



| **Created By:** | Prachi Jain | **Approved By:** | <Domain Lead Name> |
| --- | --- | --- | --- |
| **Created On:** | 19-08-2025 | **Approved On:** | DD-MMM-YYYY |



Page left blank intentionally

**INDEX**

[**1**](#_dd37xo3lgw8y) **PURPOSE** 2

[**2**](#_7rw9d9p32ua4) **PROJECT SCOPE** 2

[**3**](#_6d5f2p1pygkh) **SYSTEM OVERVIEW** 2

[**4**](#_i2tb25nzfqbq) **DESIGN CONSIDERATIONS** 2

[4.1](#_jppdooxvfqyf) Requirements 3

[4.2](#_ko1kujtux2qn) Assumptions 3

[4.3](#_9q3znxagyzzj) Dependencies 3

[**5**](#_hqc75pctmqz) **SYSTEM ARCHITECTURE** 3

[5.1](#_o6vs3nidam4t) Architectural Strategies 4

[5.2](#_g12k9er7lvko) Structure & Relationships 4

[**6**](#_wwfsvocd59mm) **DETAILED DESCRIPTION OF COMPONENTS** 4

[**7**](#_hbqffm4y7zu3) **INTEGRATION** 5

[**8**](#_l06jo8nxt8s9) **APPENDICES** 1

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

**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**

1. **Software Design Specification (SDS) Overview**

This document is created based on the requirement specification and outlines the **Software Design Specification (SDS)** for the project. The purpose of this SDS is to **break down the application into key components**, providing a detailed description of the **functionality, structure, and implementation** of each part. It serves as a **blueprint for development** and acts as a reference for **verification and validation** of the final product to ensure that it meets the specified requirements and user expectations.

# **PROJECT SCOPE**

1. **Scope of the Project – COLLEGE FINDER**

The **COLLEGE FINDER** web application aims to simplify access to reliable information about **IT and Management colleges** across India. Its scope encompasses outlining **key features**, **benefits**, and **limitations**.

Designed with a focus on usability, responsiveness, and clarity, the application provides a clean, ad-free, and login-free user interface. Users can easily search for colleges by **city or state** and access **detailed information**, including courses offered, contact details, website, and fee structure. The application was developed using a **modular, API-driven approach**.

1. **Tools and Inputs Used**

**Frontend Development:**

* **React JS:** Utilized for building dynamic, component-based user interfaces.
* **HTML & CSS:** Employed for structuring and styling the application.
* **JavaScript:** Provided core logic for data handling, user interaction, and API integration.

**Data & APIs:**

* **External/Public APIs:** Leveraged to fetch college-related data. Static files were used when APIs were unavailable.

**Development Tools & Practices:**

* **VS Code:** The primary code editor used throughout development.
* **Git & GitHub:** Employed for version control and project collaboration.
* **Browser DevTools:** Used for debugging and testing responsiveness.
* **Responsive Design Principles:** Applied to ensure compatibility across various devices (mobile, tablet, desktop).

# **SYSTEM OVERVIEW**

**COLLEGE FINDER: System Components Overview**

This document outlines the key components and subsystems of the **COLLEGE FINDER** application.

**1. Frontend (React.js)**

* Developed with **React.js**, utilizing a **component-based architecture** for efficient UI development.
* Offers an **interactive interface** enabling students to input preferences such as **location (city/state)**.
* Features a **responsive design** compatible with both desktop and mobile devices.
* Displays **dynamic college listings** based on user selections, presenting structured information (contact, website, fees, etc.).
* Designed for a **clean user experience**, free from ads or login requirements.

**2. Data Handling via API Integration**

* College data is retrieved using **external GEMINI APIs** or **curated static JSON files**.
* The system operates without a traditional backend or database; all data is managed client-side.
* React components dynamically fetch and render data, ensuring fast performance and real-time updates.

**3. Search & Filter System**

* Allows users to **filter colleges** based on location (city/state).
* Supports **real-time filtering** of college listings as users input criteria.
* Optimized for fast rendering and intuitive navigation, with future potential to incorporate additional criteria (e.g., stream, scores).

