Login System - Project Structure

Generated on: January 22, 2026

This document provides a comprehensive overview of the Login System project structure. The project consists of a React frontend and Node.js/Express backend, organized in a clean, modular architecture.

Complete Project Structure

```
login-system/
│
├── backend/
│   ├── server.js              # Main Express server with login API
│   ├── package.json           # Backend dependencies and scripts
│   ├── .env                   # Environment variables (PORT, NODE_ENV)
│   └── .gitignore             # Git ignore patterns
│
├── frontend/
│   ├── public/                # Public static assets
│   ├── src/
│   │   ├── components/
│   │   │   └── Login.jsx      # Login component with form validation
│   │   ├── services/
│   │   │   └── api.js         # Axios API service for backend calls
│   │   ├── styles/
│   │   │   ├── App.css        # Global application styles
│   │   │   └── Login.css      # Login component specific styles
│   │   ├── App.jsx            # Main App component
│   │   └── main.jsx           # Application entry point
│   ├── index.html             # HTML template
│   ├── vite.config.js         # Vite build configuration
│   ├── package.json           # Frontend dependencies and scripts
│   ├── .env                   # Environment variables (API URL)
│   └── .gitignore             # Git ignore patterns
│
├── README.md                  # Complete project documentation
└── .gitignore                 # Root level git ignore
```

Backend Structure Details

The backend is built with Node.js and Express, providing a RESTful API for authentication.

```
backend/
│
├── server.js
│   ├── Express application setup
│   ├── CORS and body-parser middleware
│   ├── POST /api/login endpoint
│   │   ├── Email and password validation
│   │   ├── Email format validation with regex
│   │   ├── Hardcoded credentials check
│   │   └── Success/error response handling
│   ├── GET /api/health endpoint
│   └── 404 error handler
│
├── package.json
│   ├── Dependencies:
│   │   ├── express: ^4.18.2
│   │   ├── cors: ^2.8.5
│   │   ├── body-parser: ^1.20.2
│   │   └── dotenv: ^16.3.1
│   ├── DevDependencies:
│   │   └── nodemon: ^3.0.1
│   └── Scripts:
│       ├── start: node server.js
│       └── dev: nodemon server.js
│
└── .env
    ├── PORT=5000
    └── NODE_ENV=development
```

Frontend Structure Details

The frontend is built with React 18 and Vite, providing a modern, fast development experience with hot module replacement.

```
frontend/
│
├── src/
│   ├── components/
│   │   └── Login.jsx
│   │       ├── Form state management with useState
│   │       ├── Email and password validation
│   │       ├── Real-time error display
│   │       ├── Success/error message handling
│   │       ├── Loading state during API calls
│   │       └── Disabled inputs during submission
│   │
│   ├── services/
│   │   └── api.js
│   │       ├── Axios instance configuration
│   │       ├── loginUser() function
│   │       ├── checkHealth() function
│   │       └── Error handling wrapper
│   │
│   ├── styles/
│   │   ├── App.css
│   │   │   ├── Global CSS reset
│   │   │   ├── Base typography
│   │   │   └── Layout utilities
│   │   │
│   │   └── Login.css
│   │       ├── Login container with gradient background
│   │       ├── Login card with shadow and animations
│   │       ├── Form input styles with focus states
│   │       ├── Error and success message styles
│   │       ├── Button hover and active states
│   │       └── Responsive media queries
│   │
│   ├── App.jsx
│   │   └── Main application component
│   │
│   └── main.jsx
│       └── React DOM rendering with StrictMode
│
├── package.json
│   ├── Dependencies:
│   │   ├── react: ^18.2.0
│   │   ├── react-dom: ^18.2.0
│   │   └── axios: ^1.6.2
│   ├── DevDependencies:
│   │   ├── @vitejs/plugin-react: ^4.2.1
│   │   └── vite: ^5.0.8
│   └── Scripts:
│       ├── dev: vite
│       ├── build: vite build
│       └── preview: vite preview
│
├── vite.config.js
│   ├── React plugin configuration
│   └── Dev server with proxy to backend
│
└── .env
    └── VITE_API_BASE_URL=http://localhost:5000/api
```

API Endpoints

1. POST /api/login
   Purpose: Authenticate user with email and password
   Request Body:
   ```
   {
     "email": "admin@example.com",
     "password": "Admin@123"
   }
   ```

   Success Response (200):
   ```
   {
     "success": true,
     "message": "Login successful",
     "data": {
       "email": "admin@example.com",
       "token": "dummy-jwt-token-xxxxx"
     }
   }
   ```

   Error Response (401):
   ```
   {
     "success": false,
     "message": "Invalid email or password"
   }
   ```

2. GET /api/health
   Purpose: Check server health status
   Response (200):
   ```
   {
     "success": true,
     "message": "Server is running",
     "timestamp": "2024-01-22T10:30:00.000Z"
   }
   ```

Key Features

Frontend Features:
• Clean and responsive login interface with gradient design
• Form validation (required fields, email format)
• Real-time error handling and display
• Success/Error message notifications
• Loading states during API calls
• Axios for API communication
• Modern CSS with smooth animations and transitions
• Mobile-responsive design

Backend Features:
• RESTful API design
• Input validation and sanitization
• Email format validation with regex
• Comprehensive error handling
• CORS enabled for cross-origin requests
• Environment variable configuration
• Health check endpoint
• Structured JSON responses

Development Features:
• Hot Module Replacement (HMR) with Vite
• Auto-restart backend with nodemon
• Proxy configuration for seamless API calls
• Clean folder structure and separation of concerns
• ESLint ready configuration
• Production-ready build scripts

Quick Setup Instructions

Backend Setup:
1. cd backend
2. npm install
3. npm start (or npm run dev for development)

Frontend Setup:
1. cd frontend
2. npm install
3. npm run dev

Access:
• Frontend: http://localhost:3000
• Backend: http://localhost:5000

Demo Credentials:
• Email: admin@example.com
• Password: Admin@123

Note: *This is a demonstration project with hardcoded credentials. For production use, implement proper authentication, password hashing, database integration, and JWT token management.*