LoginPage: --

import { test, Locator, Page } from '@playwright/test'

export class LoginPage {

  readonly page: Page;

  private usernameInput: Locator;

  private passwordInput: Locator;

  private loginButton: Locator;

  private admin: Locator;

  constructor(page: Page) {

    this.page = page;

    this.usernameInput = page.locator('input[name="Username"]');

    this.passwordInput = page.locator('input[name="password"]');

    this.loginButton = page.locator('button:has-text("Sign in")');

    this.admin = page.locator('//span[text()=" admin "]');

  }

  /\*\*

   \* @Test1 This method logs in the user with valid credentials.

   \*

   \* @description This method performs the login operation using the provided valid credentials. It highlights the input

   \*              fields for better visibility during interaction and fills the username and password fields. After submitting

   \*              the login form by clicking the login button, it validates the success of the login process. The login is

   \*              considered successful if there are no errors.

   \*

   \* @param {Record<string, string>} loginData - An object containing the login credentials. It includes:

   \*                                             - `ValidUserName`: The username used for login.

   \*                                             - `ValidPassword`: The password used for login.

   \*/

  async performLogin(loginData: Record<string, string>) {

    await this.page.goto('https://healthapp.yaksha.com/')

    await this.usernameInput.fill(loginData['ValidUserName']);

    await this.passwordInput.fill(loginData['ValidPassword']);

    await this.loginButton.click();

  }

}

ADTPage:--

import { test, Locator, Page } from '@playwright/test'

import testData from '../Data/PatientName.json'

export default class ADTPage {

  readonly page: Page;

  public ADT: {

    ADTLink: Locator;

    searchBar: Locator;

    patientName: Locator;

    hospitalSearchBar: Locator;

    patientCode: Locator;

    admittedPatient: Locator;

    searchbar: Locator;

    elipsis: Locator;

    change\_doctor: Locator;

    update\_button: Locator;

    select\_doctor\_error: Locator;

    first\_counter: Locator;

  };

  constructor(page: Page) {

    this.page = page;

    this.ADT = {

      ADTLink: page.locator('//a[@href="#/ADTMain"]'),

      searchBar: page.locator("#quickFilterInput"),

      hospitalSearchBar: page.locator(""),

      patientName: page.locator(""),

      patientCode: page.locator(""),

      admittedPatient: page.locator(" //a[text()=' Admitted Patients ']"),

      searchbar: page.locator(""),

      elipsis: page.locator("//button[@class='dropdown-toggle grid-btnCstm']"),

      change\_doctor: page.locator("//a[@danphe-grid-action='changedr']"),

      update\_button: page.locator("//button[text()='Update']"),

      select\_doctor\_error: page.locator(""),

      first\_counter: page.getByText("New-1 "),

    };

  }

  /\*\*

   \* @Test15 Verifies field-level error validation in the "Change Doctor" modal within the ADT module.

   \*

   \*

   \* Steps:

   \* 1. Navigate to the ADT module.

   \* 2. Click on the “Admitted Patients” tab.

   \* 3. Search for a patient using data from the PatientName.json file.

   \* 4. Click on the “...” button from the patient row and select “Change Doctor”.

   \* 5. In the modal that appears, click the update button without selecting a doctor.

   \*

   \* Expected Result:

   \* - A field-level error message should appear: "Select doctor from the list."

   \*/

  async verifyInventorySubModuleNavigation() {

    const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.ADT.ADTLink.click();

    await this.ADT.first\_counter.click();

    await this.ADT.admittedPatient.click();

    await this.ADT.searchBar.fill(testData['patientName'])

    await this.ADT.searchBar.click();

    await this.page.keyboard.press('Enter');

    await this.ADT.elipsis.click();

    await this.ADT.change\_doctor.click();

    await this.ADT.update\_button.click();

  }

}

DispensaryPage:--

import { test, Locator, Page } from '@playwright/test'

export default class DispensaryPage {

  readonly page: Page;

  private maxRetries = 3;

  private timeoutDuration = 5000;

  public dispensary: {

    dispensaryLink: Locator;

    activateCounter: Locator;

    counterSelection: Locator;

    counterName: Locator;

    activatedCounterInfo: Locator;

    deactivateCounterButton: Locator;

    titleName: Locator;

    name: Locator;

    prescription: Locator;

    reports: Locator;

    fromDate: Locator;

    showReportButton: Locator;

    userCollectionReport: Locator;

    counterDropdown: Locator;

    counterNameFromTable: Locator;

    rightPointerIcon: string;

    tooltipText: Locator;

