CISC 327 Group 34 - CH

Assignment 4

November 11, 2024

Hayden Jenkins 20344883
Ryan Van Drunen 20331633
Madison MacNeil 20285877

Github Username – Name | *Student Number*

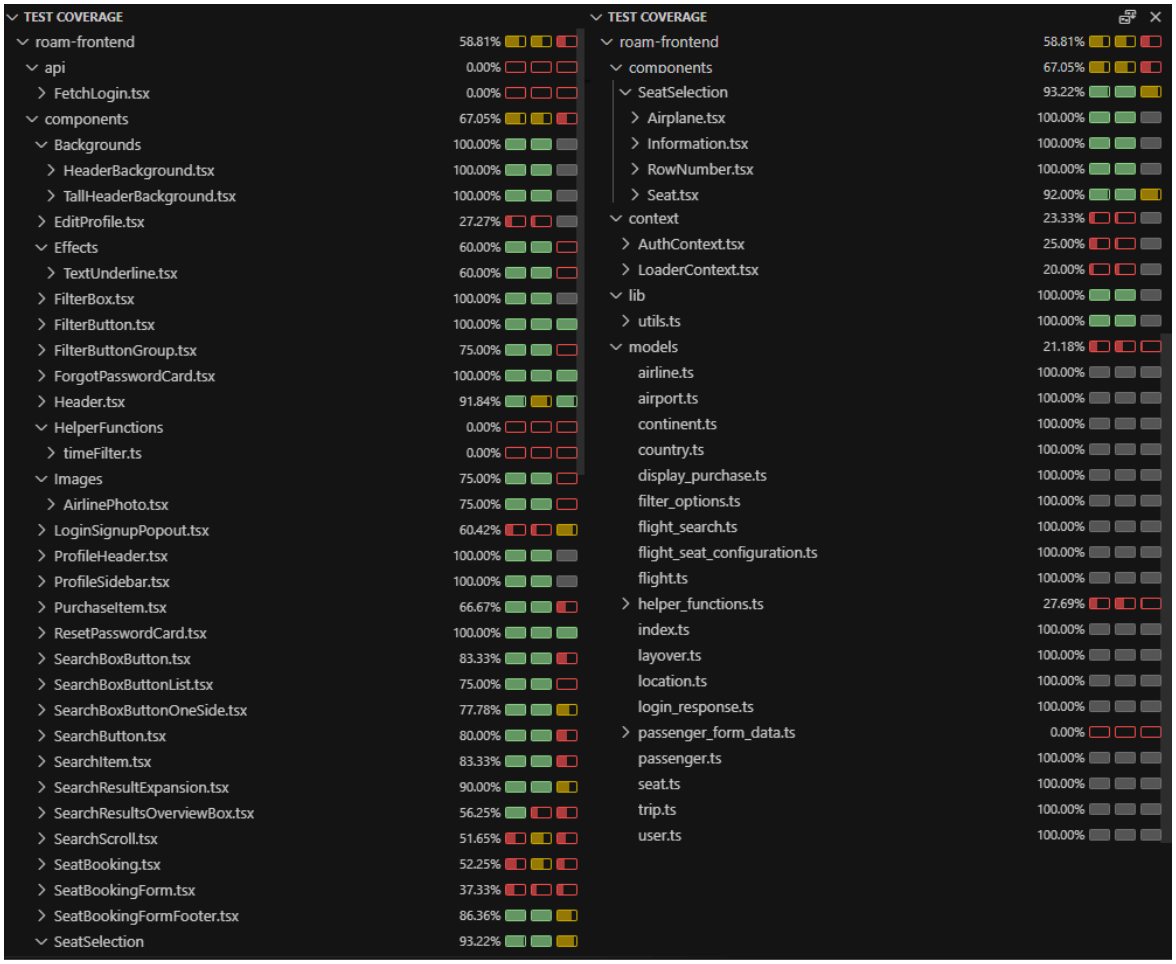
ryanvandrunen - Ryan Van Drunen | 20331633
hjenkins04/xonixdev - Hayden Jenkins | 20344883
madisonmacneil - Madison MacNeil | *20285877*

Task Distribution Table:

Component Name	Former Coverage	Completed By
TrendingLocationHomeGrid	89	Hayden
Edit Profile	27.27	Ryan
Effects/TextUnderline	60.0	Maddie
Filter Button Group	75.0	Maddie
Footer	71	Hayden
Header	91.84	Ryan
Helper Functions/timeFilter	0	Ryan
Images/AirlinePhoto	0	Ryan
LoginSignupPopup	60.42	Ryan
PurchaseItem	66.67	Ryan
SearchBox	33.55	Maddie
SearchBoxButtonSkeleton	80.00	Hayden
SearchBoxSkeletonLoader	42.86	Hayden
SearchButtonBox	83.33	Maddie
SearchButtonBoxList	75	Maddie

SearchButtonBoxOneSide	77.78	Maddie
SearchButton	80.0	Maddie
SearchItem	83	Maddie
SearchResultsExpansion	90	Maddie
SearchResultsOverviewBox	56.25	Maddie
SeachScroll	51.65	Maddie
SeatBooking	52.25	Hayden
SeatBookingForm	37.33	Hayden
SeatBookingFormFooter	86.36	Hayden
SeatSelection	93.22	Hayden
AuthContext	25.00	Hayden
LoaderContext	25.0	Hayden
DestinationContext	35.0	Hayden
SearchContext	37.50	Hayden
models/helper_functions	27.69	Ryan
Passenger_form_data	0	Ryan
HelperFunctions/setTripContextData	74.36	Ryan

Initial Test Coverage Report



Final Test Coverage Report

Component Test Coverage Report

TrendingLocationsHomeGrid:

Original Coverage: 89%

Trending Locations Home Grid didn't have any tests handling the case of only showing 4 destinations on small screens and displaying a fallback image when no image url were passed. The initial test suite was also missing tests for cases where image_path was undefined, requiring a fallback image and error handling in the fetchPopDestinations when the API call failed. We introduced tests to handle these 4 issues verifying that a placeholder image would be displayed if no image were passed and also mocked fetchPopDestionations to throw an error and confirm the error handling functionality. By creating tests and passing isSmallScreen as true, we were able to check that the 5th destination had the hidden attribute in its css attributes. And finally by mocking a destination to not pass an image url we were also able to force the fallback image condition.

4 destinations on small screens

```
{destinations.map((destination, index) => (
  <li
    key={destination.guid}
    className={`flex flex-col items-center ${
      index === 4 && (isSmallScreen === true || isSmallScreen === undefined)
        ? "hidden lg:flex"
        : ""
    }`}
    data-testid={`destination-item-${index}`}
```

```
test("Shows 4 destinations on medium (md) screens", async () => {
  // Arrange: Rebder the component and set the screen width to small

  render(<TrendingLocationsHomeGrid destinations={mockDestinations} isSmallScreen={true

  // Assert: Check that only the first 4 destinations are visible
  mockDestinations.slice(0, 4).forEach((_, index) => {
    const destinationItem = screen.getByTestId(`destination-item-${index}`);
    expect(destinationItem).toBeVisible();
  });

  // Assert: Ensure the fifth destination is hidden on md and sm screens
  const hiddenDestination = screen.queryByTestId("destination-item-4");
  expect(hiddenDestination).toHaveClass("hidden");
});
```

Fallback placeholder image

```
<Image
  src={destination.image_path || "/assets/placeholder.svg"}
  alt={destination.name}
  fill
  style={{ objectFit: "cover" }}
  className="transition-transform duration-300 group-hover:scale-110"
  data-testid="destination-image"
/>
```

```
test("Displays placeholder image when image_path is missing", () => {
  renderComponent();

  // Assert: Check that the destination with a missing image path shows the placeholder
  const sydneyImage = screen.getByAltText("Sydney");
  expect(sydneyImage).toHaveAttribute("src", expect.stringContaining("placeholder.svg"));
});
```

Fallback image

```
src={destination.image_path || "/assets/placeholder.svg"}
```

```
test("renders placeholder image when image_path is missing", () => {
  const destinationsWithMissingImages: PopularDestination[] = [
    { guid: "1", name: "London", image_path: undefined },
  ];
  render(<TrendingLocationsHomeGrid destinations={destinationsWithMissingImages} />);

  // Assert placeholder image is displayed
  const image = screen.getByTestId("destination-image");
  expect(image).toHaveAttribute("src", "/assets/placeholder.svg");
});
```

refreshDestinations error catch

```
refreshDestinations: async () => {
  try {
    const destinations = await fetchPopDestinations();
    set({ popularDestinations: destinations });
  } catch (error) {
    console.error("Error fetching popular destinations:", error);
  }
},
```

```
test("refreshDestinations handles error when fetchPopDestinations throws", async () => {
  // Arrange: Mock fetchPopDestinations to throw an error
  const consoleErrorSpy = jest.spyOn(console, "error").mockImplementation();
  (fetchPopDestinations as jest.Mock).mockRejectedValue(new Error("Failed to fetch"));

  // Act: Render component and call refreshDestinations
  render(
    <TrendingLocationsHomeGrid destinations={useDestinationsStore.getState().popularDestinations} />
  );

  await act(async () => {
    await useDestinationsStore.getState().refreshDestinations();
  });

  // Assert: Check that an error occurred, and destinations is an empty array
  expect(useDestinationsStore.getState().popularDestinations).toEqual([]);
  expect(consoleErrorSpy).toHaveBeenCalledWith(
    "Error fetching popular destinations:",
    expect.any(Error)
  );

  // Cleanup
  consoleErrorSpy.mockRestore();
});
```

AuthContext:

Original Coverage: 25%

Was originally missing tests for SignIn SignOut functionality aswell as SignInPopup and BadAccessPopup. Use the HomePage component to test popup functionality and also test the state persistence for the sign-in and sign-out logic.

