



SOFTWARE DESIGN SPECIFICATION

FULL STACK DEVELOPMENT  
  
CONNECTIN WEB APPLICATION

|  |  |  |  |
| --- | --- | --- | --- |
| **Created By:** | Shreyash Deogirkar | **Approved By:** | <Domain Lead Name> |
| **Created On:** | 22-07-2024 | **Approved On:** | DD-MMM-YYYY |

Page left blank intentionally

**INDEX**

[**1** **PURPOSE** 2](#_Toc142418236)

[**2** **PROJECT SCOPE** 2](#_Toc142418237)

[**3** **SYSTEM OVERVIEW** 2](#_Toc142418238)

[**4** **DESIGN CONSIDERATIONS** 2](#_Toc142418239)

[4.1 Requirements 3](#_Toc142418240)

[4.2 Assumptions 3](#_Toc142418241)

[4.3 Dependencies 3](#_Toc142418242)

[**5** **SYSTEM ARCHITECTURE** 3](#_Toc142418243)

[5.1 Architectural Strategies 4](#_Toc142418244)

[5.2 Structure & Relationships 4](#_Toc142418245)

[**6** **DETAILED DESCRIPTION OF COMPONENTS** 4](#_Toc142418246)

[**7** **INTEGRATION** 5](#_Toc142418247)

[**8** **APPENDICES** 1](#_Toc142418248)

[8.1 Appendix A – Detailed Description of Components 1](#_Toc142418249)

**General Instructions for using the Live Project POC Document**

* This template and the subsequent document created using this template is a confidential document and is the intellectual property of Cloud Counselage Pvt. Ltd. Circulating it outside of the organisation without the consent of Cloud Counselage Pvt. Ltd. is the breach of company policies and will lead to legal actions
* The Design Specification of a software forms the basis of development of software
* The **text between inequality (< >) is to be replaced** by relevant text
* Please **remove the yellow highlight on the Text** between the inequality (< >). This is done to help you notice the text to be changed/replaced
* The text in *italics* highlighted in grey is just for reference and should be removed after adding the relevant text

# **PURPOSE**

This document is created based on the requirement specification document. The purpose of this Software Design Specification (SDS) Document is to break down the project into components to describe in detail what the purpose of each component is and how it will be implemented. The SDS will also serve as a tool for verification and validation of the final product.

# **PROJECT SCOPE**

The scope of the **Connectin web application** includes its distinct features, its benefits, and its limitations. The system's distinct features allow users to connect professionally and learn the required skills required to succeed in their career by using the functions in the application. The users can authenticate and start using the website, they can post, comment, add skills etc The system enables the user to develop themselves professionally.

# **SYSTEM OVERVIEW**

This section will provide an outline of the various components and subsystems of Connectin application

**View**

Sets up and displays the user interface elements such as the login/register forms, sidebar with profile, feed section, and navbar.

Shows the user's profile details in the sidebar.

Displays posts in the feed section, allowing users to add, like, and comment on them.

**Data Access (DA)**

Obtains data from the Model, such as user profiles and post content.

Converts data into database operations.

Interacts with the database API to execute operations, such as storing user data and retrieving posts.

**Model**

Gathers data from the Controller, such as new user information and post interactions.

Organizes data for Data Access or Notification Access Modules.

Receives processed data from DA and NA modules.

Formats data and sends it back to the Controller.

**Controller**

Manages actions triggered by the user interface, such as logging in, registering, liking posts, and commenting.

* Updates the user interface with information obtained from the Model.

**Notification Access (NA)**

Receives data from the Model.

Converts data into notification commands.

Notifies the Model about the success or failure of the notification operation.

# **DESIGN CONSIDERATIONS**

This section describes requirements, assumptions and dependencies to be addressed to devise a complete design solution.

## Requirements

* VS Code
* React
* NextJs
* NodeJs
* MongoDB
* Vercel

## Assumptions

**User Authentication and Management**

* **Authentication**: Clerk is used for user authentication and management. It assumes that users are authenticated and their data is securely managed.
* **User Data**: User data (like skills, experience, profile details) is stored and managed in a MongoDB database.

