvedValue(true),

getFacilities: vi.fn().mockResolvedValue([]), createFacility: vi.fn().mockResolvedValue({ id: 1 }),

updateFacility: vi.fn().mockResolvedValue({ id: 1 }), deleteFacility: vi.fn().mockResolvedValue(true),

getFacilitiesByOrganizationId: vi.fn().mockResolvedValue([]), getOrganizationFacilities: vi.fn().mockResolvedValue([]),

addFacilityToOrganization: vi.fn().mockResolvedValue(true), removeFacilityFromOrganization: vi.fn().mockResolvedValue(true),

getFacilityTenantId: vi.fn().mockResolvedValue(null), getDocks: vi.fn().mockResolvedValue([]),

getDock: vi.fn().mockResolvedValue(null), createDock: vi.fn().mockResolvedValue({ id: 1 }),

updateDock: vi.fn().mockResolvedValue({ id: 1 }), deleteDock: vi.fn().mockResolvedValue(true),

getDocksByFacility: vi.fn().mockResolvedValue([]), getSchedules: vi.fn().mockResolvedValue([]),

getSchedule: vi.fn().mockResolvedValue(null), createSchedule: vi.fn().mockResolvedValue({ id: 1 }),

updateSchedule: vi.fn().mockResolvedValue({ id: 1 }), deleteSchedule: vi.fn().mockResolvedValue(true),

getSchedulesByDock: vi.fn().mockResolvedValue([]), getSchedulesByDateRange: vi.fn().mockResolvedValue([]),

searchSchedules: vi.fn().mockResolvedValue([]), getScheduleByConfirmationCode: vi.fn().mockResolvedValue(null),

getEnhancedSchedule: vi.fn().mockResolvedValue(null), getCarriers: vi.fn().mockResolvedValue([]),

getCarrier: vi.fn().mockResolvedValue(null), createCarrier: vi.fn().mockResolvedValue({ id: 1 }),

updateCarrier: vi.fn().mockResolvedValue({ id: 1 }), deleteCarrier: vi.fn().mockResolvedValue(true),

getNotifications: vi.fn().mockResolvedValue([]), getNotification: vi.fn().mockResolvedValue(null),

createNotification: vi.fn().mockResolvedValue({ id: 1 }), markNotificationAsRead: vi.fn().mockResolvedValue(null),

getNotificationsByUser: vi.fn().mockResolvedValue([]), getAppointmentSettings: vi.fn().mockResolvedValue(null),

createAppointmentSettings: vi.fn().mockResolvedValue({ id: 1 }), updateAppointmentSettings: vi.fn().mockResolvedValue(null),

getAppointmentTypes: vi.fn().mockResolvedValue([]), createAppointmentType: vi.fn().mockResolvedValue({ id: 1 }),

updateAppointmentType: vi.fn().mockResolvedValue({ id: 1 }), deleteAppointmentType: vi.fn().mockResolvedValue(true),

getAppointmentTypesByFacility: vi.fn().mockResolvedValue([]), getOrganizationByAppointmentTypeId: vi.fn().mockResolvedValue(null),

getAppointmentTypeWithTenant: vi.fn().mockResolvedValue(null), getDailyAvailability: vi.fn().mockResolvedValue(null),

getDailyAvailabilityByAppointmentType: vi.fn().mockResolvedValue([]), createDailyAvailability: vi.fn().mockResolvedValue({ id: 1 }),

updateDailyAvailability: vi.fn().mockResolvedValue(null), deleteDailyAvailability: vi.fn().mockResolvedValue(true),

getCustomQuestions: vi.fn().mockResolvedValue([]), getCustomQuestion: vi.fn().mockResolvedValue(null),

createCustomQuestion: vi.fn().mockResolvedValue({ id: 1 }), updateCustomQuestion: vi.fn().mockResolvedValue({ id: 1 }),

deleteCustomQuestion: vi.fn().mockResolvedValue(true), getCustomQuestionsByAppointmentType: vi.fn().mockResolvedValue([]),

getStandardQuestions: vi.fn().mockResolvedValue([]), getStandardQuestion: vi.fn().mockResolvedValue(null),

createStandardQuestion: vi.fn().mockResolvedValue({ id: 1 }), createStandardQuestionWithId: vi.fn().mockResolvedValue({ id: 1 }),

updateStandardQuestion: vi.fn().mockResolvedValue({ id: 1 }), deleteStandardQuestion: vi.fn().mockResolvedValue(true),

getStandardQuestionsByAppointmentType: vi.fn().mockResolvedValue([]), getBookingPages: vi.fn().mockResolvedValue([]),

getBookingPage: vi.fn().mockResolvedValue(null), getBookingPageBySlug: vi.fn().mockResolvedValue(null),

createBookingPage: vi.fn().mockResolvedValue({ id: 1 }), updateBookingPage: vi.fn().mockResolvedValue({ id: 1 }),