# **DESIGN CONSIDERATIONS**

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

## Requirements

<Add requirements as identified in the Software Requirement (SRS) document> The list of components

## Assumptions

<Add assumptions as listed in the Software Requirement (SRS) document> The list of components

## Dependencies

<Add assumptions as listed in the Software Requirement (SRS) document> The list of components

# **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

* User opens the app.
* User selects "enter location."
* User clicks "find colleges."
* The app calls gemini.searchCollege with the specified location.
* Gemini AI receives the prompt.
* Gemini AI returns a JSON list of colleges.
* The app parses and validates the data.
* **Fallback:** If the initial search is unsuccessful, a simpler Gemini prompt is used.
* The app parses the fallback data.
* The app displays colleges in the college results screen.
* The user can load more results.
* The user can view college details.

## Structure & Relationships

## 

# **DETAILED DESCRIPTION OF COMPONENTS**

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

The below template will be used to specify the details of all the components

***Table 1: Detailed Design Specification Template***

| **Identification** | The unique name for the component and the location of the component in the system. |
| --- | --- |
| **Type** | A module, a subprogram, a form, a data file, a control procedure, a class, etc. |
| **Purpose** | Function and performance requirements implemented by the design component, including derived requirements. Derived requirements are not explicitly stated in the SRS - but are implied or adjunct to formally stated SDS requirements. |
| **Subordinates** | The internal structure of the component, the constituents of the component, and the functional requirements satisfied by each part. |
| **Dependencies** | How the component’s function and performance relate to other components. How this component is used by other components. The other components that use this component. Interaction details such as timing, interaction conditions (such as order of execution and data sharing), and responsibility for creation, duplication, use, storage, and elimination of components. |
| **Interfaces** | Detailed description of all external or internal interfaces as well as of any mechanism for communicating through messages, parameters, or common data areas. All error messages and error codes should be identified. All screen formats, interactive messages, and other user interface components (originally defined in the SRS) should be given here. |
| **Resources** | A complete description of all resources (hardware or software) external to the component but required to carry out its functions. |
| **Processing** | A full description of the functions presented in the Function subsection. Pseudocode can be used to document algorithms, equations, and logic. |
| **Data** | For the data internal to the component, describes the representation method, initial values, use, semantics, and format. |

# **INTEGRATIONS**

# The **College Finder** is a web application designed for a smooth user experience. It utilizes a **frontend-only architecture** for efficiency and responsiveness. The application integrates various tools and components to achieve its functionality.

# **Key Features and Technologies:**

1. **API Integration:** The application retrieves college-related data from **external APIs or static JSON files**, enabling **real-time rendering of college listings**. This approach eliminates the need for a backend server or database and allows dynamic updates based on user selections like city or state.
2. **Frontend Development:** Built entirely with **React**, the application features a **component-based structure**, promoting scalability and modularity. React's state and props system effectively manages user input and dynamically updates the UI without requiring page reloads.
3. **Hosting and Deployment:** The application is deployed on **static hosting platforms** such as **Vercel** or **Netlify**. These platforms offer **continuous deployment from GitHub**, ensuring fast loading times and supporting custom domain management.
4. **Development and Debugging:** **Visual Studio Code** was used for development, with **Git and GitHub** integrated for version control and collaborative efforts. **Browser DevTools** were extensively employed for **testing and debugging**, guaranteeing optimal performance across diverse devices and screen sizes.

# **APPENDICES**

## Appendix A – Detailed Description of Components

| **Component** | **Technologies Used** | **Description** |
| --- | --- | --- |
| **Front-End Interface** | React.js, HTML, CSS, JavaScript | Provides the user-facing interface for exploring college details. Includes dropdowns for state/city selection, filters, and detailed results (contact info, website, fees). Fully responsive across desktops, tablets, and mobiles. |
| **Back-End Server** | Not applicable (No backend used) | No backend server is implemented. All logic and data handling occur on the client-side using React and external/static APIs. |
| **Database** | Static JSON files or Public APIs | Institutional data is retrieved through API integration or pre-structured JSON files. This enables fast rendering without needing a dedicated database. |
| **API Endpoints** | Not applicable (No custom APIs) | The application does not include custom REST APIs. Instead, it consumes external data sources directly or uses predefined JSON structures. |
| **Deployment Environment** | Netlify, Vercel | The app is deployed as a static site using Netlify or Vercel, offering continuous deployment, custom domains, CDN support, and high scalability. |
| **Development Tools** | Visual Studio Code (VS Code), Git, GitHub | Used for coding, version control, and collaboration. GitHub supports team-based development and smooth integration with deployment platforms. |