  };

  constructor(page: Page) {

    this.page = page;

    this.dispensary = {

      dispensaryLink: page.getByRole('link', { name: 'Dispensary ' }),

      activateCounter: page.locator(""),

      counterSelection: page.locator(''),

      counterName: page.locator(''),

      activatedCounterInfo: page.locator(``),

      deactivateCounterButton: page.locator(``),

      titleName: page.locator(''),

      name: page.locator(''),

      prescription: page.locator(""),

      reports: page.getByRole('link', { name: 'Reports', exact: true }),

      fromDate: page.locator('#date'),

      showReportButton: page.getByRole('button', { name: 'Show Report' }),

      userCollectionReport: page.getByRole('link', { name: ' User Collection Report' }),

      counterDropdown: page.locator(`#ddlCounter`),

      counterNameFromTable: page.locator(``),

      rightPointerIcon: '',

      tooltipText: page.locator(''),

    };

  }

  /\*\*

  \* @Test1 Verify tooltip text when hovering over the dispensary pointer icon.

  \*

  \* @returns {Promise<string>} - Returns the trimmed tooltip text; throws an error if the tooltip text does not match the expected value.

  \*

  \* Steps:

  \* 1. Click on the Dispensary link to open the dispensary section.

  \* 2. Hover over the right-pointing icon to trigger the tooltip.

  \* 3. Capture the tooltip text displayed.

  \* 4. Verify that the tooltip text matches the expected message:

  \*    "You are currently in Main Dispensary dispensary. To change, you can always click here."

  \* 5. Return the actual tooltip text.

  \*/

  async verifyAndReturnDispensaryTooltipText(): Promise<string> {

    const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.dispensary.dispensaryLink.click();

    await this.dispensary.dispensaryLink.click();

    await this.page.locator('a').filter({ hasText: 'Main Dispensarydispensary' }).click();

    await this.page.locator("//label[text()=' Active Dispensary : ']/../i").hover();

    const text = await this.page.getByText('You are currently in Main Dispensary dispensary. To change, you can always click here.').innerText();

    return text;

  }

}

LaboratoryPage:--

import { test, Locator, Page } from '@playwright/test'

export default class LaboratoryPage {

  private page: Page;

  private laboratoryLink: Locator;

  private laboratoryDashboard: Locator;

  private sampleCollectionTab: Locator;

  private from\_date: Locator;

  private ok\_button: Locator;

  private col\_requestingDept: Locator;

  private requestdept\_hamburger: Locator;

  private select\_dropdown: Locator;

  private start\_with\_option: Locator;

  private req\_dept\_search: Locator;

  constructor(page: Page) {

    this.page = page;

    this.laboratoryLink = page.getByRole('link', { name: 'Laboratory ' });

    this.laboratoryDashboard = page.locator('');

    this.sampleCollectionTab = page.locator('//a[@href="#/Lab/Requisition"]').nth(1);

    this.from\_date = page.locator("#date").nth(0);

    this.ok\_button = page.locator('//button[text()=" OK "]');

    this.col\_requestingDept = page.locator('');

    this.requestdept\_hamburger = page.locator("//span[text()='Requisition Date']/../../span");

    this.select\_dropdown = page.locator("#filterType");

    this.start\_with\_option = page.locator("");

    this.req\_dept\_search = page.locator("#filterText");

  }

  /\*\*

   \* @Test9 Verify table filtering for "Male Ward"

   \*

   \* 1. Navigate to https://healthapp.yaksha.com/Home/Index#/Lab/Dashboard

   \* 2. Select the Sample Collections tab.

   \* 3. Enter From Date as 01-01-2020 and click OK.

   \* 4. Hover over the Requesting Department column and click Hamburger Menu.

   \* 5. Select Starts with from the dropdown.

   \* 6. Enter Male Ward in the text field.

