Group Project: Quality Assurance Activity Report, including unit tests

PA2550: SEMINARIER I PROGRAMVARUTEKNIK

October 14, 2024

Student 1	Name	Gubala Haribabu Laasya Sai
	E-Mail	lagu23@student.bth.se
	BTH ID	20021117-T129
Student 2	Name	Shyam Naga Suraj Dodla
	E-Mail	shdd23@student.bth.se
	BTH ID	20001011-T433
Student 3	Name	Harsha Vardhan Devulapalli
	E-Mail	hade23@student.bth.se
	BTH ID	20010623-T251
Student 4	Name	Bhaskar Satyendra Kumar Nagireddy
	E-Mail	bhna21@student.bth.se
	BTH ID	19990416-T214
Student 5	Name	Venkata Sai Srija Nandipati
	E-Mail	venn23@student.bth.se
	BTH ID	20030504-T147

1 Unit Testing

We have developed 2 unit tests with multiple scenarios to verify Login/Registration components of our project work as expected. We tested the $AdminLoginRegister$ component, the test focuses on rendering the login form, successful login, proper form handling for registration, and error handling when incorrect details are provided. Similarly, for the $UserLogin$ component, this test evaluates proper form rendering, the behaviour of successful and failed logins (both email/password and Google), and accurate error messaging.

AdminLoginRegister Component

Test 1: "Renders login form correctly."
We made sure that the login form is shown with all the necessary fields and buttons for the admin to log in.

```
test('renders login form correctly', () => {
  render(<AdminLoginRegister />);

  expect(screen.getByPlaceholderText('Email')).toBeInTheDocument();
  expect(screen.getByPlaceholderText('Password')).toBeInTheDocument()
   ;
```

```
  expect(screen.getByRole('button', { name: /Login as Admin/i })).
   toBeInTheDocument();
  expect(screen.queryByPlaceholderText('Enter Admin Key')).not.
   toBeInTheDocument();
});
```

Test 2: "Renders registration form when switching to register mode."
This test ensures that when the admin switches to the registration form, the 'Admin Key' field appears as expected.

```
test('renders registration form correctly when isRegister is true',
   () => {
  render(<AdminLoginRegister />);

  fireEvent.click(screen.getByText(/Register here/i));

  expect(screen.getByPlaceholderText('Email')).toBeInTheDocument();
  expect(screen.getByPlaceholderText('Password')).toBeInTheDocument()
   ;
  expect(screen.getByPlaceholderText('Enter Admin Key')).
   toBeInTheDocument();
  expect(screen.getByRole('button', { name: /Register as Admin/i })).
   toBeInTheDocument();
});
```

Test 3: "Handles successful login."
We simulated a successful admin login and checked if the user is directed to the right page afterward.

```
test('submits login form and handles successful login', async () => {
  signInWithEmailAndPassword.mockResolvedValue({
    user: { uid: '123', email: 'admin@example.com' },
  });

  const mockNavigate = jest.fn();
  useNavigate.mockReturnValue(mockNavigate);

  render(<AdminLoginRegister />);

  fireEvent.change(screen.getByPlaceholderText('Email'), {
    target: { value: 'admin@example.com' },
  });
  fireEvent.change(screen.getByPlaceholderText('Password'), {
    target: { value: 'password123' },
  });
  fireEvent.click(screen.getByRole('button', { name: /Login as Admin/
   i }));

  await waitFor(() => {
    expect(signInWithEmailAndPassword).toHaveBeenCalledWith(
      {},
      'admin@example.com',
      'password123'
    );
```

```
  });

  expect(useAuth().logSuccessfulLogin).toHaveBeenCalledWith(
    { uid: '123', email: 'admin@example.com' },
    'AdminLogin'
  );
  expect(mockNavigate).toHaveBeenCalledWith('/readmendat/admin/
   dashboard');
});
```

Test 4: "Handles successful registration with the correct admin key."

