```
import React from 'react';
import { render, screen, fireEvent, waitFor } from '@testing-library/
react';
import userEvent from '@testing-library/user-event';
import SignUp from '@/app/(auth-pages)/sign-up/page';
import '../../../jest-dom-setup';

// Mock console.error to avoid polluting test output
const originalConsoleError = console.error;
console.error = jest.fn();

// Mock next/navigation
const mockPush = jest.fn();
jest.mock('next/navigation', () => ({
  useRouter: () => ({
    push: mockPush,
  }),
}));

// Mock the Message component
jest.mock('@/components/ui/message', () => ({
  Message: ({ type, message, onDismiss }: { type: string; message:
string; onDismiss?: () => void }) => (
    <div data-testid={`message-${type}`} role={type === 'error' ?
'alert' : 'status'} onClick={onDismiss}>{message}</div>
  ),
  SearchParamsMessage: () => <div data-testid="search-params-
message" />,
}));

describe('SignUp Page', () => {
  beforeEach(() => {
    // Reset fetch mock and router mock before each test
    global.fetch = jest.fn();
    mockPush.mockReset();
  });

  afterAll(() => {
    // Restore console.error
    console.error = originalConsoleError;
  });

  it('renders the signup form correctly', () => {
    render(<SignUp />);

    // Check for page elements
    expect(screen.getByText('Course Mix')).toBeInTheDocument();
    expect(screen.getByText('Create your
account')).toBeInTheDocument();
    expect(screen.getByText('Join Course Mix to start planning your
```

```
academic journey')).toBeInTheDocument();
    expect(screen.getByLabelText(/Email address/
i)).toBeInTheDocument();
    expect(screen.getByRole('button', { name: /Sign up/
i })).toBeInTheDocument();
    expect(screen.getByText('Only @brocku.ca email addresses are
allowed.')).toBeInTheDocument();
    expect(screen.getByRole('button', { name: /Sign in/
i })).toBeInTheDocument();
  });

  it('shows validation error for empty email', async () => {
    render(<SignUp />);

    // Submit form without entering email
    const submitButton = screen.getByRole('button', { name: /Sign up/
i });
    fireEvent.click(submitButton);

    // Since HTML validation prevents form submission with empty
required fields,
    // we need to check for the browser's default validation message
    const emailInput = screen.getByLabelText(/Email address/i);
    expect(emailInput).toBeInTheDocument();
    expect(emailInput).toHaveAttribute('required');

    // We can also check if the form was prevented from submitting
    expect(mockPush).not.toHaveBeenCalled();
    expect(global.fetch).not.toHaveBeenCalled();
  });

  it('shows validation error for non-Brock University email', async ()
=> {
    render(<SignUp />);

    // Enter a non-Brock email
    const emailInput = screen.getByLabelText(/Email address/i);
    await userEvent.type(emailInput, 'test@example.com');

    // Submit form
    const submitButton = screen.getByRole('button', { name: /Sign up/
i });
    fireEvent.click(submitButton);

    // Check if appropriate error message appears
    await waitFor(() => {
      const errorElement = screen.getByTestId('message-error');
      expect(errorElement).toBeInTheDocument();
      expect(errorElement).toHaveTextContent('Only @brocku.ca emails
are allowed');
```

```
    });
  });

  it('validates uppercase Brock University email domain correctly',
async () => {
    // Mock successful API response
    (global.fetch as jest.Mock).mockResolvedValueOnce({
      ok: true,
      json: async () => ({ success: true }),
    });

    render(<SignUp />);

    // Enter a valid Brock email with uppercase domain
    const emailInput = screen.getByLabelText(/Email address/i);
    await userEvent.type(emailInput, 'student@BROCKU.CA');

    // Submit form
    const submitButton = screen.getByRole('button', { name: /Sign up/
i });
    fireEvent.click(submitButton);

    // Wait for API call to complete
    await waitFor(() => {
      // Check if fetch was called (validation passed)
      expect(global.fetch).toHaveBeenCalled();
      expect(mockPush).toHaveBeenCalledWith(
        `/verify?email=${encodeURIComponent('student@BROCKU.CA')}`
      );
    });
  });

  it('shows validation error for invalid email format', async () => {
    render(<SignUp />);

    // Bypass HTML5 validation to test the form's internal validation
    // First get the email input element
    const emailInput = screen.getByLabelText(/Email address/i) as
HTMLInputElement;

    // Simulate user typing an invalid email that isn't caught by
HTML5 validation
    // Setting the value directly to bypass HTML5 validation
    fireEvent.change(emailInput, { target: { value: 'not-a-valid-
email' } });

    // Submit form
    const form = emailInput.closest('form') as HTMLFormElement;
    fireEvent.submit(form);
```

```
    // Check that API wasn't called and form submission was prevented
    expect(global.fetch).not.toHaveBeenCalled();
  });

  it('shows validation error for emails with typos in the brocku.ca
domain', async () => {
    render(<SignUp />);

    // Enter email with typo in the domain
    const emailInput = screen.getByLabelText(/Email address/i);
    await userEvent.type(emailInput, 'student@brockuu.ca');

    // Submit form
    const submitButton = screen.getByRole('button', { name: /Sign up/
i });
    fireEvent.click(submitButton);

    // Check if appropriate error message appears
    await waitFor(() => {
      const errorElement = screen.getByTestId('message-error');
      expect(errorElement).toBeInTheDocument();
      expect(errorElement).toHaveTextContent('Only @brocku.ca emails
are allowed');
    });
  });

  it('submits the form with valid Brock University email', async () =>
{
    // Mock successful API response
    (global.fetch as jest.Mock).mockResolvedValueOnce({
