



SOFTWARE DESIGN SPECIFICATION

WEB DEVELOPMENT   
  
COLLEGE FINDER

|  |  |  |  |
| --- | --- | --- | --- |
| **Created By:** | Abuzaid Inamdar | **Approved By:** | <Domain Lead Name> |
| **Created On:** | 10-07-2025 | **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**

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 scope of the **COLLEGE FINDER** web application includes outlining its **key features**, **benefits**, and **limitations**. The system is designed to simplify access to reliable information about **IT and Management colleges** in India through a clean, responsive user interface. It enables users to search colleges by **city or state** and access **detailed information** such as courses offered, contact details, website, and fee structure — all without the clutter of advertisements or login requirements.

The application was developed using a **modular, API-driven approach** with a strong focus on usability, responsiveness, and clarity.

1. **Tools and Inputs Used**

* **React JS** – Frontend library used to build a dynamic, component-based user interface
* **HTML & CSS** – For structuring and styling the application
* **JavaScript** – Core logic for data handling, user interaction, and API integration
* **External/Public APIs** – Used to fetch college-related data (or static files if APIs were not available)
* **VS Code** – Code editor used throughout the development
* **Git & GitHub** – For version control and project collaboration
* **Browser DevTools** – For debugging and responsive testing
* **Responsive Design Principles** – Ensuring compatibility across devices (mobile, tablet, desktop)

# **SYSTEM OVERVIEW**

1. **System Components Overview – COLLEGE FINDER**

This section provides an outline of the various components and subsystems of the **COLLEGE FINDER** application:

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

* Built using **React.js**, leveraging a **component-based architecture** for efficient UI development
* Provides an **interactive interface** where students can input preferences such as **location (city/state)**
* Implements a **responsive design** compatible with both desktop and mobile devices
* Displays **dynamic college listings** based on user selections with structured information (contact, website, fees, etc.)
* Designed for **clean user experience** without ads or login requirements

1. **Data Handling via API Integration**

* College data is retrieved using **external/public APIs** or **curated static JSON files**
* The system does not use a traditional backend or database; all data is managed client-side
* React components fetch and render data dynamically to ensure fast performance and real-time updates

1. **Search & Filter System**

* Users can **filter colleges** based on location (city/state)
* The system supports **real-time filtering** of college listings based on user input
* Designed for fast rendering and intuitive navigation, with the possibility of expanding to support additional criteria in the future (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 open app

Select on enter location

Click find colleges

App calls (gemini.searchCollege location )

Gemini ai receive prompt

Gemini ai return JSON college list

App parses and validate data

Fallback : simpler gemini prompt

Parse fallback data

Display colleges in college result

User can load more result

User can view college detail

## 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** web application is designed to deliver a seamless user experience using a lightweight and efficient **frontend-only architecture**. The following tools and components are integrated to enhance functionality, responsiveness, and maintainability:

# **API Integration:**

# The application integrates **external or manually curated APIs/static JSON files** to fetch college-related data. This approach allows **real-time rendering of college listings** based on user selections (city/state), without the need for a backend server or database.

# **Frontend Development (React):**

# The entire application is built using **React**, enabling a **component-based structure** for a scalable and modular user interface. React’s state and props system efficiently handles user input and dynamically updates the UI without page reloads.

# **Hosting and Deployment Platforms:**

# The application is deployed using **static hosting platforms** such as **Vercel** or **Netlify**, which provide **continuous deployment from GitHub**, fast loading times, and support for custom domain management.

# **Development and Debugging Tools:**

# Development was carried out using **Visual Studio Code**, integrated with **Git and GitHub** for version control and collaboration. **Browser DevTools** were used extensively for **testing and debugging**, ensuring the app performs well across different devices and screen sizes.

# **APPENDICES**

## Appendix A – Detailed Description of Components

*For e.g.*

|  |  |
| --- | --- |
| *Front-End Interface* | * ***Technologies Used:*** *React.js, HTML, CSS, JavaScript* * ***Description:*** *Provides the visual interface for users to search and view college information. Includes features like a* ***dropdown to select city or state****, filters, and detailed result display for each college (contact info, website, fees, etc.). The interface is* ***responsive*** *and designed for compatibility across desktops, tablets, and mobile devices.* |
| *Back-End Server:* | ** ***Technologies Used:*** *Not applicable (No backend used)*  ** ***Description:*** *The application does* ***not use a backend server****. All logic and data processing are handled on the* ***client-side using React*** *and external/static APIs.* |
| *Database:* | ** ** ***Technologies Used:*** *Static JSON files or Public APIs*  ** ***Description:*** *Data about colleges is retrieved using* ***API integration*** *or* ***locally hosted structured JSON files****. This allows fast, dynamic rendering of college listings without needing a database or backend processing.* |
| *API Endpoints:* | ** ***Technologies Used:*** *Not applicable (No custom backend APIs)*  ** ***Description:*** *The application does* ***not implement custom REST APIs****. Instead, it consumes* ***external data sources*** *directly within the frontend or uses pre-defined JSON structures.* |
| *Deployment Environment:* | ** ***Technologies Used:*** *Netlify, Vercel*  ** ***Description:*** *The application is* ***deployed as a static site*** *using platforms like Netlify or Vercel. These services offer* ***continuous deployment****, custom domains, fast global content delivery, and ensure uptime and scalability.* |
| *Development Tools:* | ** ***Tools Used:*** *Visual Studio Code (VS Code), Git, GitHub*  ** ***Description:*** *Used for coding, version control, and collaboration. GitHub enables team-based development and easy integration with deployment platforms.* |