This checks if an admin is successfully registered when the correct admin key is provided and then logged in.

```
test('submits registration form with correct admin key and handles
   successful registration', async () => {
  createUserWithEmailAndPassword.mockResolvedValue({
    user: { uid: '123', email: 'newadmin@example.com' },
  });
  doc.mockReturnValue({});
  setDoc.mockResolvedValue();

  const mockNavigate = jest.fn();
  useNavigate.mockReturnValue(mockNavigate);

  render(<AdminLoginRegister />);

  fireEvent.click(screen.getByText(/Register here/i));
  fireEvent.change(screen.getByPlaceholderText('Email'), {
    target: { value: 'newadmin@example.com' },
  });
  fireEvent.change(screen.getByPlaceholderText('Password'), {
    target: { value: 'newpassword' },
  });
  fireEvent.change(screen.getByPlaceholderText('Enter Admin Key'), {
    target: { value: 'AXFRCU' },
  });
  fireEvent.click(screen.getByRole('button', { name: /Register as
   Admin/i }));

  await waitFor(() => {
    expect(createUserWithEmailAndPassword).toHaveBeenCalledWith(
      {},
      'newadmin@example.com',
      'newpassword'
    );
  });

  expect(doc).toHaveBeenCalledWith({}, 'users', '123');
  expect(setDoc).toHaveBeenCalledWith({}, { email: 'newadmin@example.
   com', isAdmin: true });
  expect(useAuth().logSuccessfulLogin).toHaveBeenCalledWith(
    { uid: '123', email: 'newadmin@example.com' },
```

```
    'AdminLogin'
  );
  expect(mockNavigate).toHaveBeenCalledWith('/readmendat/admin/
   dashboard');
});
```

Test 5: "Shows error for incorrect admin key."
This test makes sure an error message appears if the wrong admin key is entered during registration.

```
test('shows error for incorrect admin key during registration', async
    () => {
  const mockNavigate = jest.fn();
  useNavigate.mockReturnValue(mockNavigate);

  render(<AdminLoginRegister />);

  fireEvent.click(screen.getByText(/Register here/i));
  fireEvent.change(screen.getByPlaceholderText('Email'), {
    target: { value: 'newadmin@example.com' },
  });
  fireEvent.change(screen.getByPlaceholderText('Password'), {
    target: { value: 'newpassword' },
  });
  fireEvent.change(screen.getByPlaceholderText('Enter Admin Key'), {
    target: { value: 'WRONGKEY' },
  });
  fireEvent.click(screen.getByRole('button', { name: /Register as
   Admin/i }));

  await waitFor(() => {
    expect(screen.getByText('Invalid admin key')).toBeInTheDocument()
   ;
  });

  expect(createUserWithEmailAndPassword).not.toHaveBeenCalled();
  expect(useAuth().logFailedLogin).toHaveBeenCalledWith(
    'newadmin@example.com',
    'Invalid admin key',
    'AdminLogin'
  );
});
```

UserLogin Component

Test 1: "Renders login form correctly."
Verifies that the login form for a regular user is rendered with all the required elements such as email, password fields, and login button.

```
test('renders login form correctly', () => {
    render(<Login />);
```

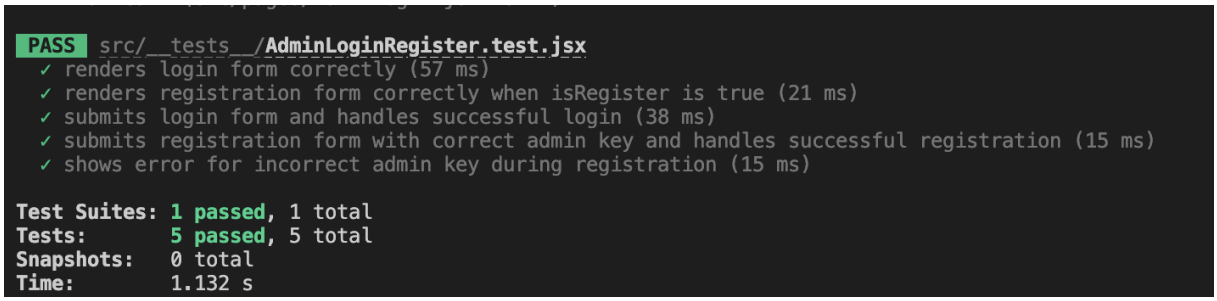

Figure 1: AdminLoginRegister Test Result

```
    expect(screen.getByRole('heading', { name: /Login/i })).
   toBeInTheDocument();
    expect(screen.getByPlaceholderText('Email')).toBeInTheDocument();
    expect(screen.getByPlaceholderText('Password')).toBeInTheDocument
   ();
    expect(screen.getByRole('button', { name: /Login/i })).
   toBeInTheDocument();
    expect(screen.getByRole('button', { name: /Sign in with Google/i
   })).toBeInTheDocument();
});
```