SignIn

```
// Function to sign in and set guid
signIn: (guid: string) =>
  set((state) => ({
    authData: { ...state.authData, isSignedIn: true, guid },
  })),
```

```
test("Sign in updates auth state", async () => {
  // Arrange: Sign in with a mock GUID
  const mockGuid = "test-guid";
  act(() => {
    useAuthStore.getState().signIn(mockGuid);
  });

  // Assert: Check that the auth state reflects the sign-in
  await waitFor(() => {
    const { authData } = useAuthStore.getState();
    expect(authData.guid).toBe(mockGuid);
    expect(authData.isSignedIn).toBe(true);
  });
});
```

SignOut

```
// Function to sign out and clear guid
signOut: () =>
  set((state) => ({
    authData: { ...state.authData, isSignedIn: false, guid: "" },
  })),
```

```
test("Sign out clears auth state", async () => {
  // Arrange: Sign in first, then sign out
  const mockGuid = "test-guid";
  act(() => {
    useAuthStore.getState().signIn(mockGuid);
  });
  act(() => {
    useAuthStore.getState().signOut();
  });

  // Assert: Check that the auth state reflects the sign-out
  await waitFor(() => {
    const { authData } = useAuthStore.getState();
    expect(authData.guid).toBe("");
    expect(authData.isSignedIn).toBe(false);
  });
});
```

SignInPopup

```
// Function to toggle showPleaseSignInPopup
setShowPleaseSignInPopup: (show: boolean) =>
  set((state) => ({
    authData: {
      ...state.authData,
      showPleaseSignInPopup: show,
    },
  })),
```

```
test("Display Please Sign In popup", async () => {
  // Arrange: Set the showPleaseSignInPopup flag to true
  act(() => {
    useAuthStore.getState().setShowPleaseSignInPopup(true);
  });

  // Act: Render a component to display the popup
  render(<HomePage />);

  // Assert: The popup should appear
  await waitFor(() => {
    const popup = screen.getByTestId("please-sign-in-popup");
    expect(popup).toBeInTheDocument();
  });
});
```

BadAccessPopup

```
// Function to toggle showBadAccessPopup
setBadAccessPopup: (show: boolean) =>
  set((state) => ({
    authData: {
      ...state.authData,
      showBadAccessPopup: show,
    },
  })),
```

```
test("Display Bad Access popup", async () => {
  // Arrange: Set the showBadAccessPopup flag to true
  act(() => {
    useAuthStore.getState().setBadAccessPopup(true);
  });

  // Act: Render a component to display the popup
  render(<HomePage />);

  // Assert: The popup should appear
  await waitFor(() => {
    const popup = screen.getByTestId("bad-access-popup");
    expect(popup).toBeInTheDocument();
  });
});
```

LoaderContext:

Original Coverage: 25%

Was missing tests for showLoader and hideloader functionality. Added two tests to ensure that the loader popup was displayed when the loader context isLoading was true and hidden otherwise.

showLoader

```
export const useLoaderStore = create<LoaderStore>((set) => ({
  isLoading: false,
  childrenHidden: false,

  showLoader: () => set({ isLoading: true }),
  hideLoader: () => set({ isLoading: false }),
  showChildren: () => set({ childrenHidden: false }),
  hideChildren: () => set({ childrenHidden: true }),
}));
```

```
test("Displays LoaderPopup when showLoader is called", async () => {
  // Arrange: Render component
  renderComponent();

  // Act: Trigger showLoader to update isLoading state
  act(() => {
    useLoaderStore.getState().showLoader();
  });

  // Assert: LoaderPopup should be visible
  await waitFor(() => {
    const loaderPopup = screen.getByTestId("loader-popup");
    expect(loaderPopup).toBeInTheDocument();
  });
});
```

```
test("Hides LoaderPopup when hideLoader is called", async () => {
  // Arrange: Render component and trigger showLoader to show LoaderPopup
  renderComponent();
  act(() => {
    useLoaderStore.getState().showLoader();
  });

  // Act: Trigger hideLoader to update isLoading state
  act(() => {
    useLoaderStore.getState().hideLoader();
  });

  // Assert: LoaderPopup should not be visible
  await waitFor(() => {
    const loaderPopup = screen.queryByTestId("loader-popup");
    expect(loaderPopup).not.toBeInTheDocument();
  });
});
```

SearchContext:

Original Coverage: 37.50%

There were no tests for setting the search data nor the persistence of data using partialize. Added tests to verify that setSearchData could update individual properties within searchData without changing other properties. Also created a test to ensure that when modifying searchData using setSearchData that searchData was correctly stored in local storage.

SearchContext

```
export const useSearchStore = create<SearchStore>()(
  persist(
    (set) => ({
      searchData: {
        departureAirport: null,
        arrivalAirport: null,
        departureDate: null,
        returnDate: null,
        passengers: 1,
        seatTypeMapping: { 1: "Business" },
        isRoundTrip: true,
        selectedAirlines: [],
      },
      setSearchData: (data: Partial<SearchData>) =>
        set((state) => ({
          searchData: { ...state.searchData, ...data },
        })),
    }),
    {
      name: "searchData-storage", // Key for local storage
      partialize: (state) => ({ searchData: state.searchData }), // Persist searchData
    }
  )
);
```

```
test("Persistence of searchData with partialize", () => {
  // Act: Modify search data
  act(() => {
    useSearchStore.getState().setSearchData({
      isRoundTrip: false,
      selectedAirlines: ["Airline1", "Airline2"],
    });
  });

  const storedData = JSON.parse(localStorage.getItem("searchData-storage") || "{}");

  // Assert: Only searchData is persisted as per partialize configuration
  expect(storedData).toHaveProperty("state.searchData");
  expect(storedData.state.searchData.isRoundTrip).toBe(false);
  expect(storedData.state.searchData.selectedAirlines).toEqual(["Airline1", "Airline2"]);
});
```

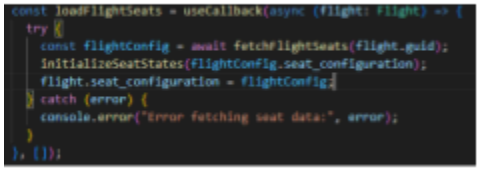
```
test("setSearchData updates searchData partially", () => {

  // Act: Update only departureAirport and passengers fields
  act(() => {
    useSearchStore.getState().setSearchData({
      departureAirport: mockAirport,
      passengers: 3,
    });
  });

  // Assert: Check that only specific fields were updated
  const { searchData } = useSearchStore.getState();
  expect(searchData.departureAirport).toEqual(mockAirport);
  expect(searchData.passengers).toBe(3);
  expect(searchData.arrivalAirport).toBe(null);
});
```

SeatBooking:

Original Coverage: 52.25%

The seat booking component was missing tests that ensured the fetchFlightSeats error catching and handling logic and that the cancel button redirect the user back to where they came from. By mocking the fetchFlightSeats API call to fail, we were able to test the catch block in the API call. By mocking the react-router and testing the cancel button click, we were able to ensure that the router back method was called and worked correctly, which originally had no code coverage.