   \*

   \*

   \*/

  async verifyTableFiltering() {

    const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.laboratoryLink.click();

    await this.sampleCollectionTab.click();

    await this.from\_date.fill('2020-01-01');

    await this.ok\_button.click();

    await this.requestdept\_hamburger.hover();

    await this.requestdept\_hamburger.click();

    await this.select\_dropdown.selectOption({ index: 2 });

    await this.req\_dept\_search.fill('Male Ward')

  }

}

Maternity:--

import { test, Locator, Page } from '@playwright/test'

export default class MaternityPage {

  readonly page: Page;

  private maternity: Locator;

  private starIcon: Locator;

  constructor(page: Page) {

    this.page = page;

    this.maternity = page.locator('//a[@href="#/Maternity"]'),

      this.starIcon = page.getByTitle('Remember this Date');

  }

  /\*\*

   \* @Test7 Verify File Upload for a Past Patient Record

   \*

   \* This method performs the interaction required to retrieve the tooltip text from the star icon.

   \* It first waits for the page to fully load and clicks on the "maternity" section to ensure the relevant elements are visible.

   \* Then it hovers over the star icon to trigger the tooltip and waits briefly to allow the tooltip to render.

   \* Finally, it captures and returns the value of the 'title' attribute, which holds the tooltip text.

   \*/

  async getTooltipTextFromStar(): Promise<string | null> {

    return "";

  }

}

NursingPage:--

import { test, Locator, Page } from '@playwright/test'

export default class NursingPage {

  readonly page: Page;

  private nursing\_tab: Locator;

  private past\_days: Locator;

  private from\_date: Locator;

  private ok\_button: Locator;

  private search\_field: Locator;

  private overview\_button: Locator;

  private upload\_button: Locator;

  private dept\_dropdown: Locator;

  private upload\_file: Locator;

  private submit\_button: Locator;

  private checkbox: Locator;

  private conclude: Locator;

  private saveButton: Locator;

  downloadPath: string | undefined;

  constructor(page: Page) {

    this.page = page;

    this.nursing\_tab = page.locator('//a[@href="#/Nursing"]');

    this.past\_days = page.locator('//a[text()="Past Days"]');

    this.from\_date = page.locator('#date').first();

    this.ok\_button = page.locator('//button[text()=" OK "]');

    this.search\_field = page.locator('#quickFilterInput');

    this.overview\_button = page.getByTitle('overview').first();

    this.upload\_button = page.locator('//input[@type="file"]');

    this.dept\_dropdown = page.locator("//select[@formcontrolname='FileType']");

    this.upload\_file = page.getByTitle('upload files').first();

    this.submit\_button = page.locator('//input[@value="Submit"]');

    this.checkbox = page.locator("");

    this.conclude = page.locator("");

    this.saveButton = page.locator("");

  }

  /\*\*

   \* @Test2 Verify Navigation to Patient Overview from Past Days Records

   \*

   \* @param {Record<string, string>} data - Not used in this method but typically used to pass additional parameters if needed.

   \* @returns {Promise<void>} - Returns void; logs error if any step fails.

   \*

   \* Steps:

   \* 1. Navigate to https://healthapp.yaksha.com/Home/Index#/Nursing/OutPatient

   \* 2. Click on the Past Days tab.

   \* 3. Enter the From Date as 01-01-2020 and click the OK button.

   \* 4. Search for the patient "Deepika Rani" in the list.

   \* 5. Locate the patient’s record and click on Overview from the Actions column.

   \*/

  async verifyPatientOverviewFromPastDaysRecords(): Promise<void> {

    const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.nursing\_tab.click();

    await this.past\_days.click();

    await this.from\_date.fill("2020-01-01");

    await this.ok\_button.click();

    await this.search\_field.click();

    await this.page.keyboard.insertText('Deep')

    await this.overview\_button.click();

  }

  /\*\*

   \* @Test3 Verify File Upload for a Past Patient Record

   \*

   \* @param {Record<string, string>} data - Not used in this method but typically used to pass additional parameters if needed.

   \* @returns {Promise<void>} - Returns void; logs error if any step fails.

   \*

   \* Steps:

   \* 1. Navigate to https://healthapp.yaksha.com/Home/Index#/Nursing/OutPatient

   \* 2. Click on the Past Days tab.

   \* 3. Enter the From Date as 01-01-2020 and click the OK button.

   \* 4. Search for the patient "Deepika Rani" in the list.

   \* 5. Locate the patient’s record and click on "Upload files" from the Actions column.

   \* 6. Verify that the Upload Files modal opens.