Test 2: "Handles successful email/password login."
Tests the successful login flow using email and password credentials and verifies the navigation after login.

```
test('handles successful email/password login', async () => {
    signInWithEmailAndPassword.mockResolvedValue({
        user: { uid: '123', email: 'test@example.com' },
    });

    const mockNavigate = jest.fn();
    useNavigate.mockReturnValue(mockNavigate);

    render(<Login />);

    fireEvent.change(screen.getByPlaceholderText('Email'), {
        target: { value: 'test@example.com' },
    });
    fireEvent.change(screen.getByPlaceholderText('Password'), {
        target: { value: 'password123' },
    });
    fireEvent.click(screen.getByRole('button', { name: /Login/i }));

    await waitFor(() => {
        expect(signInWithEmailAndPassword).toHaveBeenCalledWith(
            auth,
            'test@example.com',
            'password123'
        );
    });
```

```
    expect(useAuth().logSuccessfulLogin).toHaveBeenCalledWith(
        { uid: '123', email: 'test@example.com' },
        'EmailPassword'
    );
    expect(mockNavigate).toHaveBeenCalledWith('/');
});
```

Test 3: "Handles failed email/password login due to invalid credentials."
Checks whether an error message is displayed for invalid credentials during login.

```
test('handles failed email/password login due to invalid credentials
   ', async () => {
    const error = { code: 'auth/invalid-credential' };
    signInWithEmailAndPassword.mockRejectedValue(error);

    const mockNavigate = jest.fn();
    useNavigate.mockReturnValue(mockNavigate);

    render(<Login />);

    fireEvent.change(screen.getByPlaceholderText('Email'), {
        target: { value: 'wrong@example.com' },
    });
    fireEvent.change(screen.getByPlaceholderText('Password'), {
        target: { value: 'wrongpassword' },
    });
    fireEvent.click(screen.getByRole('button', { name: /Login/i }));

    await waitFor(() => {
        expect(signInWithEmailAndPassword).toHaveBeenCalledWith(
            auth,
            'wrong@example.com',
            'wrongpassword'
        );
    });

    expect(useAuth().logFailedLogin).toHaveBeenCalledWith(
        'wrong@example.com',
        'Invalid credentials.',
        'EmailPassword'
    );
    await waitFor(() => {
        expect(screen.getByText('Invalid credentials.')).
   toBeInTheDocument();
    });
    expect(mockNavigate).not.toHaveBeenCalled();
});
```

Test 4: "Handles successful Google login."
Simulates a successful login via Google and verifies the correct behavior and navigation after login.

```
test('handles successful Google login', async () => {
    signInWithPopup.mockResolvedValue({
```

```
        user: { uid: '123', email: 'googleuser@example.com' },
    });

    const mockNavigate = jest.fn();
    useNavigate.mockReturnValue(mockNavigate);

    render(<Login />);

    fireEvent.click(screen.getByRole('button', { name: /Sign in with
   Google/i }));

    await waitFor(() => {
        expect(signInWithPopup).toHaveBeenCalledWith(auth, provider);
    });

    expect(useAuth().logSuccessfulLogin).toHaveBeenCalledWith(
        { uid: '123', email: 'googleuser@example.com' },
        'Google'
    );
    expect(mockNavigate).toHaveBeenCalledWith('/');
});
```

Test 5: "Handles failed Google login."