Load Flight Seats

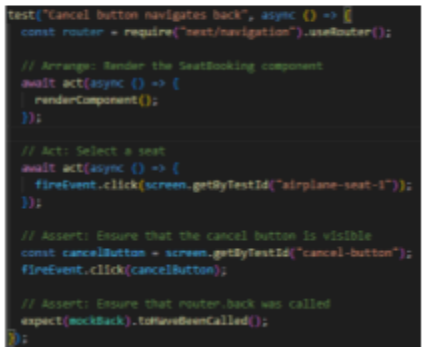


```
const loadFlightSeats = useCallback(async (flight: Flight) => {
  try {
    const flightConfig = await fetchFlightSeats(flight.guid);
    initializeSeatStates(flightConfig.seat_configuration);
    flight.seat_configuration = flightConfig;
  } catch (error) {
    console.error("Error fetching seat data:", error);
  }
}, []);
```

```
test("Error in fetchFlightSeats logs an error message", async () => {
  const consoleErrorSpy = jest.spyOn(console, "error").mockImplementation();
  (fetchFlightSeats as jest.Mock).mockRejectedValueOnce(new Error("Failed to fetch seats"));

  render(<SeatBooking />);

  await act(async () => {
    await fetchFlightSeats("mock-flight-id");
  });

  expect(console.error).toHaveBeenCalledWith("Error fetching seat data:", expect.any(Error));

  consoleErrorSpy.mockRestore();
});
```

Cancel Button Redirect

```
const router = useRouter();

const cancel = () => {
  router.back();
};
```



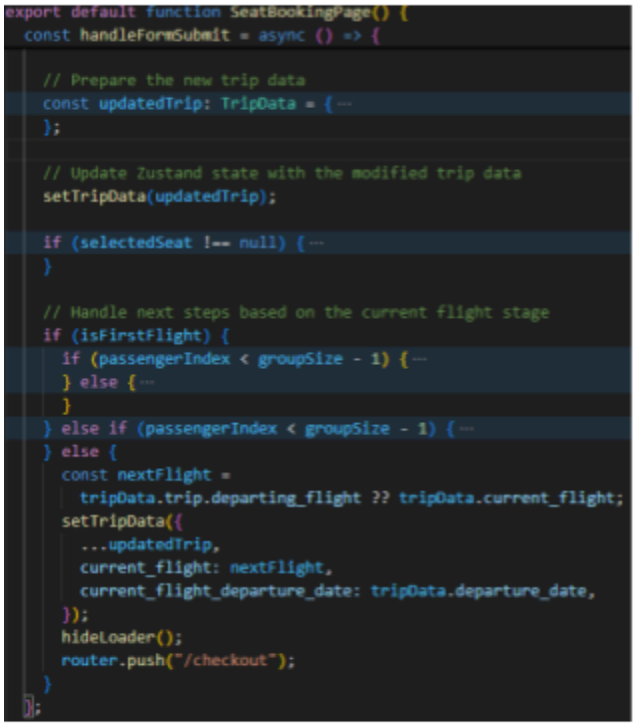
```
test("Cancel button navigates back", async () => {
  const router = require("next/navigation").useRouter();

  // Arrange: Render the SeatBooking component
  await act(async () => {
    renderComponent();
  });

  // Act: Select a seat
  await act(async () => {
    fireEvent.click(screen.getByTestId("airplane-seat-1"));
  });

  // Assert: Ensure that the cancel button is visible
  const cancelButton = screen.getByTestId("cancel-button");
  fireEvent.click(cancelButton);

  // Assert: Ensure that router.back was called
  expect(mockBack).toHaveBeenCalled();
});
```

SeatBookingForm / SeatBookingFormFooter:

Original Coverage: 37.33% / 86.36%

The seat booking form component was missing tests for the next passenger and book return flight within the seat booking form. This onFormSubmission function is what prompts the user to book a ticket for the other passengers, book the return flight or proceed to checkout. Both the next passenger and return flight conditional statements were not being executed as the original test case only considered one direct flight with one passenger. Therefore by adding a test case that mocked the tripDataStore with 2 passengers and a return flight we were able to test these conditional statements.

Seat Booking Form Submission



```
export default function SeatBookingPage() {
  const handleFormSubmit = async () => {

    // Prepare the new trip data
    const updatedTrip: TripData = {···
    };

    // Update Zustand state with the modified trip data
    setTripData(updatedTrip);

    if (selectedSeat !== null) {···
    }

    // Handle next steps based on the current flight stage
    if (isFirstFlight) {
      if (passengerIndex < groupSize - 1) {···
      } else {···
      }
    } else if (passengerIndex < groupSize - 1) {···
    } else {
      const nextFlight =
        tripData.trip.departing_flight ?? tripData.current_flight;
      setTripData({
        ...updatedTrip,
        current_flight: nextFlight,
        current_flight_departure_date: tripData.departure_date,
      });
      hideLoader();
      router.push("/checkout");
    }
  };
```

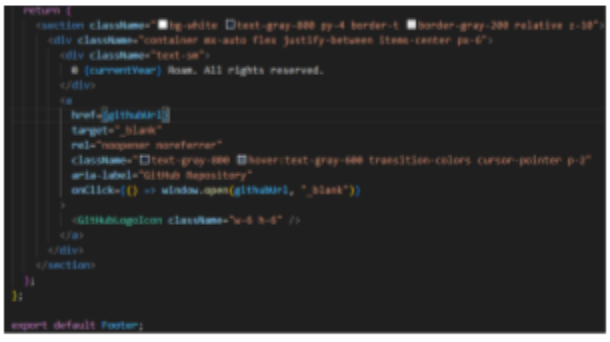
```
test("Executes full handleFormSubmit flow and reserves seat", async () => {
  await act(async () => {
    mockUseTripStore.getState().tripData = {
      ...mockUseTripStore.getState().tripData
    };
  });

  render(<SeatBooking />)

  // Departing Flight
  // First Passenger
  // Assert: Ensure that the seat is visible
  await waitFor(() => {
    expect(screen.getByTestId("airplane-seat-1")).toBeInTheDocument();
  });

  // Act: Select a seat
  await act(async () => {
    fireEvent.click(screen.getByTestId("airplane-seat-1"));
  });


  // Act: Submit form
  await act(async () => {
    fireEvent.click(screen.getByTestId("next-passenger-button"));
  });

  // Assert: Ensure seat is updated
  expect(mockUseTripStore.tripData.trip?.passengers[0].departing_seat_id).toBe(1);

  // Second Passenger
  // Assert: Ensure that the seat is visible
  await waitFor(() => {
    expect(screen.getByTestId("airplane-seat-2")).toBeInTheDocument();
  });

  // Act: Select a seat
  await act(async () => {
    fireEvent.click(screen.getByTestId("airplane-seat-2"));
  });

  // Act: Submit form
  await act(async () => {
    fireEvent.click(screen.getByTestId("book-return-flight-button"));
  });
```

Footer:

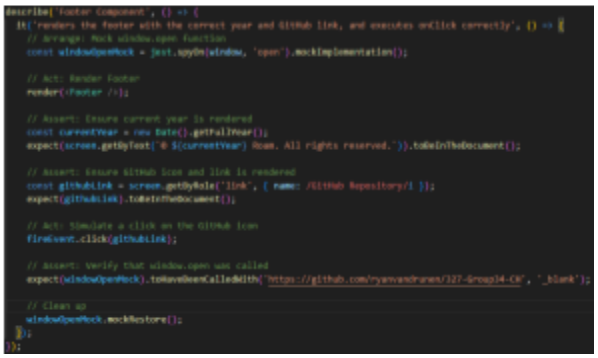
Original Coverage: 0%

The Footer component had no tests, therefore the onClick and Current year for the copyright text had no code coverage. By creating a simple test to ensure the component was rendered and displayed the correct date along with a simple button click check, we were able to achieve full code coverage.

Footer Component



```
  return (
    <section className="bg-white text-gray-800 py-4 border-t border-gray-200 relative z-10">
      <div className="container mx-auto flex justify-between items-center px-6">
        <div classNames="text-sm">
          © {currentYear} Roam. All rights reserved.
        </div>
        <a
          href={githubUrl}
          target="_blank"
          rel="noopener noreferrer"
          classNames="text-gray-600 hover:text-gray-600 transition-colors cursor-pointer p-2"
          aria-label="GitHub Repository"
          onClick={() => window.open(githubUrl, "_blank")}
        >
          <GitHubLogoIcon classNames="w-6 h-6" />
        </a>
      </div>
    </section>
  );
};

export default Footer;
```



```
describe('Footer Component', () => {
  it('renders the footer with the correct year and GitHub link, and executes onClick correctly', () => {
    // Arrange: Mock window.open function
    const windowOpenMock = jest.spyOn(window, 'open').mockImplementation();

    // Act: Render Footer
    render(<Footer />);

    // Assert: Ensure current year is rendered
    const currentYear = new Date().getFullYear();
    expect(screen.getByText(`© ${currentYear} Roam. All rights reserved.`)).toBeInTheDocument();

    // Assert: Ensure GitHub icon and link is rendered
    const githubLink = screen.getByRole('link', { name: /GitHub Repository/i });
    expect(githubLink).toBeInTheDocument();

    // Act: Simulate a click on the GitHub icon
    fireEvent.click(githubLink);

    // Assert: Verify that window.open was called
    expect(windowOpenMock).toHaveBeenCalledWith('https://github.com/ryanvandraven/327-Group14-CH', '_blank');

    // Clean up
    windowOpenMock.mockRestore();
  });
});
```

SearchBoxButtonSkeleton:

Original Coverage: 80%

The Search Box Button Skeleton was missing a test for the case when the size prop was not passed using the default "w-[230px]". Therefore by adding a test case that rendered the skeleton without an explicit size and checking that the "w-[230px]" class attribute was applied, we were able to add coverage to this behavior.