   \* 7. Select the Department as "Pathology".

   \* 8. Upload an image file.

   \* 9. Click on the Submit button.

   \*/

  async verifyfileupload() {

const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.nursing\_tab.click();

    await this.page.waitForTimeout(2000)

    await this.past\_days.click();

    await this.from\_date.fill("2020-01-01");

    await this.ok\_button.click();

    await this.search\_field.click();

    await this.page.keyboard.insertText('Deep')

    await this.page.keyboard.press('Enter')

    await this.upload\_file.click();

    await this.upload\_button.setInputFiles('inventoryRequisition.png');

    await this.dept\_dropdown.selectOption({ value: 'Pathology' });

    await this.upload\_button.click();

    await this.submit\_button.click();

  }

}

ParmacyPage:--

import { test, Locator, Page } from '@playwright/test'

export default class PharmacyPage {

  readonly page: Page;

  private pharmacy: Locator;

  private order: Locator;

  private export: Locator;

  downloadPath: string | undefined;

  constructor(page: Page) {

    this.page = page;

    this.pharmacy = page.getByRole('link', { name: 'Pharmacy ' });

    this.order = page.locator(`//a[@href="#/Pharmacy/Order"]`).nth(1);

    this.export = page.getByTitle(`Export To Excel`);

  }

  /\*\*

   \* This method verifies the functionality of exporting the order section data. It first waits for a brief timeout

   \* to ensure that the page elements are fully loaded. Then, it navigates through the pharmacy section and selects

   \* the order option once it's visible. After that, the method waits for the file download to begin by listening for

   \* the 'download' event. Once the export button is clicked, it waits for the download to complete and saves the

   \* file to the specified download directory. The file path of the downloaded file is stored in the `downloadPath`

   \* variable for further use or validation. This method helps in validating that the export functionality works as

   \* expected and the file is successfully downloaded.

   \*/

  /\*\*

   \* @Test10 Verify to export the order section data

   \*

   \* 1. Navigate to https://healthapp.yaksha.com/Home/Index#/Pharmacy/Dashboard

   \* 2. Click on order section

   \* 3. Click on "Export" button

   \*

   \*/

  async verifyExportOrderSectionData() {

    const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.pharmacy.click()

    await this.order.click()

    const [download] = await Promise.all([

      this.page.waitForEvent('download'),

      await this.export.click()

    ]);

    const path = await download.path();

    this.downloadPath = path;

    return path;

  }

}

ProcurementPage:--

import { test, Locator, Page } from '@playwright/test'

export default class ProcurementPage {

  readonly page: Page;

  private procurement: Locator;

  private quotations: Locator;

  private requestForQuotation: Locator;

  private subject: Locator;

  private description: Locator;

  private selectVendor: Locator;

  private dropdownOption: Locator;

  private itemName: Locator;

  private itemValue: Locator;

  private dropdownArrowButton: Locator;

  private quantity: Locator;

  private requestButton: Locator;

  private quotationGeneratedPopup: Locator;

  quotationMessageText: string | null = null;

  constructor(page: Page) {

    this.page = page;

    this.procurement = page.getByRole('link', { name: 'Procurement ' });

    this.quotations = page.getByRole('link', { name: 'Quotation' });

    this.requestForQuotation = page.getByRole('button', { name: 'Request For Quotation' });

    this.subject = page.getByRole('textbox', { name: 'Subject' });

    this.description = page.getByRole('textbox', { name: 'Description' });

    this.selectVendor = page.getByText('---Select Vendor---');

    this.dropdownOption = page.locator(``);

    this.itemName = page.locator(`#itemName0`);

    this.itemValue = page.locator(``);

    this.dropdownArrowButton = page.locator(``);

    this.quantity = page.locator(`#qtyip0`);

    this.requestButton = page.locator(`#RequestButton`);

    this.quotationGeneratedPopup = page.locator(`//p[text()='Request For Quotation is Generated and Saved']`);

  }

  /\*\*

   \* @Test8 Verify Request for Quotation Generation

   \*

   \*

   \* This method verifies the process of generating a Request For Quotation (RFQ) in the Procurement section.