Tests that an error message is displayed if the Google login attempt fails due to invalid credentials.

```
test('handles failed Google login', async () => {
    const error = { code: 'auth/invalid-credential', email: '
   googleuser@example.com' };
    signInWithPopup.mockRejectedValue(error);

    const mockNavigate = jest.fn();
    useNavigate.mockReturnValue(mockNavigate);

    render(<Login />);

    fireEvent.click(screen.getByRole('button', { name: /Sign in with
   Google/i }));

    await waitFor(() => {
        expect(signInWithPopup).toHaveBeenCalledWith(auth, provider);
    });

    expect(useAuth().logFailedLogin).toHaveBeenCalledWith(
        'googleuser@example.com',
        'Failed to sign in with Google',
        'Google'
    );
    await waitFor(() => {
        expect(screen.getByText('Failed to sign in with Google')).
   toBeInTheDocument();
    });
    expect(mockNavigate).not.toHaveBeenCalled();
```

```
});
```

```
PASS src/__tests__/UserLogin.test.jsx
  ✓ renders login form correctly (76 ms)
  ✓ handles successful email/password login (21 ms)
  ✓ handles failed email/password login due to invalid credentials (25 ms)
  ✓ handles successful Google login (11 ms)
  ✓ handles failed Google login (15 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        1.131 s
```

Figure 2: UserLogin Test Result

2 Percentage of Code Tested

We've tested approximately 35% of our code with these unit tests. We are planning to add a few more unit tests like testing the Navigation Bar and ProfilePage. This will cover an additional 30% of our code.

3 Integration Test

AddBookPage Component

Test 1: "Admin can add a new book."
We checked that an admin can successfully add a new book with the form and that the book is added to Firestore.

```
test('admin can add a new book to Firestore', async () => {
    render(<AddBookPage />);

    // Fill in the form fields using getByPlaceholderText
    fireEvent.change(screen.getByLabelText(/Title/i), {
      target: { value: 'Test Book Title' },
    });
    fireEvent.change(screen.getByLabelText(/Author/i), {
      target: { value: 'Test Author' },
    });
    fireEvent.change(screen.getByLabelText(/ISBN Number/i), {
      target: { value: '1234567890' },
    });
    fireEvent.change(screen.getByLabelText(/Description/i), {
      target: { value: 'This is a test description.' },
    });
    fireEvent.change(screen.getByLabelText(/Publication Date/i), {
      target: { value: '2023-10-10' },
    });
    fireEvent.change(screen.getByLabelText(/Number of Pages/i), {
```

```
    target: { value: '250' },
  });
  fireEvent.change(screen.getByLabelText(/Genres/i), {
    target: { value: 'Fiction, Adventure' },
  });

  // Mock addDoc to resolve successfully
  addDoc.mockResolvedValue({
    id: 'abc123',
  });

  // Mock getDocs to return the added book
  getDocs.mockResolvedValue({
    forEach: (callback) => {
      callback({
        id: 'abc123',
        data: () => ({
          title: 'Test Book Title',
          author: 'Test Author',
          isbn: '1234567890',
          description: 'This is a test description.',
          publicationDate: '2023-10-10',
          numberOfPages: 250,
          genres: ['Fiction', 'Adventure'],
        }),
      });
    },
  });

  // Submit the form
  fireEvent.click(screen.getByRole('button', { name: /Add Book/i })
);

  // Wait for the success message to appear
  await waitFor(() => {
    expect(screen.getByText('Book added successfully!')).toBeTruthy
();
  });

  // Verify that addDoc was called with correct data
  expect(addDoc).toHaveBeenCalledWith(
    collection(db, 'books'),
    {
      title: 'Test Book Title',
      author: 'Test Author',
      isbn: '1234567890',
      description: 'This is a test description.',
      publicationDate: '2023-10-10',
      numberOfPages: 250,
      genres: ['Fiction', 'Adventure'],
    }
```

```
    );

    // Verify that the book was added to Firestore
    const addedBook = await getBookByISBN('1234567890');
    expect(addedBook).not.toBeNull();
    expect(addedBook.title).toBe('Test Book Title');
    expect(addedBook.author).toBe('Test Author');
    expect(addedBook.isbn).toBe('1234567890');
    expect(addedBook.description).toBe('This is a test description.')
   ;
    expect(addedBook.publicationDate).toBe('2023-10-10');
    expect(addedBook.numberOfPages).toBe(250);
    expect(addedBook.genres).toEqual(['Fiction', 'Adventure']);
});
```