Search Box Button Skeleton Component

```
const SearchBoxButtonSkeleton: React.FC<SearchBoxButtonSkeletonProps> = ({ size = "w-[230px]", className = "", ...props }) => {
```

```
test('renders with default size and skeleton elements', () => {
  // Act: Render the component
  render(<SearchBoxButtonSkeleton />);

  // Assert: Verify the default size class is applied
  const container = screen.getByTestId("skeleton-button");
  expect(container).toHaveClass('w-[230px]');

  // Assert: Ensure skeleton is rendered with the correct sub attributes
  const headerSkeleton = screen.getByTestId('skeleton-header-text');
  expect(headerSkeleton).toBeInTheDocument();
  expect(headerSkeleton).toHaveClass('h-4 w-16 bg-gray-200');

  const iconSkeleton = screen.getByTestId('skeleton-icon');
  expect(iconSkeleton).toBeInTheDocument();
  expect(iconSkeleton).toHaveClass('h-4 w-4 bg-gray-300');

  const mainTextSkeleton = screen.getByTestId('skeleton-main-text');
  expect(mainTextSkeleton).toBeInTheDocument();
  expect(mainTextSkeleton).toHaveClass('h-5 w-20 bg-gray-200');

  const subTextSkeleton = screen.getByTestId('skeleton-sub-text');
  expect(subTextSkeleton).toBeInTheDocument();
  expect(subTextSkeleton).toHaveClass('h-4 w-16 bg-gray-200');
});
```

Hump Button:

Original Coverage: 0%

The Hump Button Component was missing a test for both the primary and secondary button click functions. By adding two tests that mocked this onClick functionality, we were able to increase our coverage testing to include these 2 missing function calls.

Hump Button Primary onClick

```
{/* Left Button Path (Primary) */}
<g clipPath="url(#clip0_20_24218)">
  <path
    d="M34.3072 32.293C25.3572 49.5865 8.94898 56.1745 -0.
    fill={isPrimaryActive ? secondaryColor : primaryColor}
    style={{ cursor: "pointer" }}
    onClick={handlePrimaryClick}
  />
</g>
```

```
test("calls onSecondaryClick when secondary button is clicked", () => {
  render(
    <HumpButton
      primaryColor="#FF0000"
      secondaryColor="#00FF00"
      primaryText="Primary"
      secondaryText="Secondary"
      onPrimaryClick={mockPrimaryClick}
      onSecondaryClick={mockSecondaryClick}
      isPrimaryActive={false}
    />
  );

  const secondaryButton = screen.getByText("Secondary");
  fireEvent.click(secondaryButton);
  expect(mockSecondaryClick).toHaveBeenCalled();
  expect(mockPrimaryClick).not.toHaveBeenCalled();
});
```

Hump Button Secondary onClick

```
{/* Right Button Path (Secondary) */}
<path
  d="M304.317 32.5949C295.367 50.1978 278.959 56.9037 270.009
  fill={!isPrimaryActive ? secondaryColor : primaryColor}
  style={{ cursor: "pointer" }}
  onClick={handleSecondaryClick}
/>
```

```
test("calls onPrimaryClick when primary button is clicked", () => {
  render(
    <HumpButton
      primaryColor="#FF0000"
      secondaryColor="#00FF00"
      primaryText="Primary"
      secondaryText="Secondary"
      onPrimaryClick={mockPrimaryClick}
      onSecondaryClick={mockSecondaryClick}
    />
  );

  const primaryButton = screen.getByText("Primary");
  fireEvent.click(primaryButton);
  expect(mockPrimaryClick).toHaveBeenCalled();
```

TextUnderline:

Original Coverage: 60.0%

The original test cases did not provide coverage of line 9 of the TextUnderline component, which defines the TextUnderline props, specifically; the default rendering height, width and class name values. Three test cases were developed to test the rendering of these prop values. The first test confirms that a default rendering of the component has height and width attributes of 330px and 34px. The second test confirms accurate size rendering when the component is passed custom height and width props. The third test confirms the components rendering when passed a custom class name.The addition of these tests brought our coverage of this file to 100%.

Line without coverage:

```
const TextUnderline: React.FC<TextUnderlineProps> = ({ width = 330, height = 34, className }) => (
```

Test 1:

```
test('Renders with default width and height', () => {
    render(<TextUnderline />);

    const underlineElement = screen.getByTestId('text-underline');
    expect(underlineElement).toBeInTheDocument();
    expect(underlineElement).toHaveAttribute('width', '330');
    expect(underlineElement).toHaveAttribute('height', '34');
});
```

Test 2:

```
test('Applies custom width and height props', () => {
    render(<TextUnderline width={200} height={20} />);

    const underlineElement = screen.getByTestId('text-underline');
    expect(underlineElement).toHaveAttribute('width', '200');
    expect(underlineElement).toHaveAttribute('height', '20');
});
```

Test 3:

```
test('Applies custom className', () => {
    const customClass = 'custom-underline';
    render(<TextUnderline className={customClass} />);

    const underlineElement = screen.getByTestId('text-underline');
    expect(underlineElement).toHaveClass(customClass);
});
```

FilterButtonGroup:

Original Coverage: 75.0%

The existing test cases for this component, a wrapper for the filter buttons, did not provide coverage that the children passed to the component were rendering as intended. As such, four test cases were developed to confirm that the children render as

expected, render according to a custom class name passed to the component, render correctly with the utility classes included in the props and with any additional props passed. The addition of these test cases brought our coverage of this component to 100%.

Code Without Coverage:

```
const FilterButtonGroup = ({ children, className = "", ...props }: FilterButtonListProps) => {
```

Tests 1-4 to resolve Coverage:

```
it('Renders children correctly', () => {
    render(
        <FilterButtonGroup>
            <div data-testid="child-element">Child</div>
        </FilterButtonGroup>
    );

    const childElement = screen.getByTestId('child-element');
    expect(childElement).toBeInTheDocument();
});

it('Applies the passed className', () => {
    render(
        <FilterButtonGroup className="custom-class">
            <div data-testid="child-element">Child</div>
        </FilterButtonGroup>
    );

    const groupElement = screen.getByTestId('filter-button-group');
    expect(groupElement).toHaveClass('custom-class');
});
```

```
it('Renders with default classes when no className is passed', () => {
    render(
        <FilterButtonGroup>
            <div data-testid="child-element">Child</div>
        </FilterButtonGroup>
    );

    const groupElement = screen.getByTestId('filter-button-group');
    expect(groupElement).toHaveClass('grid');
    expect(groupElement).toHaveClass('gap-4');
    expect(groupElement).toHaveClass('w-full');
    expect(groupElement).toHaveClass('justify-between');
});

it('Renders with additional props passed through', () => {
    render(
        <FilterButtonGroup data-testid="custom-group" role="group">
            <div data-testid="child-element">Child</div>
        </FilterButtonGroup>
    );

    const groupElement = screen.getByTestId('custom-group');
    expect(groupElement).toHaveAttribute('role', 'group');
});
```

SearchBox:

Original Coverage: 33.55%

The search box component, which is composed of a number of functionalities, was under-tested in our initial implementation. Nearly all functionalities of the component required a specific input test case to achieve test coverage above 90%. Some of the most notable areas without coverage were the event handlers of each search field input type - arrival Airport, departure airport, adding a passenger, removing a passenger, changing the class of one or more passengers. All such functionalities needed a test case. An example event handler on the change of an arrival airport is available below:

```
const handleDepartureChange = (value: Airport) => {
  setSearchData({
    ...searchData,
    departureAirport: value,
    arrivalAirport:
      searchData.arrivalAirport?.iata_code === value.iata_code
        ? null
        : searchData.arrivalAirport,
  });
};
```

This component contains a button toggle between a round trip search box UI and a one way search box UI. A test was needed to ensure the appropriate isRoundTrip boolean was applied to searchData when the button was toggled.

```
onPrimaryClick={() =>
  setSearchData({ ...searchData, isRoundTrip: true })
}
onSecondaryClick={() =>
  setSearchData({ ...searchData, isRoundTrip: false })
}
```

Resolution test for this coverage issue:

Another aspect of the component that required a series of test permutations was date selection from a calendar UI. Tests were developed to cover the cases of a new date selection, deselecting a date, for both arrival and departure dates etc. ensuring the searchData values were updated thereof.

```
  </PopoverTrigger>
  <PopoverContent className="w-auto p-0" align="start">
    <Calendar
      mode="single"
      selected={searchData.departureDate || undefined}
      onSelect={(date) =>
        setSearchData({
          ...searchData,
          departureDate: date || null,
        })
      }
    />
  </PopoverContent>
</Popover>
```

Example Test Case: of which many permutations were required to get coverage >90%

```
it('Remove Passengers', async () => {
    const setSearchDataMock = jest.fn();

    const useSearchStoreMock = useSearchStore as unknown as jest.Mock<typeof useSearchStore>;
    (useSearchStoreMock as jest.Mock).mockReturnValueOnce({
        searchData: {
            departureAirport: mockDepartureAirport,
            arrivalAirport: mockArrivalAirport,
            departureDate: '2024-12-01',
            returnDate: null,
            seatTypeMapping: {},
            passengers: 2,
            isRoundTrip: true,
            selectedAirlines: [],
        },
        setSearchData: setSearchDataMock,
    });

    render(
        <SearchBox
            airports={[mockDepartureAirport, mockArrivalAirport]}
            showRequiredFieldPopup={showRequiredFieldPopupMock}
        />
    );

    const passengerButton = screen.getByText('Business');
    fireEvent.click(passengerButton);

    const removePassengerButton = screen.getByTestId('remove-passenger')
    fireEvent.click(removePassengerButton)

    expect(setSearchDataMock).toHaveBeenCalledWith(expect.objectContaining({
        passengers: 1,
        returnDate: null,
    }));
});
```