**API Routes**

* **CRUD Operations**: API routes handle CRUD operations for posts and comments. The routes assume proper validation and error handling are in place.
* **Data Exchange**: The frontend communicates with the backend using JSON for data exchange.

## Dependencies

* React
* React-dom
* Next
* Tailwindcss
* Postcss
* Axios
* Clerk
* Node.js
* Mongoose
* Dotenv .etc

# **SYSTEM ARCHITECTURE**

The software system architecture refers to the logical organization of a distributed system into software components. It defines how components of a software system are assembled, their relationship and communication between them. It serves as a blueprint for software application and development basis for developer team. An effective architecture serves as the conceptual glue that holds every phase of the project together for all of its stakeholders, enabling agility, time and cost savings, and early identification of design risks.

The Software architecture:

* Defines structure of a system
* Defines behaviour of a system
* Defines component relationship
* Defines communication structure
* Balances stakeholder’s needs
* Influences team structure
* Focuses on significant elements
* Captures early design decisions

Below some important characteristics which are commonly considered are explained.

**Operational Architecture Characteristics:**

* Availability
* Performance
* Reliability
* Low fault tolerance
* Scalability

**Structural Architecture Characteristics:**

* Configurability
* Extensibility
* Supportability
* Portability
* Maintainability

**Cross-Cutting Architecture Characteristics:**

* Accessibility
* Security
* Usability
* Privacy
* Feasibility

## Architectural Strategies

* Frontend

-UI components

-Pages

-Client-side state

* API Routes

-CRUD Operations

-Authentication

-File Storage

* Database

-MongoDB

-Mongoose Scheme

* Authentication

-Authorization clerk

* Deployment

-Vercel

## Structure & Relationships

**Main Menu Screen**

-Home page

-Navigation to Login, Register .etc.

**Login Screen**

- Authenticates Users - Redirects on Success

- Shows Errors on Fail

**Issue validation**

- Displays Errors

- Handles Validation

**Register Screen**

- Creates New Account

- Redirects on Success

- Displays Errors

- Shows Errors on Fail

**Profile screen**

- Displays User Info

- Updates Profile

- Shows Confirmation on Success

**Issue violation screen**

- Displays Errors

**Logout**

- Ends User Session

- Displays Errors

- Redirects to Login or Home Page

**Confirmation screen**

- Shows Status of Actions

# **DETAILED DESCRIPTION OF COMPONENTS**

For detailed description of the components, please refer **Appendix A – Detailed Description of Components**

|  |  |
| --- | --- |
| **Identification** | Login Screen/ Register Screen  Location: Authentication module |
| **Type** | Form/User Interface |
| **Purpose** | Authenticates users by validating credentials and manages redirection post-login.  Allows users to create new accounts and handles account creation. |
| **Subordinates** | Login form with fields for username and password  Registration form with fields for user details |
| **Dependencies** | Depends on authentication service for credential validation.  Redirects to Profile Screen or shows error messages (Issue Violation Screen Part I) if login fails.  Redirects to Profile Screen or shows errors using Issue Violation Screen Part I if registration fails. |
| **Interfaces** | * Login API endpoint * Error messages for failed authentication * Registration API endpoint * Error messages for failed registration |
| **Resources** | Authentication API service  Registration API service |
| **Processing** | Validates user credentials and redirects to the Profile Screen on success.  Handles new user account creation and redirects to Profile Screen on success. |
| **Data** | User credentials (username, email, password) |

|  |  |
| --- | --- |
| **Identification** | Main Menu Screen  Location: Main entry point of the application |
| **Type** | User Interface |
| **Purpose** | Provides navigation to various parts of the application, such as the profile page, feed, and other functionalities. |
| **Subordinates** | Navbar with links to Home, Profile, Feed, and Settings  Sidebar with quick links to posts, skills, and experience |
| **Dependencies** | Relies on the user authentication state to display the correct navigation options.  Other components: Profile Screen, Feed Page, Settings Page |
| **Interfaces** | User interface elements (buttons, links)  No explicit error messages; errors are handled by the respective pages. |
| **Resources** | Frontend frameworks/libraries (React, Next.js) |
| **Processing** | Displays navigation options based on user state. |
| **Data** | User authentication state  Navigation settings |

|  |  |
| --- | --- |
| **Identification** | Logout  Location: Authentication module |
| **Type** | Control Procedure |
| **Purpose** | Ends the user session and redirects to the Login Screen or Home Page. |
| **Subordinates** | Session termination procedure |
| **Dependencies** | Requires user authentication to terminate the session. |
| **Interfaces** | Session management |
| **Resources** | Session management service |
| **Processing** | Logs out the user and handles redirection. |
| **Data** | Session data |