      ok: true,
      json: async () => ({ success: true }),
    });

    render(<SignUp />);

    // Enter a valid Brock email
    const emailInput = screen.getByLabelText(/Email address/i);
    await userEvent.type(emailInput, 'student@brocku.ca');

    // Submit form
    const submitButton = screen.getByRole('button', { name: /Sign up/
i });
    fireEvent.click(submitButton);

    // Wait for API call to complete
    await waitFor(() => {
      // Check if fetch was called with correct parameters
      expect(global.fetch).toHaveBeenCalledWith(
        '/api/auth/send-verification',
```

```
        expect.objectContaining({
          method: 'POST',
          headers: expect.objectContaining({
            'Content-Type': 'application/json',
          }),
          body: JSON.stringify({ email: 'student@brocku.ca' }),
        })
      );
    });

    // Verify router.push is called (now in a separate waitFor to
handle async state updates)
    await waitFor(() => {
      expect(mockPush).toHaveBeenCalledWith(
        `/verify?email=${encodeURIComponent('student@brocku.ca')}`
      );
    });
  });

  it('handles API error response', async () => {
    // Mock error API response
    (global.fetch as jest.Mock).mockResolvedValueOnce({
      ok: false,
      json: async () => ({ error: 'Account with this email already
exists' }),
    });

    render(<SignUp />);

    // Enter a valid Brock email
    const emailInput = screen.getByLabelText(/Email address/i);
    await userEvent.type(emailInput, 'student@brocku.ca');

    // Submit form
    const submitButton = screen.getByRole('button', { name: /Sign up/
i });
    fireEvent.click(submitButton);

    // Wait for API call to complete and check if error message
appears
    await waitFor(() => {
      const errorElement = screen.getByTestId('message-error');
      expect(errorElement).toBeInTheDocument();
      expect(errorElement).toHaveTextContent('Account with this email
already exists');
    });
  });

  it('handles network error during form submission', async () => {
    // Mock network error
```

```
    (global.fetch as jest.Mock).mockRejectedValueOnce(new
Error('Network error'));

    render(<SignUp />);

    // Enter a valid Brock email
    const emailInput = screen.getByLabelText(/Email address/i);
    await userEvent.type(emailInput, 'student@brocku.ca');

    // Submit form
    const submitButton = screen.getByRole('button', { name: /Sign up/
i });
    fireEvent.click(submitButton);

    // Wait for error to be handled and check if error message appears
    await waitFor(() => {
      const errorElement = screen.getByTestId('message-error');
      expect(errorElement).toBeInTheDocument();
      expect(errorElement).toHaveTextContent('An unexpected error
occurred. Please try again.');
    });
  });

  it('shows loading state during form submission', async () => {
    // Mock a delayed response to test loading state
    (global.fetch as jest.Mock).mockImplementationOnce(() => {
      return new Promise(resolve => {
        setTimeout(() => {
          resolve({
            ok: true,
            json: async () => ({ success: true }),
          });
        }, 100);
      });
    });

    render(<SignUp />);

    // Enter a valid Brock email
    const emailInput = screen.getByLabelText(/Email address/i);
    await userEvent.type(emailInput, 'student@brocku.ca');

    // Submit form
    const submitButton = screen.getByRole('button', { name: /Sign up/
i });
    fireEvent.click(submitButton);

    // Check if loading state is shown
    expect(screen.getByText('Sending
verification...')).toBeInTheDocument();
```

```
    expect(screen.getByRole('button', { name: /Sending
verification.../i })).toBeDisabled();
  });

  it('disables the submit button while loading', async () => {
    // Mock a delayed response to test loading state
    (global.fetch as jest.Mock).mockImplementationOnce(() => {
      return new Promise(resolve => {
        setTimeout(() => {
          resolve({
            ok: true,
            json: async () => ({ success: true }),
          });
        }, 100);
      });
    });

    render(<SignUp />);

    // Enter a valid Brock email
    const emailInput = screen.getByLabelText(/Email address/i);
    await userEvent.type(emailInput, 'student@brocku.ca');

    // Submit form
    const submitButton = screen.getByRole('button', { name: /Sign up/
i });
    fireEvent.click(submitButton);

    // Verify button is disabled during loading
    expect(submitButton).toBeDisabled();

    // Wait for submission to complete and verify button is re-enabled
    await waitFor(() => {
      expect(mockPush).toHaveBeenCalled();
    });
  });

  it('allows dismissing error messages', async () => {
    render(<SignUp />);

    // Enter a non-Brock email
    const emailInput = screen.getByLabelText(/Email address/i);
    await userEvent.type(emailInput, 'test@example.com');

    // Submit form
    const submitButton = screen.getByRole('button', { name: /Sign up/
i });
    fireEvent.click(submitButton);

    // Wait for error message to appear
```

```
    let errorElement;
    await waitFor(() => {
      errorElement = screen.getByTestId('message-error');
      expect(errorElement).toBeInTheDocument();
    });

    // Click on the error message to dismiss it
    fireEvent.click(errorElement!);

    // Verify error message is removed
    await waitFor(() => {
      expect(screen.queryByTestId('message-
error')).not.toBeInTheDocument();
    });
  });

  it('navigates to sign-in page when sign-in button is clicked', () =>
{
    render(<SignUp />);

    // Verify link to sign-in page
    const linkElement = screen.getByRole('link', { name: /Sign in/
i });
    expect(linkElement).toHaveAttribute('href', '/sign-in');
  });
});
```