The last aspect of this component that was missing coverage was rendering the appropriate pop-up when a search field contained a null value and the user attempted to proceed.

```
const handleButtonClick = () => {
  // Check if all required fields are selected
  if (!searchData.departureAirport) {
    showRequiredFieldPopup("Departure City");
  } else if (!searchData.arrivalAirport) {
    showRequiredFieldPopup("Arrival City");
  } else if (!searchData.departureDate) {
    showRequiredFieldPopup("Departure Date");
  } else if (searchData.isRoundTrip && !searchData.returnDate) {
    showRequiredFieldPopup("Return Date");
  } else if (!searchData.passengers || searchData.passengers < 1) {
    showRequiredFieldPopup("Passengers");
  } else {
    router.push("/search-results");
  }
};
```

Example Test Case: of which many permutations were required to get coverage >90%

```
it('should call showRequiredFieldPopup for null Passengers', async () => {

    const setSearchDataMock = jest.fn();
    const useSearchStoreMock = useSearchStore as unknown as jest.Mock<typeof useSearchStore>;
    (useSearchStoreMock as jest.Mock).mockReturnValueOnce({
        searchData: {
            departureAirport: mockDepartureAirport,
            arrivalAirport: mockArrivalAirport,
            departureDate: '2024-12-05',
            returnDate: '2024-12-06',
            seatTypeMapping: { 0: "Economy", 1: "Business" },
            passengers: null,
            isRoundTrip: true,
            selectedAirlines: [],
        },
        setSearchData: setSearchDataMock,
    });

    const pushMock = jest.fn();
    (useRouter as jest.Mock).mockReturnValue({ push: pushMock });

    // Render the SearchBox component
    render(
        <SearchBox
            airports={[mockDepartureAirport, mockArrivalAirport]}
            showRequiredFieldPopup={showRequiredFieldPopupMock}  // Ensure it's passed here
        />
    );

    // Simulate clicking the "Search Flights" button
    fireEvent.click(screen.getByText('Search Flights'));  // Assuming 'Search' is the button text

    // Wait for the mock function to be called
    await waitFor(() => {
        expect(showRequiredFieldPopupMock).toHaveBeenCalledWith("Passengers");
    });
});
```

SearchButtonBox:

Original Coverage: 83.33%

The original test suite did not provide coverage of the default component rendering size. A test was developed to ensure when rendered, the component size was as expected w-[200px]. The addition of this test brought the coverage of this component to 100%.

Missing Test Coverage (Line 30):

```
20  const SearchBoxButton = forwardRef<HTMLDivElement, SearchBoxButtonProps>(
21    (
22      {
23        leftIcon,
24        rightIcon,
25        headerText,
26        mainTextLeft,
27        subTextLeft,
28        mainTextRight,
29        subTextRight,
30        size = "w-[200px]",
31        className = "",
32        onClickLeftIcon,
33        onClickRightIcon,
34      },
```

Test Cases to Resolve Coverage:

```
describe("SearchBoxButton", () => {
    it("renders with the correct default size class", () => {
        const { container } = render(<SearchBoxButton headerText="Search" />);

        // Check if the rendered component has the default size class "w-[200px]"
        expect(container.firstChild).toHaveClass("w-[200px]");
    });

    it("renders with a custom size class when provided", () => {
        const { container } = render(
            <SearchBoxButton headerText="Search" size="w-[300px]" />
        );

        // Check if the rendered component has the custom size class "w-[300px]"
        expect(container.firstChild).toHaveClass("w-[300px]");
    });
});
```

SearchButtonBoxList:

Original Coverage: 75%

The SearchButtonBoxList component was missing test coverage to ensure that children passed to it are rendered correctly. This missing coverage was resolved with one test that confirmed the intended rendering of the children passed to the component.

Missing Test Coverage:

```
const SearchBoxButtonList = ({ children, className = "", ...props }: SearchBoxButtonListProps) => {
```

Test Case to Resolve Coverage:

```
import SearchBoxButtonList from "@/components/SearchBoxButtonList";

describe("SearchBoxButtonList", () => {
    test("renders children correctly", () => {
        // Render SearchBoxButtonList with some children
        render(
            <SearchBoxButtonList>
                <div>Child Element 1</div>
                <div>Child Element 2</div>
            </SearchBoxButtonList>
        );

        // Assert that the children are rendered in the component
        expect(screen.getByText("Child Element 1")).toBeInTheDocument();
        expect(screen.getByText("Child Element 2")).toBeInTheDocument();
    });
});
```

SearchBoxButtonOneSide:

Original Coverage: 77.78%

The initial test suite failed to provide coverage confirming the size (width) of the SearchBoxButtonOneSide component when rendered. Missed also in the initial suite was coverage of the event handlers within the component. The first test added ensures accurate rendering andt the second confirms that the event handlers of this component act as intended.

Missing Component Size Rendering Coverage (Line 24):

```
18    const SearchBoxButtonOneSide: FC<SearchBoxButtonOneSideProps> = ({
19        leftIcon,
20        rightIcon,
21        headerText,
22        mainText,
23        subText,
24        size = "w-[200px]",
25        className = "",
26        onClickLeftIcon,
27        onClickRightIcon
28    }) => {
```

Missing Component onClick Coverage (Line 56):

```
55     rightIcon && (
56        <div onClick={onClickRightIcon} className="cursor-pointer">
57            {rightIcon}
58        </div>
```

Confirmation of Size rendering:

```
it("renders with the correct default size class", () => {
    const onClickLeftIcon = jest.fn();
    const onClickRightIcon = jest.fn();
    const { container } = render(<SearchBoxButtonOneSide
        headerText="Header"
        mainText="Main Text"
        subText="Sub Text"
    />);

    // Check if the rendered component has the default size class "w-[200px]"
    expect(container.firstChild).toHaveClass("w-[200px]");
});
```

Confirmation of OnClick Events:

```
test("calls onClick handlers when icons are clicked", () => {
    const onClickLeftIcon = jest.fn();
    const onClickRightIcon = jest.fn();

    const leftIcon = <span data-testid="left-icon">Left</span>;
    const rightIcon = <span data-testid="right-icon">Right</span>;

    // Render the component with onClick handlers
    render(
        <SearchBoxButtonOneSide
            headerText="Header"
            mainText="Main Text"
            subText="Sub Text"
            leftIcon={leftIcon}
            rightIcon={rightIcon}
            onClickLeftIcon={onClickLeftIcon}
            onClickRightIcon={onClickRightIcon}
        />
    );

    // Simulate clicks on the icons
    fireEvent.click(screen.getByTestId("left-icon"));
    fireEvent.click(screen.getByTestId("right-icon"));

    // Assert that the onClick handlers were called
    expect(onClickLeftIcon).toHaveBeenCalledTimes(1);
    expect(onClickRightIcon).toHaveBeenCalledTimes(1);
});
```

SearchButton:

Original Coverage: 80%

The SearchButton component did not have coverage that confirmed accurate rendering of the mainText prop value passed to it. One test was developed that passed a mainText value to the component and ensured it appeared on the page as expected. This brought the component coverage to 100%.

Without Coverage (Line 12):

```
11  const SearchButton: FC<SearchButtonProps> = ({
12      mainText,
13      className = "",
14      customTextColour = 'text-white',
15      onClick,
16
17  }) => {
```

Test to Resolve Coverage:

```
describe("SearchButton", () => {
    test("renders mainText correctly", () => {
        const mainText = "Search";

        // Render the SearchButton with mainText prop
        render(<SearchButton mainText={mainText} />);

        // Assert that the main text is rendered
        expect(screen.getByText(mainText)).toBeInTheDocument();
    });

});
```

SearchItem:

Original Coverage: 80%

This component did not have coverage of the airline logo path being passed to it. Two test cases were added to bring the coverage of this component to 100%. The first test case confirms that the correct airline logo path is rendered, and the confirms that when no airline logo path argument is passed, a default image is rendered in its place.

Code Without Coverage:

```
{/* Box for Left Icon and Airline */}
<div className="flex items-center w-[200px]">
  <Image
    src={flight.airline.logo_path || "/images/default.png"}
    alt="Left Icon"
    width={36}
    height={36}
    className="mr-2"
  />
```

Test Cases to Resolve Coverage:

```
it("renders the correct logo when logo_path is provided", () => {
    render(<SearchItem index={0} flight={mockFlight} onClick={() => { }} />);

    const imgElement = screen.getByAltText("Left Icon");
    expect(imgElement).toHaveAttribute("src", "/_next/image?url=%2Fimages%2Fairline-logos%2FTAM.png&w=96&q=75");
});

it("renders the default logo when logo_path is not provided", () => {
    const mockFlightWithoutLogo = {
        ...mockFlight,
        airline: {
            ...mockFlight.airline,
            logo_path: "", // No logo
        },
    };

    render(<SearchItem index={0} flight={mockFlightWithoutLogo} onClick={() => { }} />);

    const imgElement = screen.getByAltText("Left Icon");
    expect(imgElement).toHaveAttribute("src", "/_next/image?url=%2Fimages%2Fdefault.png&w=96&q=75");
});
```

SearchResultsExpansion:

Original Coverage: 90%

This component was missing test coverage for cases where no flight data is passed to it. To address this, a test case was added to confirm that when flight is set to undefined, no flight-related values are displayed on the page.