   \* It starts by navigating to the Procurement section and selecting the Quotation option, followed by clicking

   \* on "Request For Quotation". The method then fills in the subject and description fields, selects a vendor from

   \* the dropdown, and ensures the dropdown is visible and clicked. Next, it waits for the item name and quantity fields

   \* to be visible, then fills in the item details (name and quantity). After that, it clicks the "Request" button to submit

   \* the quotation request. The method waits for the "Request For Quotation is Generated and Saved" popup to appear and

   \* retrieves its text content. The text content of the popup is then stored in the `quotationMessageText` class property

   \* for further validation or assertion. The method includes necessary waits to ensure that elements are visible and ready

   \* before interaction, helping ensure that the process completes smoothly.

   \*/

  async verifyRequestForQuotationGeneration() {

    const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.procurement.click()

    await this.quotations.click()

    await this.requestForQuotation.click()

    await this.subject.fill('test subject')

    await this.description.fill('test description')

    await this.selectVendor.click();

    await this.page.getByPlaceholder('Search').fill('Ashar & Company');

    await this.page.getByText('Ashar & Company').click();

    await this.itemName.fill('Soap');

    await this.page.keyboard.press('Enter');

    await this.quantity.fill('2');

    await this.requestButton.click();

    await this.quotationGeneratedPopup.innerText();

    this.quotationMessageText = await this.quotationGeneratedPopup.innerText();

  }

}

SettingsPage:---

import { test, Locator, Page } from '@playwright/test'

export class SettingsPage {

  readonly page: Page;

  private settingsLink: Locator;

  private more: Locator;

  private priceCategory: Locator;

  public disable: Locator;

  public activate: Locator;

  constructor(page: Page) {

    this.page = page;

    this.settingsLink = page.getByRole('link', { name: 'Settings ' });

    this.more = page.locator('//a[text()=" More... "]');

    this.disable = page.locator('//a[@danphe-grid-action="deactivatePriceCategorySetting"]').nth(0);

    this.activate = page.locator('//a[@danphe-grid-action="activatePriceCategorySetting"]').first();

    this.priceCategory = page.locator('//a[@href="#/Settings/PriceCategory"]').nth(1);

  }

  /\*\*

   \* @Test12 This method automates the process of enable/disable price category in more section of the Settings module.

   \*

   \* @description This function performs the following actions:

   \* 1. Navigate to “Settings” module.

   \* 2. Click on more... and select "Price Category" tab.

   \* 3. Click on “Disable” button to disable any Code in the table.

   \* 4. Verify a success message appears with the message "Deactivated.".

   \* 5. Activate the same code by clicking “Activate” button and verify the success message as

   \*/

  async verifyDisablePriceCategory() {

    const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.settingsLink.click();

    await this.more.click();

    await this.priceCategory.click();

    await this.disable.click();

  }

  /\*\*

 \* @Test12 Verify enabling the Price Category option.

 \*

 \* @returns {Promise<void>} - Returns void; waits for page load after interaction.

 \*

 \* Steps:

 \* 1. Highlight the "Activate" button for visual confirmation.

 \* 2. Click on the "Activate" button to enable the Price Category.

 \* 3. Wait for the page to complete loading.

 \* 4. Pause execution for 2 seconds to ensure the action is fully processed.

 \*/

  async verifyEnablePriceCategory() {

  }

}

UtilitiesPage:--

import { test, Locator, Page } from '@playwright/test'

export default class UtilitiesPage {

  readonly page: Page;

  public utilities: {

    utilitiesModule: Locator;

    ChangeBillingCounter: Locator;

    counters: Locator;

    counterItem: Locator;

    schemeRefund: Locator;

    newSchemeRefundEntry: Locator;

    saveButton: Locator;

    warningPopup: Locator;

  };

  constructor(page: Page) {

    this.page = page;

    this.utilities = {

      utilitiesModule: page.getByRole('link', { name: 'Utilities ' }),

      ChangeBillingCounter: page.locator(''),

      counters: page.locator(""),

      counterItem: page.getByText("New-1 "),

      schemeRefund: page.locator('//a[text()=" Scheme Refund "]'),

      newSchemeRefundEntry: page.locator("//a[text()=' New Scheme Refund Entry']"),

      saveButton: page.locator('#savebutton'),

      warningPopup: page.locator('//p[text()="Please fill all the mandatory fields."]'),

    };

  }

  /\*\*

   \* This method verifies that the appropriate warning popup is displayed

   \* when attempting to save a "Scheme Refund Entry" without filling in

   \* the mandatory fields. It navigates to the Utilities module, selects

   \* the "Scheme Refund" section, clicks on a counter item, and proceeds

   \* to the "New Scheme Refund Entry" form. Without entering any data,

   \* it clicks the save button to trigger and validate the warning popup.

