PAU Bookit - Login Page Technical Documentation

Introduction

This document provides technical specifications and implementation details for the PAU Bookit application's login page. The login page has been refactored from a vanilla JavaScript implementation to a React-based solution with a focus on security, scalability, and responsive design to handle 2,000-5,000 users.

Project Structure

```
pau-bookit/
├── public/
│   ├── index.html
│   ├── favicon.ico
│   └── assets/
│       └── logo.jpeg
├── src/
│   ├── App.jsx
│   ├── index.jsx
│   ├── assets/
│   │   └── logo.jpeg
│   ├── components/
│   │   └── common/
│   │       ├── Spinner.jsx
│   │       └── Alert.jsx
│   ├── contexts/
│   │   └── AuthContext.jsx
│   ├── hooks/
│   │   └── useForm.jsx
│   ├── pages/
│   │   ├── auth/
│   │   │   ├── Login.jsx
│   │   │   ├── Login.css
│   │   │   ├── Login.test.jsx
│   │   │   └── ForgotPassword.jsx
│   │   ├── dashboard/
│   │   │   ├── UserDashboard.jsx
│   │   │   └── AdminDashboard.jsx
│   │   └── NotFound.jsx
│   ├── services/
│   │   ├── api.js
│   │   └── auth.service.js
│   ├── utils/
│   │   ├── constants.js
│   │   ├── validators.js
│   │   └── tokenHelpers.js
│   └── styles/
│       └── global.css
├── .env
├── .env.example
├── .gitignore
├── package.json
├── README.md
└── jest.config.js
```

Technology Stack

- **Frontend Framework**: React 18
- **HTTP Client**: Axios
- **Routing**: React Router v6
- **State Management**: React Context API
- **Authentication**: JWT (JSON Web Tokens)
- **Testing**: Jest + React Testing Library
- **Backend**: NestJS (API endpoints)
- **Database**: PostgreSQL

Login Page Architecture

Components

1. **Login.jsx**: Main login page component
   - Handles form state and submission
   - Validates user inputs
   - Communicates with backend API
   - Provides user feedback for errors and success
2. **AuthContext.jsx**: Authentication context provider
   - Manages authentication state across the application
   - Handles token storage, refresh, and validation
   - Provides login/logout functionality
   - Ensures authenticated routes are protected

Authentication Flow

1. **Initial Load**:
   - Check for existing authentication token
   - Verify token validity with backend
   - Redirect to appropriate dashboard if already authenticated
2. **Login Process**:
   - Validate email format (must be a PAU email address)
   - Submit credentials to backend API
   - Store JWT token (temporarily in localStorage, will be enhanced for production)
   - Update application authentication state

  - Redirect user to appropriate dashboard based on role
3. **Token Management**:
   - Automatic token refresh before expiration
   - Token validation on protected routes
   - Secure token storage

Security Considerations

1. **Authentication Token**:
   - Current implementation uses localStorage for simplicity during development
   - For production, will be enhanced with:
     - HttpOnly cookies for token storage
     - CSRF protection mechanisms
     - Short-lived access tokens with refresh token rotation
2. **Input Validation**:
   - Client-side validation for immediate feedback
   - Server-side validation to prevent malicious inputs
   - PAU email domain verification
3. **API Security**:
   - HTTPS-only communication
   - Rate limiting to prevent brute force attacks
   - Token-based authentication for all protected endpoints
4. **Error Handling**:
   - Generic error messages to users (not exposing system details)
   - Detailed error logging on the server side
   - Graceful degradation for network failures

Responsive Design

The login page is designed to be fully responsive across all devices with a minimum width of 250px. The stylesheet uses:

1. **Mobile-First Approach**:
   - Base styles designed for smallest screens
   - Progressive enhancement for larger screens
2. **Breakpoints**:
   - Ultra small: 250px-320px

- Extra small: 320px-360px
- Small: 360px-480px
- Medium: 480px-600px
- Tablet: 600px-768px
- Desktop: 768px+

3. **Flexible Layout**:
   - Uses CSS Flexbox for adaptable layouts
   - Percentage-based widths
   - Fluid typography with rem units

Testing Strategy

The login functionality is tested using Jest and React Testing Library with the following test cases:

1. **Rendering Tests**:
   - Verify that all UI elements are rendered correctly
   - Check accessibility of form elements
2. **Validation Tests**:
   - Email format validation (PAU domain)
   - Required field validation
   - Error message display
3. **Authentication Tests**:
   - Successful login flow for different user roles
   - Failed login scenarios (wrong credentials, network errors)
   - Token persistence and validation
4. **Integration Tests**:
   - Form submission and API interaction
   - Redirect behavior after authentication
   - Token refresh mechanism

API Integration

The login page interacts with the following backend endpoints:

1. **POST /auth/login**
   - Request: `{ email: string, password: string }`
   - Response: `{ token: string, user: UserObject }`

2. **GET /auth/verify**
   - Headers: `{ Authorization: "Bearer {token}" }`
   - Response: `{ isValid: boolean, role: string }`
3. **POST /auth/refresh**
   - Headers: `{ Authorization: "Bearer {token}" }`
   - Response: `{ token: string }`
4. **GET /auth/profile**
   - Headers: `{ Authorization: "Bearer {token}" }`
   - Response: User profile details
5. **POST /auth/logout**
   - Headers: `{ Authorization: "Bearer {token}" }`
   - Response: Success status

Default User Configuration

When the backend database has no users, the system will create default users:

1. **Admin User**:
   - Email: `elvis.ebenuwah@pau.edu.ng`
   - Password: `Admin123`
   - Role: `admin`
2. **New Student Users**:
   - Default password: `User123`
   - Role: `student`
   - Auto-generated ID in format: `STU{5-digit number}`

How to Test

Unit and Integration Testing

Run the automated test suite:

```bash
# Install dependencies
npm install

# Run tests
npm test

# Run tests with coverage report
npm test -- --coverage
```

Manual Testing

1. **Development Environment Setup**:

   ```bash
   # Clone repository
   git clone [repository-url]
   cd pau-bookit

   # Install dependencies
   npm install

   # Create .env file from example
   cp .env.example .env

   # Update .env with API endpoint
   # REACT_APP_API_URL=http://localhost:3001/api

   # Start development server
   npm start
   ```

2. **Testing Scenarios**:
   - Try logging in with non-PAU email (should show validation error)
   - Try logging in with incorrect password (should show authentication error)
   - Test admin login flow (should redirect to admin dashboard)
   - Test student login flow (should redirect to student dashboard)
   - Test device responsiveness (resize browser window)
   - Test network errors (disable internet connection)

Deployment Considerations

For production deployment, consider the following enhancements:

1. **Security Hardening**:

   - Move from localStorage to HttpOnly cookies for token storage
   - Implement CSRF protection
   - Enable Content Security Policy headers
   - Set up proper CORS configuration

2. **Performance Optimization**:
   - Code splitting and lazy loading
   - Asset optimization (image compression, minification)
   - Implement caching strategies

3. **Monitoring and Logging**:
   - Add error tracking (Sentry, LogRocket)
   - Implement analytics for user behavior tracking
   - Set up performance monitoring

Future Enhancements

1. **Multi-Factor Authentication**:
   - Add optional 2FA for admin users
   - Email verification for new accounts
2. **Password Management**:
   - Complete "Forgot Password" functionality
   - Password strength requirements
   - Password change policies
3. **User Experience**:
   - Remember me functionality
   - Login history tracking
   - Device management for users
4. **Accessibility**:
   - ARIA attributes for screen readers
   - Keyboard navigation improvements
   - Color contrast compliance