Code Without Coverage:

```
if (!flight) return null; // Render nothing if no flight is selected
```

Test Case to Resolve Coverage:

```
describe("SearchResultExpansion", () => {

    test("renders nothing if no flight prop is provided", () => {
        render(<SearchResultExpansion flight={undefined} onClick={mockOnClick} />);
        expect(screen.queryByText("CDG")).not.toBeInTheDocument();
        expect(screen.queryByText("GRU")).not.toBeInTheDocument();
        expect(screen.queryByText("Book My Ticket Now")).not.toBeInTheDocument();
    });
});
```

SearchResultsOverviewBox:

Original Coverage: 56.25%

The initial test suite lacked coverage for verifying that the component conditionally renders the single-user or multiple-user icon based on the number of passengers and that the arrival and departure airports are rendered correctly. The test cases validate that the SearchResultsOverviewBox component renders correctly depending on the number of passengers in tripData. Specifically, the tests confirm that the correct icon (Users or UserIcon) and traveler count are displayed based on the number of passengers passed from tripData. As well as a test that confirms the appropriate airport details appear in the rendering.

Code Without Coverage:

```
<SearchBoxButtonOneSide
  leftIcon={
    (tripData.trip?.passengers?.length ?? 0) > 1 ? (
      <Users className="text-gray-500 h-4 w-4" data-testid="users-icon" />
    ) : (
      <UserIcon className="text-gray-500 h-4 w-4" data-testid="user-icon" />
    )
  }
  headerText="TRAVELERS"
  mainText={`${tripData.trip?.passengers?.length || 1}`}
  subText=""
  size="w-[120px]"
/>
</SearchBoxButtonList>
```

Test Cases 1-3 to Resolve Coverage:

```tsx
it("should render traveler count with appropriate icon when there are passengers", () => {
    const mockTripData = {
        trip: {
            passengers: [{}, {}], // Two passengers
        },
    };

    const useTripStoreMock = useTripStore as jest.MockedFunction<typeof useTripStore>;
    useTripStoreMock.mockReturnValue({ tripData: mockTripData });

    render(<SearchResultsOverviewBox />);

    // Check if traveler count is displayed correctly
    expect(screen.getByText("2")).toBeInTheDocument(); // Two passengers

    // Check if the correct icon (Users for multiple passengers) is displayed
    expect(screen.queryByTestId("users-icon")).toBeInTheDocument();
});
```

```tsx
it("should render '1' traveler with user icon for single passenger", () => {
    const mockTripData = {
        trip: {
            passengers: [{}], // One passenger
        },
    };

    const useTripStoreMock = useTripStore as jest.MockedFunction<typeof useTripStore>;
    useTripStoreMock.mockReturnValue({ tripData: mockTripData });

    render(<SearchResultsOverviewBox />);

    // Check if traveler count is displayed correctly
    expect(screen.getByText("1")).toBeInTheDocument(); // One passenger

    // Check if the correct icon (User for single passenger) is displayed
    expect(screen.queryByTestId("user-icon")).toBeInTheDocument();
});
```

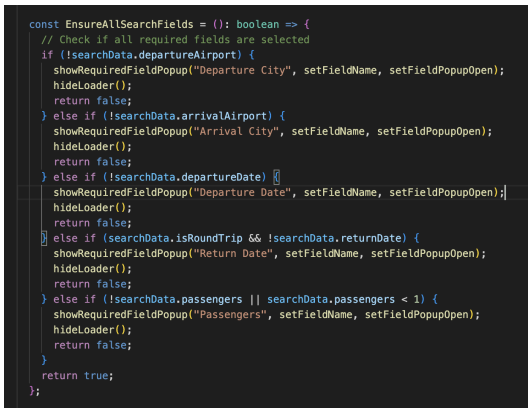
```tsx
it("should display departure and arrival cities correctly", () => {
    const mockTripData = {
        current_flight: {
            departure_airport: {
                iata_code: "NYC",
                municipality_name: "New York",
                short_name: "JFK",
            },
            arrival_airport: {
                iata_code: "LAX",
                municipality_name: "Los Angeles",
                short_name: "LAX",
            },
        },
        trip: {
            is_round_trip: true,
            passengers: [{}, {}], // Two passengers
        },
    };

    const useTripStoreMock = useTripStore as jest.MockedFunction<typeof useTripStore>;
    useTripStoreMock.mockReturnValue({ tripData: mockTripData });

    render(<SearchResultsOverviewBox />);

    // Check if the departure and arrival cities are rendered correctly
    expect(screen.getByText("NYC")).toBeInTheDocument();
    expect(screen.getByText("New York")).toBeInTheDocument();
    expect(screen.getByText("JFK")).toBeInTheDocument();
    expect(screen.getByText("Los Angeles")).toBeInTheDocument();
});
```

SearchScroll:

Original Coverage: 56.25%

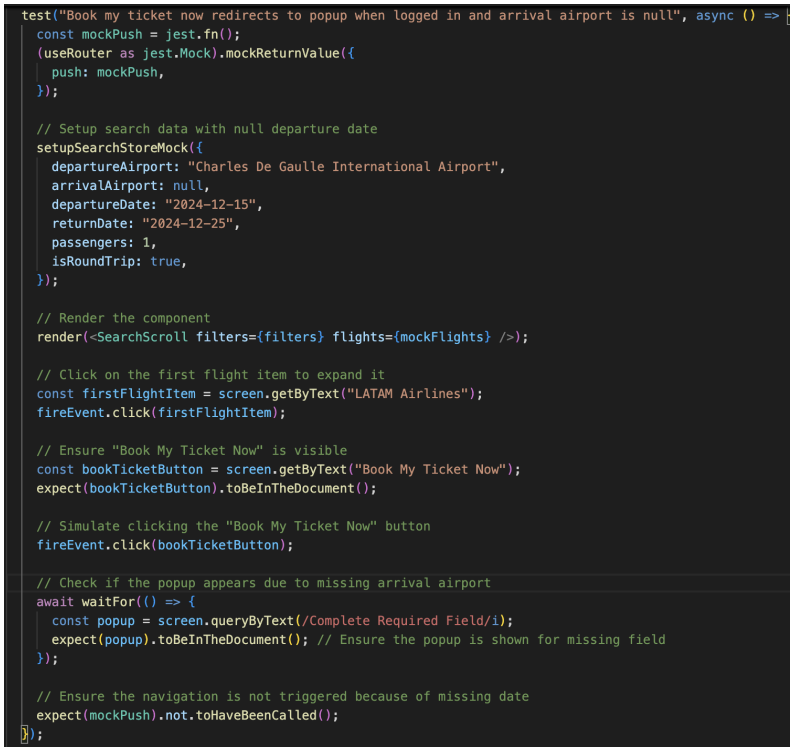
This component was missing test coverage of the conditional rendering of a warning popup to the user for a missing search input field. The ‘else if’ nature of the below code meant each conditional case was tested in its own test case, whereby one of the search input values was missing and the test confirmed the necessary pop-up was rendered thereof, warning the user of the absent value when they attempt to proceed to the next page. A final test case was included that ensures the pop-up does not appear when all search values are present.

Code without Coverage:



```
const EnsureAllSearchFields = (): boolean => {
  // Check if all required fields are selected
  if (!searchData.departureAirport) {
    showRequiredFieldPopup("Departure City", setFieldName, setFieldPopupOpen);
    hideLoader();
    return false;
  } else if (!searchData.arrivalAirport) {
    showRequiredFieldPopup("Arrival City", setFieldName, setFieldPopupOpen);
    hideLoader();
    return false;
  } else if (!searchData.departureDate) {
    showRequiredFieldPopup("Departure Date", setFieldName, setFieldPopupOpen);
    hideLoader();
    return false;
  } else if (searchData.isRoundTrip && !searchData.returnDate) {
    showRequiredFieldPopup("Return Date", setFieldName, setFieldPopupOpen);
    hideLoader();
    return false;
  } else if (!searchData.passengers || searchData.passengers < 1) {
    showRequiredFieldPopup("Passengers", setFieldName, setFieldPopupOpen);
    hideLoader();
    return false;
  }
  return true;
};
```

