# SSI STUDIOS Technical Documentation(with admin panel): ssicrs:[https://ssicrsweb.vercel.app/Register](https://ssicrsweb.vercel.app) ssicrs-adminpanel:<https://ssicrsadmin.vercel.app> (username: ssi , password: ssi) Figma Flow: [https://www.figma.com/board/MSXbqGr0dAwlVU9HzNp66F/User-Flow-Diagram-for-FigJam--Community-?node-id…](https://www.figma.com/board/MSXbqGr0dAwlVU9HzNp66F/User-Flow-Diagram-for-FigJam--Community-?node-id%E2%80%A6)

**Project Scope:** SSI CRS (Surgical Systems Innovation - Clinical Registration System) Admin Dashboard and Public Marketing Pages (Home, Programs, Registration).

**By :Puneet Shukla**

**Date:** 11 October 2025

## 1. Technology Stack and Architecture Overview

The SSI CRS application is built on a modern, robust, and scalable Next.js (React) and MongoDB architecture. This approach emphasizes developer efficiency, system security, and fast, reliable performance for end-users.

|  |  |  |
| --- | --- | --- |
| **Component** | **Technology** | **Strategic Value & Key Benefit** |
| **Full-Stack Framework** | **Next.js (App Router)** | **Performance & SEO:** Enables **Server-Side Rendering (SSR)** for fast initial loading of public marketing pages (crucial for SEO) and a seamless **Single-Page Application (SPA)** experience for the authenticated dashboard. Unified environment drastically improves developer productivity. |
| **Frontend/UI** | **React** | **Maintainability & Scalability:** Adopts a component-based, modular structure that reduces codebase complexity, accelerates feature development, and ensures strict UI consistency across all pages. |
| **Styling** | **Tailwind CSS** | **Speed & Consistency:** Utility-first methodology accelerates design iteration and guarantees high responsiveness across all devices (mobile-first approach) with minimal custom CSS overhead. |
| **State & Interactivity** | **React Hooks, Framer Motion** | **Modern UX:** Ensures fluid, smooth, and native-feeling interactions (e.g., sidebar motion, table sorting animations) while maintaining predictable and performant state management for complex dashboard features. |
| **Backend/API** | **Next.js API Routes** | **Security & Efficiency:** Serverless-friendly backend handles data validation, security checks, and authentication outside the client browser environment, simplifying deployment. |
| **Database/Storage** | **MongoDB (Mongoose) / GridFS** | **Flexibility & Reliability:** Non-relational structure allows rapid schema evolution. Mongoose provides robust validation. Dedicated **GridFS** handles large binary file storage (certificates/credentials) reliably and efficiently. |

## 2. Frontend Implementation: Public Pages Deep Dive

The public-facing components focus on user acquisition and optimal front-end performance, essential for marketing visibility and form completion rates.

### 2.1 Public Pages (Home, Programs, Registration) Implementation Details

|  |  |  |
| --- | --- | --- |
| **Page/Section** | **Key Implementation Details** | **Performance & User Experience Goal** |
| **Home Page** | **Rendering Strategy:** Primarily rendered via **SSR** (Server-Side Rendering) or Static Generation using Next.js. **Styling:** Heavy reliance on dynamic Tailwind utility classes (e.g., lg:w-1/2, sm:flex-col) ensuring the layout instantly adapts across all device breakpoints. | Maximize load speed (Core Web Vitals) and ensure excellent SEO ranking for brand visibility. |
| **Programs Page** | **Data Visualization:** Uses React to map program data into easily digestible cards. **Modals:** Implements simple conditional rendering via useState for displaying deep program detail pop-ups without navigating away. | Provides clear, hierarchical information access, reducing cognitive load for prospective participants. |
| **Registration Page** | **Data Accuracy:** Implements **Controlled Form Inputs** (state-driven inputs) to maintain real-time data synchronization. **Validation:** Uses immediate **client-side validation** (e.g., regex for email format, minimum length) to provide instant feedback and correct user errors before hitting the server. | Achieves high data integrity and low form drop-off rates by guiding users through the necessary inputs smoothly. |
| **File Upload Flow** | **Decoupled API Call:** The client separates the file submission. It prepares the file as FormData and sends it to the lightweight /api/uploads endpoint, receiving a unique uploadId in return. | Ensures robust handling of potentially large credential documents and keeps the primary registration API clean and responsive. |

## 3. Backend Architecture and API Routes

The server layer ensures data persistence and robust security controls via dedicated Next.js API Routes.

### 3.1 Data Flow and Database Structure

|  |  |  |
| --- | --- | --- |
| **Model/Endpoint** | **Purpose** | **Key Implementation** |
| **Registration Model** | Stores user registration details (name, email, programs, status). | Defined using **Mongoose Schema** for validation and consistent data types. Includes fields like status (upcoming, pending, completed) and references for uploaded files (uploadId). |
| **/api/registrations** | Handles **GET** (fetch all registrations) and **POST** (new registration submission). | **GET:** Implements server-side logic for dynamic querying, sorting, and filtering based on query parameters (e.g., ?search=X&status=pending). |
| **/api/registrations/[id]** | Handles **PUT** (update status/details) and **DELETE** (remove registration). | **PUT:** Uses findByIdAndUpdate with $set to update the document based on the full payload sent from the dashboard's edit modal. **DELETE:** Uses findByIdAndDelete. |
| **Typing Fix** | N/A | Dynamic route handlers use the consistent context: { params: Promise<{ id: string }> } signature with internal await context.params to resolve Next.js compilation |