# **INTEGRATIONS**

ConnectIn will interface with Clerk for authentication and Cloudinary for cloud storage. The integration details are as follows:

• **Authentication with Clerk**: ConnectIn uses Clerk to manage user authentication. Clerk provides a seamless integration with Next.js, allowing the application to handle user registration, login, and session management securely. Clerk SDK is used to add authentication functionalities and integrate with the frontend. Clerk’s API ensures secure communication and management of user data.

• **Cloud Storage with Cloudinary**: ConnectIn utilizes Cloudinary for storing and managing media files such as profile photos and post images. Cloudinary SDK is integrated with the application to facilitate uploading, retrieving, and transforming media files. The integration allows the frontend to directly interact with Cloudinary for efficient media handling.

• **Database Integration**: ConnectIn employs MongoDB for data storage, utilizing Mongoose for object data modeling (ODM). This integration allows the application to store and manage user data, posts, comments, skills, and experiences. Mongoose schemas define the structure of the data, ensuring consistency and validation.

• **Frontend Framework**: ConnectIn uses Next.js for server-side rendering and React for building the user interface. Next.js facilitates API routes and server-side logic, ensuring efficient data fetching and rendering. React hooks and components manage the state and behavior of the application’s UI.

• **Deployment and Hosting**: Vercel is used for deploying and hosting ConnectIn. Vercel provides seamless integration with Next.js, enabling continuous deployment and efficient scaling of the application.

# **APPENDICES**

## Appendix A – Detailed Description of Components

|  |  |
| --- | --- |
| **Identification** | Login Screen/ Register Screen  Location: Authentication module |
| **Type** | Form/User Interface |
| **Purpose** | Authenticates users by validating credentials and manages redirection post-login.  Allows users to create new accounts and handles account creation. |
| **Subordinates** | Login form with fields for username and password  Registration form with fields for user details |
| **Dependencies** | Depends on authentication service for credential validation.  Redirects to Profile Screen or shows error messages (Issue Violation Screen Part I) if login fails.  Redirects to Profile Screen or shows errors using Issue Violation Screen Part I if registration fails. |
| **Interfaces** | * Login API endpoint * Error messages for failed authentication * Registration API endpoint * Error messages for failed registration |
| **Resources** | Authentication API service  Registration API service |
| **Processing** | Validates user credentials and redirects to the Profile Screen on success.  Handles new user account creation and redirects to Profile Screen on success. |
| **Data** | User credentials (username, email, password) |

|  |  |
| --- | --- |
| **Identification** | Main Menu Screen  Location: Main entry point of the application |
| **Type** | User Interface |
| **Purpose** | Provides navigation to various parts of the application, such as the profile page, feed, and other functionalities. |
| **Subordinates** | Navbar with links to Home, Profile, Feed, and Settings  Sidebar with quick links to posts, skills, and experience |
| **Dependencies** | Relies on the user authentication state to display the correct navigation options.  Other components: Profile Screen, Feed Page, Settings Page |
| **Interfaces** | User interface elements (buttons, links)  No explicit error messages; errors are handled by the respective pages. |
| **Resources** | Frontend frameworks/libraries (React, Next.js) |
| **Processing** | Displays navigation options based on user state. |
| **Data** | User authentication state  Navigation settings |

|  |  |
| --- | --- |
| **Identification** | Logout  Location: Authentication module |
| **Type** | Control Procedure |
| **Purpose** | Ends the user session and redirects to the Login Screen or Home Page. |
| **Subordinates** | Session termination procedure |
| **Dependencies** | Requires user authentication to terminate the session. |
| **Interfaces** | Session management |
| **Resources** | Session management service |
| **Processing** | Logs out the user and handles redirection. |
| **Data** | Session data |