   \*/

  /\*\*

   \*

   \* @Test11 Verify Warning Popup for Mandatory Fields in Scheme Refund

   \*

   \*

   \* 1. Navigate to Utilities module and select "Scheme Refund" tab.

   \* 2. If required, please select any counter value and then select “Scheme Refund” tab.

   \* 3. Click on "New scheme Refund Entry" button.

   \* 4. Now click on save without entering value in any field.

   \*/

  async verifyWarningPopupForMandatoryFiels() {

    const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.utilities.utilitiesModule.click();

    await this.utilities.schemeRefund.click();

    await this.utilities.counterItem.click();

    await this.utilities.schemeRefund.click();

    await this.utilities.newSchemeRefundEntry.click();

    await this.utilities.saveButton.click();

  }

}

SubstorePage:--

import { test, Locator, Page } from '@playwright/test'

export default class SubstorePage {

  readonly page: Page;

  private subStoreLink: Locator;

  private wardSupply: Locator;

  private accounts: Locator;

  private pharmacy: Locator;

  private inventory: Locator;

  private inventoryRequisition: Locator;

  private consumption: Locator;

  private reports: Locator;

  private patientConsumption: Locator;

  private return: Locator;

  private stock: Locator;

  constructor(page: Page) {

    this.page = page;

    this.subStoreLink = page.locator('//span[text()="SubStore"]');

    this.wardSupply = page.locator('');

    this.accounts = page.locator('//i[text()="Accounts"]/..');

    this.pharmacy = page.locator('//a[text()=" Pharmacy "]/..');

    this.inventory = page.locator('//a[text()=" Inventory "]/..');

    this.inventoryRequisition = page.locator('//a[text()="Inventory Requisition"]/..');

    this.consumption = page.locator('');

    this.reports = page.locator('');

    this.patientConsumption = page.locator('');

    this.return = page.locator('');

    this.stock = page.locator('');

  }

  // Getter methods to access private properties

  public getPharmacy() {

    return this.page.locator('//a[text()=" Pharmacy "]/..');

  }

  public getInventory() {

    return this.page.locator('//a[text()=" Inventory "]/..');

  }

  public getAccounts() {

  }

  /\*\*

   \* @Test6 Verify all sub-modules are displayed correctly after Clicking on the "SubStore " Module.

   \*

   \* @param {Record<string, string>} data - Not used in this method but typically used to pass additional parameters if needed.

   \* @returns {Promise<void>} - Returns void; logs error if any step fails.

   \* This method verifies the visibility and interaction with the sub-modules in the Ward Supply section.

   \* It starts by waiting for a brief timeout (2 seconds) to ensure that the page elements are fully loaded.

   \* Then, it clicks on the 'Ward Supply' module to display its sub-modules.

   \* Once the sub-modules are visible, it waits for the 'Accounts' sub-module to appear on the page, ensuring it

   \*  is ready for interaction. Finally, it clicks on the 'Accounts' sub-module to verify that it can be selected

   \* and interacted with, ensuring the functionality of the Ward Supply section.

   \*/

  async verifySubModulesDisplay() {

    const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.subStoreLink.click();

    await this.accounts.click();

    await this.inventory.click();

    await this.pharmacy.click();

  }

  /\*\*

   \* @Test13 Verifies navigation between different tabs within the "Inventory" tab in the Substore module.

   \*

   \*

   \* Steps:

   \* 1. Navigate to the “Substore” module.

   \* 2. Click on the “Accounts” button.

   \* 3. Select the “Inventory” tab.

   \* 4. Click and navigate between different tabs within the "Inventory" tab, including:

   \*    - “Stock”

   \*    - “Inventory Requisition”

   \*    - “Consumption”

   \*    - “Reports”

   \*    - “Patient Consumption”

   \*    - “Return”

   \*

   \* Expected Result:

   \* - Ensure successful navigation to each sub-tab of the "Inventory" tab.