## ssi-crs

## ├─ app → dashboard | login | managedatabase | \_not-found | globals.css

## ├─ api → admin-login | logout | me | registrations → [id] | send-certificate | files | uploads → [id]

## ├─ components → forms → LoginForm | RequestAccessForm

## │ → backgrounds → AuthBg

## │ → icons → CustomIcons

## │ → Layout | Logo

## ├─ contexts → AuthContext

## ├─ lib → mongodb | auth | uploadHelpers

## ├─ models → User | Registration | Upload

## 

## 4. SSI CRS Admin Panel: Authentication and Access Control

This section details the critical processes for accessing and managing the secure Admin Dashboard.

### 4.1 Step-by-Step Login Flow (/api/admin-login)

|  |  |  |
| --- | --- | --- |
| **Step** | **API Route Logic** | **Security Mechanism** |
| **1. Credential Submission** | Server receives credentials via req.json() and checks them against the hardcoded allowedUsers list. | Ensures only predefined administrators can initiate a session. |
| **2. Token Creation** | A **JWT (JSON Web Token)** is generated using jwt.sign(), including identity claims (username, role: 'admin'). | Token is signed using a secret key (process.env.JWT\_SECRET!) and set to expire in 1h (expiresIn: '1h'). |
| **3. Session Creation** | The server responds with a **Set-Cookie header** containing the signed token. | **Crucial Security:** The **HttpOnly** flag prevents client-side JavaScript access to the token, effectively blocking XSS attacks from hijacking the session. |
| **4. Dashboard Entry** | Client redirects to /dashboard. The browser automatically attaches the secure HttpOnly cookie to all subsequent requests. | Access to protected resources is verified server-side via middleware checking this cookie. |

### 4.2 Logout and Session Termination (/api/logout)

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Location** | **Action** | **Mechanism** |
| **1. Logout Request** | Sidebar.tsx triggers a POST request to /api/logout. | The handleLogout function ensures the action is non-disruptive. |  |
| **2. Cookie Deletion** | Server responds with a Set-Cookie header that overwrites the existing session cookie with parameters (Max-Age=0, Expires=past date) forcing browser deletion. | **Immediate Invalidation:** The session is instantly terminated on the browser side. |  |
| **3. Client Redirect** | Client executes router.push('/login'). | Guarantees the user returns to the public area without ability to access sensitive data. |  |

## 5. SSI CRS Admin Panel Core Feature Breakdown (For Management)

This table outlines the essential features and capabilities of the secure Admin Dashboard, detailing their purpose and scope.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature Area** | **Key Functionality** | **Business Value** | **Implementation Scope** |
| **Authentication & Access** | **Secure JWT Login/Logout** | Ensures only authorized administrators can access sensitive registration data, maintaining data integrity and compliance. | Login API (/api/admin-login) sets HttpOnly cookies; Logout API (/api/logout) instantly clears the session. |
| **Data Visibility (Read)** | **Real-Time Data Table** | Provides a single source of truth for all participant registrations, viewable in a responsive, fast-loading interface. | DashboardTable component fetching data from /api/registrations GET endpoint. |
| **Data Filtering & Search** | **Advanced Table Controls** | Allows staff to quickly locate specific records or groups of participants by Status, Name, Email, Profession, or custom Date Ranges. | Client-side filtering logic coupled with server-side data fetching based on URL query parameters. |
| **Registration Status Mgt.** | **Inline Status Update** | **Enables administrators to quickly track and update participant status** (Upcoming, Pending, Completed) directly within the table view. | Optimistic UI updates triggering targeted /api/registrations/[id] PUT requests. |
| **Ticket Number System** | **Sequential Registration ID** | Provides a guaranteed, simple reference number for quick administrative and communication purposes, often sequential based on MongoDB insertion order. | Managed primarily on the backend during the initial registration POST to ensure uniqueness and sequence. |
| **Detailed Editing (CRUD)** | **Modal-Based Editing** | Allows comprehensive modification of all registration fields (e.g., correcting contact info, updating program choices) in a structured modal form. | Controlled inputs using React state (useState(editing)) submitted via PUT request to registrations/[id]. |
| **File Management** | **Credential Viewing/Deletion** | Provides direct access to uploaded supporting documents (e.g., medical certifications) linked to the participant's record. | Secured hyperlink accessing /api/uploads/[id] GET endpoint (GridFS download stream). |
| **Certificate Sending** | **Email Generation & PDF Attachment** | Automates the process of sending personalized training certificates directly to the participant's email upon program completion. | Client generates PDF (in-memory Blob) which is sent to the /api/send-certificate API, which uses **SendGrid** to deliver the email. |
| **Data Export** | **Bulk & Single Record Export** | Facilitates regulatory compliance, reporting, and offline management by allowing export of data in common formats (PDF, Excel). | Dedicated Export Modal triggers handleExport logic (placeholder for actual file generation/API). |
| **UX & Reliability** | **Non-Blocking Notifications** | Replaces disruptive browser alerts with modern Toast components, improving workflow continuity during administrative tasks. | Custom React Toast component managed via component state. |



  
  
  
  
  