To avoid redundancy, only one test case screenshot is included. All other test cases developed to complete the coverage of this component are permutations of this exact test - examining the absence of different input values.



```
test("Book my ticket now redirects to popup when logged in and arrival airport is null", async () => {
  const mockPush = jest.fn();
  (useRouter as jest.Mock).mockReturnValue({
    push: mockPush,
  });

  // Setup search data with null departure date
  setupSearchStoreMock({
    departureAirport: "Charles De Gaulle International Airport",
    arrivalAirport: null,
    departureDate: "2024-12-15",
    returnDate: "2024-12-25",
    passengers: 1,
    isRoundTrip: true,
  });

  // Render the component
  render(<SearchScroll filters={filters} flights={mockFlights} />);

  // Click on the first flight item to expand it
  const firstFlightItem = screen.getByText("LATAM Airlines");
  fireEvent.click(firstFlightItem);

  // Ensure "Book My Ticket Now" is visible
  const bookTicketButton = screen.getByText("Book My Ticket Now");
  expect(bookTicketButton).toBeInTheDocument();

  // Simulate clicking the "Book My Ticket Now" button
  fireEvent.click(bookTicketButton);

  // Check if the popup appears due to missing arrival airport
  await waitFor(() => {
    const popup = screen.queryByText(/Complete Required Field/i);
    expect(popup).toBeInTheDocument(); // Ensure the popup is shown for missing field
  });

  // Ensure the navigation is not triggered because of missing date
  expect(mockPush).not.toHaveBeenCalled();
});
```

EditProfile:

Original Coverage: 27.27%

Original test cases did not verify onChange event handlers for required fields. This was an issue for the first name, last name, email, and phone number fields. The same test was

performed for each, where the component is rendered, an event is fired to change the input, and it checks that the onChange function was called with the intended string.

```
test("Update firstName onChange event", () => {
  render(
    <EditProfile
      handleSubmit={mockHandleSubmit}
      firstName=""
      setFirstName={mockSetFirstName}
      lastName=""
      setLastName={mockSetLastName}
      email=""
      setEmail={mockSetEmail}
      phoneNumber=""
      setPhoneNumber={mockSetPhoneNumber}
    />
  );

  const firstNameInput = screen.getByPlaceholderText("First Name*");
  fireEvent.change(firstNameInput, { target: { value: "John" } });
  expect(mockSetFirstName).toHaveBeenCalledWith("John");
});
```

EditProfile onChange Testing

```
<Input
  name="firstName"
  placeholder="First Name*"
  required
  value={firstName}
  onChange={(e) => setFirstName(e.target.value)}
/>
```

ProfileHeader:

Original Coverage: 91.84%

Original test cases did not ensure that the login signup modal was able to be closed when the X button was clicked. To test this, the header component is rendered and the login button is pressed to open the modal. Then, the X in the upper right corner is found and a user event is fired, expecting the login modal to not be in the document anymore.

Header closeDrawer Test

```
const closeDrawer = () => {
  setIsPopoutOpen(false);
};
```

```
test("Clicking X on the popout closes the popout", async () => {
  render(
    <Header
      headerSize="small"
      backgroundImage={true}
      logoColour="black"
      displayProfilePicture={true}
    />
  );

  const user = userEvent.setup();
  const loginButton = screen.getByTestId("login-button");
  await user.click(loginButton);

  const loginPopout = await screen.findByTestId("login-popout");
  const xButton = screen.getByRole("button", { name: "Close" });

  await user.click(xButton);

  expect(loginPopout).not.toBeInTheDocument();
});
```

HelperFunctions/timeFilter:

Original Coverage: 0%

The timeFilter previously had no testing. The timeFilter file contains a getTimeCategory function that is used to categorize a 12 hour time to morning, afternoon, or evening. One test was written to test the base and edge cases of each scenario.

getTimeCategory Tests

```
if (isAM) {
    // Morning: 12:00 AM (midnight) to 11:59 AM
    return "Morning";
} else {
    // Afternoon: 12:00 PM to 4:59 PM
    if (hourNum === 12 || (hourNum >= 1 && hourNum < 5)) {
        return "Afternoon";
    }
    // Evening: 5:00 PM to 11:59 PM
    return "Evening";
}
```

```
describe("getTimeCategory", () => {
  test("returns 'Morning' for times in the morning range", () => {
    expect(getTimeCategory("12:00 AM")).toBe("Morning"); // Midnight
    expect(getTimeCategory("6:00 AM")).toBe("Morning"); // Early morning
    expect(getTimeCategory("11:59 AM")).toBe("Morning"); // End of morning
  });

  test("returns 'Afternoon' for times in the afternoon range", () => {
    expect(getTimeCategory("12:00 PM")).toBe("Afternoon"); // Noon
    expect(getTimeCategory("1:00 PM")).toBe("Afternoon"); // Early afternoon
    expect(getTimeCategory("4:59 PM")).toBe("Afternoon"); // End of afternoon
  });

  test("returns 'Evening' for times in the evening range", () => {
    expect(getTimeCategory("5:00 PM")).toBe("Evening"); // Start of evening
    expect(getTimeCategory("8:00 PM")).toBe("Evening"); // Middle of evening
    expect(getTimeCategory("11:59 PM")).toBe("Evening"); // Late evening
  });
});
```

Images/AirlinePhoto:

Original Coverage: 0%

The AirlinePhoto component previously had no testing. The purpose of this component is to display airline photos with the given image path prop. Two tests were written. The first test is to render the component with a custom image path. The second test is to render the component with the default image path, for when one is not provided.

```
const AirlinePhoto: React.FC<AirlinePhotoProps> = ({
  imagePath = "/images/default.png",
  testid,
}) => (
  <div
    style={{
      position: "relative",
      width: "40px",
      height: "40px",
      overflow: "hidden",
    }}
  >
    <Image
      src={imagePath}
      alt="Airline photo"
      fill
      style={{ position: "absolute", objectFit: "cover" }}
      data-testid={testid}
    />
  </div>
);
```

AirlinePhoto Tests

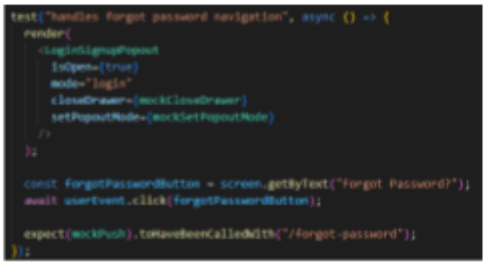
```
describe("AirlinePhoto Component", () => {
  test("renders with a custom image path", () => {
    const customPath = "/images/custom.png";
    render(<AirlinePhoto imagePath={customPath} />);

    const imageElement = screen.getByAltText("Airline photo");
    expect(imageElement).toBeInTheDocument();
    expect(imageElement).toHaveAttribute("src", customPath);
  });

  test("renders with the default image when no imagePath is provided", () => {
    render(<AirlinePhoto />);

    const imageElement = screen.getByAltText("Airline photo");
    expect(imageElement).toBeInTheDocument();
    expect(imageElement).toHaveAttribute("src", "/images/default.png");
  });
});
```

LoginSignupPopup:

Original Coverage: 60.42%

This component originally had no specific tests written for it, but had lots of coverage from other tests that overlapped. Naturally, none of the functionality of the popout was being tested, including toggling between login and signup modes, handling successful login submission, handling forgot password navigation, and the props and interfaces that it requires.

LoginSignupPopout Tests



LoginSignupPopout Missing Coverage

```
const handleForgotPassword = () => {
  router.push("/forgot-password");
};
```

```
// Toggle between login and signup modes
const toggleLoginSignup = () => {
  if (isLogin) {
    setPopoutMode("signup");
  } else {
    setPopoutMode("login");
  }
};
```

```
// Handle form submission for standard login/signup
const handleLoginSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  // Simulate successful login
  try {
    const data = await fetchLogin();

    if (data.token) {
      signIn(data.guid);
      closeDrawer();
      router.push("/");
    }
  } catch (error) {
    console.error(error);
  }
};
```

PurchaseItem:

Original Coverage: 66.67%

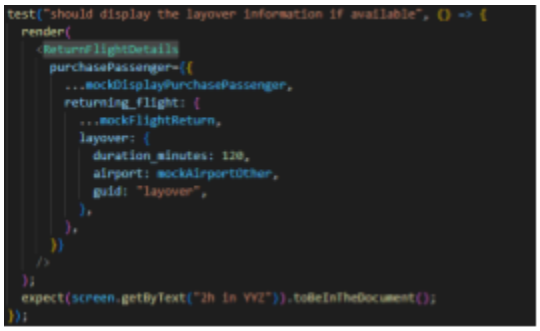
The code that was not covered in this component was related to handling null or undefined values relating to flights and passengers, and the rendering of a layover for the returning flight. The following tests were written to ensure that null passengers are handled correctly, null flights are handled correctly, and layover information is rendered correctly.