Test 2: "Shows error when required fields are missing."
This test ensures an error message is shown if the admin tries to submit the form without filling in all required fields.

```
test('shows error message when required fields are missing', async ()
    => {
    render(<AddBookPage />);

    // Attempt to submit the form without filling in any fields
    fireEvent.click(screen.getByRole('button', { name: /Add Book/i })
   );

    // Wait for the error message to appear
    await waitFor(() => {
      expect(screen.getByText('All fields are required')).toBeTruthy
   ();
    });
});
```

Test 3: "Handles Firestore failure gracefully."
We simulated a failure when trying to add a book to Firestore and checked if the error message appeared correctly.

```
test('handles Firestore addDoc failure gracefully', async () => {
    const consoleErrorSpy = jest.spyOn(console, 'error').
   mockImplementation(() => {}); // Mock console.error

    jest.spyOn(require('firebase/firestore'), 'addDoc').
   mockImplementation(() => {
      throw new Error('Firestore addDoc failed');
    });

    render(<AddBookPage />);

    fireEvent.change(screen.getByLabelText(/Title/i), {
      target: { value: 'Test Book Title' },
    });
    fireEvent.change(screen.getByLabelText(/Author/i), {
      target: { value: 'Test Author' },
```

```
    });
    fireEvent.change(screen.getByLabelText(/ISBN Number/i), {
      target: { value: '0987654321' },
    });
    fireEvent.change(screen.getByLabelText(/Description/i), {
      target: { value: 'This is another test description.' },
    });
    fireEvent.change(screen.getByLabelText(/Publication Date/i), {
      target: { value: '2023-11-11' },
    });
    fireEvent.change(screen.getByLabelText(/Number of Pages/i), {
      target: { value: '300' },
    });
    fireEvent.change(screen.getByLabelText(/Genres/i), {
      target: { value: 'Non-Fiction, Science' },
    });

    fireEvent.click(screen.getByRole('button', { name: /Add Book/i })
   );

    await waitFor(() => {
      expect(screen.getByText('Failed to add book. Please try again
   .')).toBeTruthy();
    });

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

```
PASS src/__tests__/AddBookPage.test.jsx
  AddBookPage Integration Test
    ✓ admin can add a new book to Firestore (104 ms)
    ✓ shows error message when required fields are missing (16 ms)
    ✓ handles Firestore addDoc failure gracefully (23 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        1.596 s
```

Figure 3: AddBook Integration Test Result

4 Acceptance Testing

- **Login Form Rendering:**
  This ensures that the login page shows all required fields (email, password) and buttons (login, Google sign-in). If any of them are missing, the user can't proceed.
- **Registration Form Rendering:**
  When the admin chooses to register, the system shows the registration form with the additional "Admin Key" field. This field ensures that only authorized users can register.

- **Successful Logins and Registrations:**
  The system checks user and admin credentials. Once valid credentials are provided, users and admins are logged in and redirected to their dashboard. For admins, a correct admin key is also required to register.
- **Handling Invalid Credentials:**
  The system shows clear error messages for failed login attempts (e.g., wrong email/password or invalid Google sign-in). It also shows an "Invalid admin key" message if the wrong key is entered during admin registration.
- **Adding a Book:**
  Admins can fill out a form to add books to the Firestore database. If successful, the system confirms with a "Book added successfully!" message. If fields are missing, it shows "All fields are required."
- **Handling Firestore Errors:**
  If there's an issue adding a book to Firestore (e.g., `addDoc` fails), the system informs the admin with "Failed to add a book. Please try again."

5 Bugs and Demonstration

During the development and testing phase, we identified several bugs, a few bugs are fixed and a few bugs need to be fixed. We are currently spending at least 15 - 20 hours every week to fix the bugs raised which include Firebase integration, UI-related, and Firestore Rules-related bugs. We are working hard to resolve all the bugs to deliver a bug-free demo.