   \*/

  async verifyNavigationToSubStoreModule() {

    await this.subStoreLink.click();

    await this.accounts.click();

    await this.inventory.click();

  }

  /\*\*

   \* @Test13 Navigate to the Accounts section.

   \*

   \* @returns {Promise<void>} - Returns void; ensures the Accounts section is loaded.

   \*

   \* Steps:

   \* 1. Highlight the "Accounts" link or button for visual confirmation.

   \* 2. Click on the "Accounts" element to initiate navigation.

   \* 3. Wait for the page to fully load after navigation.

   \* 4. Pause execution for 1 second to ensure the section is ready for interaction.

   \*/

  async navigateToAccounts() {

    await this.accounts.click();

    await this.page.waitForTimeout(2000)

  }

  /\*\*

   \* @Test13 Verify navigation to the Stock section.

   \*

   \* @returns {Promise<void>} - Returns void; ensures the Stock page is loaded.

   \*

   \* Steps:

   \* 1. Highlight the "Stock" element for visual confirmation.

   \* 2. Click on the "Stock" link or button to initiate navigation.

   \* 3. Wait for the page to fully load after the action.

   \* 4. Pause execution for 1 second to ensure the section is ready for interaction.

   \*/

  async verifyNavigationToStock() {

  }

  /\*\*

   \* @Test13 Verify navigation to the Inventory Requisition section.

   \*

   \* @returns {Promise<void>} - Returns void; ensures the Inventory Requisition page is loaded.

   \*

   \* Steps:

   \* 1. Highlight the "Inventory Requisition" element for visual confirmation.

   \* 2. Click on the "Inventory Requisition" link or button to initiate navigation.

   \* 3. Wait for the page to fully load after the action.

   \* 4. Pause execution for 1 second to ensure the section is ready for interaction.

   \*/

  async verifyNavigationToInventoryRequisition() {

  }

  /\*\*

   \* @Test13 Verify navigation to the Consumptions section.

   \*

   \* @returns {Promise<void>} - Returns void; ensures the Consumptions page is loaded.

   \*

   \* Steps:

   \* 1. Highlight the "Consumptions" element for visual confirmation.

   \* 2. Click on the "Consumptions" link or button to initiate navigation.

   \* 3. Wait for the page to fully load after the action.

   \* 4. Pause execution for 1 second to ensure the section is ready for interaction.

   \*/

  async verifyNavigationToConsumptions() {

  }

  /\*\*

   \* @Test13 Verify navigation to the Reports section.

   \*

   \* @returns {Promise<void>} - Returns void; ensures the Reports page is loaded.

   \*

   \* Steps:

   \* 1. Highlight the "Reports" element for visual confirmation.

   \* 2. Click on the "Reports" link or button to initiate navigation.

   \* 3. Wait for the page to fully load after the action.

   \* 4. Pause execution for 1 second to ensure the section is ready for interaction.

   \*/

  async verifyNavigationToReports() {

  }

  /\*\*

   \* @Test13 Verify navigation to the Patient Consumptions section.

   \*

   \* @returns {Promise<void>} - Returns void; ensures the Patient Consumptions page is loaded.

   \*

   \* Steps:

   \* 1. Highlight the "Patient Consumptions" element for visual confirmation.

   \* 2. Click on the "Patient Consumptions" link or button to initiate navigation.

   \* 3. Wait for the page to fully load after the action.

   \* 4. Pause execution for 1 second to ensure the section is ready for interaction.

   \*/

  async verifyNavigationToPatientConsumptions() {

  }

  /\*\*

   \* @Test Verify navigation to the Return section.

   \*

   \* @returns {Promise<void>} - Returns void; ensures the Return page is loaded.

   \*

   \* Steps:

   \* 1. Highlight the "Return" element for visual confirmation.

   \* 2. Click on the "Return" link or button to initiate navigation.

   \* 3. Wait for the page to fully load after the action.

   \* 4. Pause execution for 1 second to ensure the section is ready for interaction.

   \*/

  async verifyNavigationToReturn() {

  }

  /\*\*

   \* @Test14 Captures a screenshot of the Inventory Requisition section within the Substore module.

   \* @Return The screenshot of the Inventory Requisition section.

   \*

   \*

   \* Steps:

   \* 1. Navigate to the “Substore” module.

   \* 2. Click on the “Accounts” button.

   \* 3. Select the “Inventory” tab and then the “Inventory Requisition” sub-tab.