PurchaseItem Missing Coverage

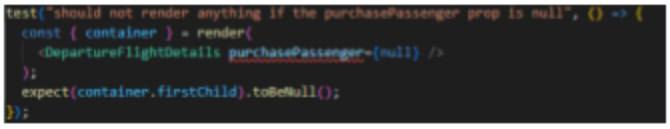
```
export const ReturnFlightDetails: React.FC<{
  purchasePassenger?: DisplayPurchasePassenger;
}> = ({ purchasePassenger }) => {
  const {
    returning_flight: returningFlight = null,
    return_date: returnDate = null,
    return_seat: returnSeat = null,
  } = purchasePassenger || {};
  const formattedReturnDate = returnDate
    ? format(new Date(returnDate), "MMMM do, yyyy")
    : "Date not available";

  if (!returningFlight) return null;
```

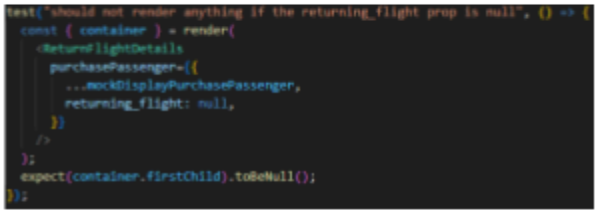
```
export const DepartureFlightDetails: React.FC<{
  purchasePassenger?: DisplayPurchasePassenger;
}> = ({ purchasePassenger }) => {
  const {
    departing_flight: departingFlight = null,
    departure_date: departureDate = null,
    departure_seat: departureSeat = null,
  } = purchasePassenger || {};
  const formattedDepartureDate = departureDate
    ? format(new Date(departureDate), "MMMM do, yyyy")
    : "Date not available";

  if (!departingFlight) return null;
```

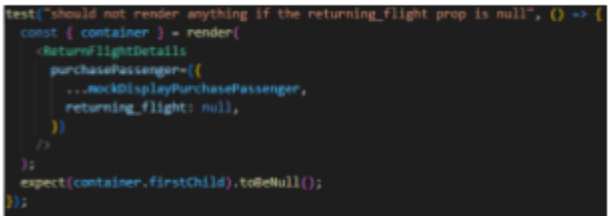
```
{/* Flight duration, times, and layover information */}
<div className="flex flex-col text-base text-right min-h-[74px] text-slate-800">
  <p>{formatTimeMinutes(returningFlight.flight_time_minutes)}</p>
  <p className="mt-1">
    {returningFlight.departure_time} -{" "}
    {returningFlight.arrival_time}
  </p>
  {returningFlight.layover && (
    <p className="mt-1 text-black">
      {getLayoverSummary(returningFlight)}
    </p>
  )}
</div>
```



PurchaseItem New Tests







---

models/helper_functions:

Original Coverage: 27.69%

The helper_functions file previously had no tests written, but had some coverage from other tests overlapping. The helper_functions file contains many functions to help with operations that are repeated many times, such as calculating prices, mapping data types to another data type, and getting a summary of an object. There were many smaller tests written, but I will highlight some larger ones, specifically for getting Trip to Purchase mapping, and price and seat type calculations.

```
describe("Price and Seat Type Calculations", () => {
  const economyMapping: SeatTypeMapping = { 1: "Economy", 2: "Economy" };
  const businessMapping: SeatTypeMapping = { 1: "Business", 2: "Business" };
  const mixedMapping: SeatTypeMapping = { 1: "Economy", 2: "Business" };

  test("returns business price for all business seats", () => {
    expect(getPriceByPassengerType(businessMapping, mockFlightReturn)).toBe(
      300
    );
  });

  test("returns economy price for mixed or economy seats", () => {
    expect(getPriceByPassengerType(economyMapping, mockFlightReturn)).toBe(200);
    expect(getPriceByPassengerType(mixedMapping, mockFlightReturn)).toBe(200);
  });

  test("returns correct seat type based on mapping", () => {
    expect(getSeatTypeByPassengerType(businessMapping)).toBe("Business");
    expect(getSeatTypeByPassengerType(economyMapping)).toBe("Economy");
    expect(getSeatTypeByPassengerType(mixedMapping)).toBe("Economy");
  });
});
```

Price and Seat Type Calculation Tests

```
export function getPriceByPassengerType(seatTypeMapping: SeatTypeMapping, flight: Flight) {
    const seatTypes = Object.values(seatTypeMapping || {});
    const allBusiness = seatTypes.every((type) => type === "business");

    if (allBusiness) return flight.price_business;
    else return flight.price_economy;
};

export function getSeatTypeByPassengerType(seatTypeMapping: SeatTypeMapping) {
    const seatTypes = Object.values(seatTypeMapping || {});
    const allBusiness = seatTypes.every((type) => type === "business");

    if (allBusiness) return "Business";
    else return "Economy";
};
```

Trip to Purchase Mapping Tests

```
export function mapTripToPurchase(trip: Trip): DisplayPurchase {
    // Check if passengers is undefined or null and handle it
    const passengers: DisplayPurchasePassenger[] = trip.passengers?.map((passenger, index) => ({
        name: getNameOrDefault(passenger.name, index),
        departing_flight: trip.departing_flight ?? null,
        returning_flight: trip.returning_flight ?? null,
        departure_seat: trip.departing_flight ? getDepartingFlightSeatSummary(trip, index) : "No seat assigned",
        return_seat: trip.returning_flight ? getReturningFlightSeatSummary(trip, index) : "No seat assigned",
        departure_date: trip.departure_date ?? null,
        return_date: trip.return_date ?? null,
    })) || []; // Fallback to an empty list

    const subtotal = calculateSubtotal(passengers, trip);
    const tax = calculateTax(subtotal);
    const total = Math.round((subtotal + tax) * 100) / 100; // Round to 2 decimals

    return {
        guid: trip.guid ?? "",
        title: trip.name,
        passengers,
        subtotal,
        taxes: tax,
        total_cost: total,
    };
}
```

```
describe("Trip to Purchase Mapping", () => {
  test("correctly maps trip to purchase display", () => {
    const purchase = mapTripToPurchase(mockTrip);

    expect(purchase.guid).toBe(mockTrip.guid);
    expect(purchase.title).toBe(mockTrip.name);
    expect(purchase.passengers).toHaveLength(2);
    expect(purchase.passengers[0].name).toBe("John Doe");
    expect(typeof purchase.subtotal).toBe("number");
    expect(typeof purchase.taxes).toBe("number");
    expect(typeof purchase.total_cost).toBe("number");
  });

  test("handles missing passenger data", () => {
    const emptyTrip: Trip = {
      ...mockTrip,
      passengers: [],
    };
    const purchase = mapTripToPurchase(emptyTrip);
    expect(purchase.passengers).toHaveLength(0);
  });
});
```

Passenger_form_data

Original Coverage: 0%

The PassengerFormData interface previously had no testing. Its purpose is to take the PassengerFormData interface and return a Passenger object. Tests were written to test the transformation with only required fields, test the transformation with all fields, test existing passenger data preservation, test form data override behaviour, and to test the handling of optional fields. The tests are shown on the next page.

```ts
export function transformToPassenger( formData: PassengerFormData, existingPassenger?: Passenger ): Passenger {
  return {
    guid: existingPassenger?.guid ?? "null",
    trip_id: existingPassenger?.trip_id ?? "null",
    name: formData.name,
    departing_seat_id: existingPassenger?.departing_seat_id ?? 0,
    returning_seat_id: existingPassenger?.returning_seat_id ?? null,
    middle: formData.middle || "",
    last: formData.last,
    prefix: formData.prefix || "",
    dob: formData.dob || new Date(),
    passport_number: formData.passport_number || "",
    known_traveller_number: formData.known_traveller_number || "",
    email: formData.email || "",
    phone: formData.phone || "",
    street_address: formData.street_address || "",
    apt_number: formData.apt_number || "",
    province: formData.province || "",
    zip_code: formData.zip_code || "",
    emerg_name: formData.emerg_name || "",
    emerg_last: formData.emerg_last || "",
    emerg_email: formData.emerg_email || "",
    emerg_phone: formData.emerg_phone || ""
  };
}
```

PassengerFormData Tests

```ts
test("should preserve existing passenger data for non-form fields", () => {
  const result = transformToPassenger(minimalFormData, existingPassenger);

  expect(result.guid).toBe(existingPassenger.guid);
  expect(result.trip_id).toBe(existingPassenger.trip_id);
  expect(result.departing_seat_id).toBe(existingPassenger.departing_seat_id);
  expect(result.returning_seat_id).toBe(existingPassenger.returning_seat_id);
});

test("should override existing passenger data with form data", () => {
  const result = transformToPassenger(fullFormData, existingPassenger);

  expect(result.name).toBe(fullFormData.name);
  expect(result.middle).toBe(fullFormData.middle);
  expect(result.last).toBe(fullFormData.last);
  expect(result.email).toBe(fullFormData.email);
});
```

```ts
test("should handle undefined optional fields correctly", () => {
  const partialFormData: PassengerFormData = {
    name: "John",
    middle: undefined,
    last: undefined,
    email: undefined,
  };

  const result = transformToPassenger(partialFormData);

  expect(result.middle).toBe("");
  expect(result.last).toBeUndefined();
  expect(result.email).toBe("");
});
```

HelperFunctions/setTripContextData:

Original Coverage: 74.36%

The missing coverage in setTripContextData was primarily authentication validation, and search fields validation. To test these, tests were written to verify proper handling when the user is not authenticated, ensure the showing of the sign-in popup, confirm redirect to home page, and ensure the trip creation process is halted for unauthenticated users. The search fields were validated by testing required search fields, and confirming proper loader state management.