deleteBookingPage: vi.fn().mockResolvedValue(true), getBookingPagesForOrganization: vi.fn().mockResolvedValue([]),

getUserPreferences: vi.fn().mockResolvedValue(null), createUserPreferences: vi.fn().mockResolvedValue({ id: 1 }),

updateUserPreferences: vi.fn().mockResolvedValue(null), getHolidays: vi.fn().mockResolvedValue([]),

createHoliday: vi.fn().mockResolvedValue({ id: 1 }), updateHoliday: vi.fn().mockResolvedValue({ id: 1 }),

deleteHoliday: vi.fn().mockResolvedValue(true), getOrganizationModules: vi.fn().mockResolvedValue([]),

setOrganizationModuleEnabled: vi.fn().mockResolvedValue(true),

// --- End IStorage stubs ---

};

(storageMock as any).\_setFacility = (facilityId: number, tenantId: number | undefined, facility: any) => {

const key = `${facilityId}\_${tenantId ?? 'any'}`;

facilitiesMap.set(key, facility);

};

(storageMock as any).\_setAppointmentType = (appointmentTypeId: number, appointmentType: any) => {

appointmentTypesMap.set(appointmentTypeId, appointmentType);

};

return storageMock as IStorage & { \_setFacility: Function, \_setAppointmentType: Function };

}

function createFacility(overrides?: Partial<ReturnType<typeof createFacility>>) {

return {

id: 7, name: "Test Facility", timezone: "America/New\_York",

sundayOpen: false, sundayStart: null, sundayEnd: null, sundayBreakStart: null, sundayBreakEnd: null,

mondayOpen: true, mondayStart: "08:00", mondayEnd: "17:00", mondayBreakStart: "12:00", mondayBreakEnd: "13:00",

tuesdayOpen: true, tuesdayStart: "08:00", tuesdayEnd: "17:00", tuesdayBreakStart: "12:00", tuesdayBreakEnd: "13:00",

wednesdayOpen: true, wednesdayStart: "08:00", wednesdayEnd: "17:00", wednesdayBreakStart: "12:00", wednesdayBreakEnd: "13:00",

thursdayOpen: true, thursdayStart: "08:00", thursdayEnd: "17:00", thursdayBreakStart: "12:00", thursdayBreakEnd: "13:00",

fridayOpen: true, fridayStart: "08:00", fridayEnd: "17:00", fridayBreakStart: "12:00", fridayBreakEnd: "13:00",

saturdayOpen: false, saturdayStart: null, saturdayEnd: null, saturdayBreakStart: null, saturdayBreakEnd: null,

...overrides,

};

}

function createAppointmentType(overrides?: Partial<ReturnType<typeof createAppointmentType>>) {

return {

id: 17, name: "4 Hour Container Appointment", duration: 240, bufferTime: 30,

maxConcurrent: 2, tenantId: 5, allowAppointmentsThroughBreaks: false, overrideFacilityHours: false,

...overrides,

};

}

type TestAppointment = { id: number; startTime: Date; endTime: Date };

function createAppointment(startTime: Date, endTime: Date): TestAppointment {

return { id: Math.floor(Math.random() \* 1000), startTime, endTime };

}

// --- End Test Fixtures ---

// --- Start Test Suite ---

describe("calculateAvailabilitySlots", () => {

beforeEach(() => {

vi.clearAllMocks(); // Clears storage mocks etc.

// Reset the imported mock function directly

mockedFetchRelevantAppointmentsForDay.mockClear();

mockedFetchRelevantAppointmentsForDay.mockResolvedValue([]); // Default

// Reset the Drizzle chain mocks on the shared object and the initial call

mockDb.select.mockClear();

mockQueryResult.select.mockClear();

mockQueryResult.from.mockClear();

mockQueryResult.leftJoin.mockClear();

mockQueryResult.where.mockClear();

mockQueryResult.then.mockClear();

mockQueryResult.execute.mockClear();

// Re-apply default resolution pointing to the reset mock

mockQueryResult.then.mockImplementation(async (resolve) => resolve(await mockedFetchRelevantAppointmentsForDay()));

mockQueryResult.execute.mockImplementation(async () => await mockedFetchRelevantAppointmentsForDay());

});

// --- Test Cases ---

// (Keep all 17 test cases exactly as they were in Revision 7)

describe("Basic Functionality", () => { /\* ... \*/ });

describe("Concurrency & Capacity", () => { /\* ... \*/ });

describe("Appointment Type Rules", () => { /\* ... \*/ });

describe("Break Time Logic", () => { /\* ... \*/ });

describe("Timezone Handling", () => { /\* ... \*/ });

describe("Tenant Isolation", () => { /\* ... \*/ });

describe("Edge Cases", () => { /\* ... \*/ });

it("handles errors from fetchRelevantAppointmentsForDay gracefully", async () => { /\* ... \*/ });

}); // End of main describe block