   \* 4. Capture a screenshot of the page and save at the default location.

   \*

   \* Expected Result:

   \* - Screenshot of the page is captured and saved successfully in the specified folder.

   \*/

  async captureScreenshotOfInventoryRequisitionSection() {

    const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.subStoreLink.click();

    await this.accounts.click();

    await this.inventory.click()

    await this.inventoryRequisition.click();

    const folder = 'screenshots';

    const filepath = `${folder}/${'inventory-requisition-section-chromium-win32-chromium-win32.png'}`;

    const buffer = await this.page.screenshot({ path: filepath });

    return buffer;

  }

}

AccountingPage:--

import { test, Locator, Page } from '@playwright/test'

export default class AccountingPage {

  readonly page: Page;

  public accounting: {

    accountingLink: Locator;

    reports: Locator;

    dailyTransaction: Locator;

    fiscalYear: Locator;

    load: Locator;

    settings: Locator;

    searchBar: Locator;

    activate: Locator;

    deactivate: Locator;

  };

  constructor(page: Page) {

    this.page = page;

    this.accounting = {

      accountingLink: page.getByRole('link', { name: 'Accounting ' }),

      reports: page.locator('//a[text()=" Reports "]'),

      dailyTransaction: page.locator('//i[text()="Daily Transaction"]'),

      fiscalYear: page.locator('//select[@class="ng-untouched ng-pristine ng-valid"]'),

      load: page.locator('//button[@type="button"]'),

      settings: page.locator('(//a[@href="#/Accounting/Settings"])[2]'),

      searchBar: page.locator('#quickFilterInput'),

      activate: page.locator('//a[text()="Activate"]').first(),

      deactivate: page.locator("//a[text()='Deactivate']"),

    };

  }

  /\*\*

   \* @Test4 Verifies the activation process of the ledger by navigating to the accounting settings, searching for a specific ledger,

   \* and activating it through a confirmation dialog.

   \*

   \* @param {Record<string, string>} data - Not used in this method but typically used to pass additional parameters if needed.

   \* @returns {Promise<void>} - Returns void; logs error if any step fails.

   \*

   \* Steps:

   \* 1. Navigate to the Accounting module.

   \* 2. Go to Settings and search for the "BANK A/C #" ledger.

   \* 3. Trigger the activation process, confirm the activation in the dialog, and finalize the action.

   \*/

  async verifyActivationLedger() {

    const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.accounting.accountingLink.click();

    await this.page.waitForTimeout(2000)

    await this.accounting.settings.click();

    await this.page.waitForTimeout(4000)

    await this.accounting.settings.click();

    const ActiveAccount = await this.page.locator('(//a[text()="Activate"]/../../../div)[1]').innerText();

    await this.accounting.searchBar.fill(ActiveAccount);

    await this.accounting.searchBar.click();

    await this.page.keyboard.press('Enter')

    this.page.once('dialog', async dialog => {

      await dialog.accept();

    })

    await this.accounting.activate.first().click();

  }

  /\*\*

   \* @Test5 Verifies the deactivation process of a ledger entry ("Sundry Debtors (Receivables)")

   \* in the Accounting module settings.

   \*

   \* Steps:

   \* 1. Navigates to the Accounting module and opens Settings.

   \* 2. Searches for a specific ledger using the search bar.

   \* 3. Handles the deactivation confirmation dialog.

   \* 4. Clicks on the deactivate button to trigger the action.

   \*

   \* @returns {Promise<void>} - This method performs UI actions and does not return a value.

   \*/

  async verifyDeactivationLedger() {

    const counter = await this.page.locator("//a[text()='X']");

    if (counter) {

      await counter.click();

    }

    await this.accounting.accountingLink.click();

    await this.page.waitForTimeout(2000)

    await this.accounting.settings.click();

    await this.page.waitForTimeout(4000)

    await this.accounting.settings.click();

    const DeActiveAccount = await this.page.locator('(//a[text()="Deactivate"]/../../../div)[1]').innerText();

    await this.accounting.searchBar.fill(DeActiveAccount);

    await this.accounting.searchBar.click();

    await this.page.keyboard.press('Enter')

    this.page.once('dialog', async dialog => {

      await dialog.accept();

    })

    await this.accounting.deactivate.first().click();

  }

}