SB Works

Freelancing Platform

SDLC Phase 3

SYSTEM DESIGN REPORT

Project Name	SB Works — Online Freelancing Platform
Phase	Phase 3: System Design
Prepared By	I. Sai Ganesh — Lead Developer
Date	January 2024

1. Introduction to System Design

1.1 What is System Design?

System Design is Phase 3 of the SDLC. At this point we know what to build (from Phase 1) and what it should do (from Phase 2). Now we figure out how it will be built — the architecture, the database structure, the API endpoints, the page layouts, and the folder structure. Think of it as drawing the complete engineering blueprint before any actual construction begins.

Good design at this stage saves enormous time in Phase 4 (coding). When a developer sits down to write code, they should already know exactly what database tables they need, what API routes to create, and how the frontend pages will be organized. That is exactly what this phase produced for SB Works.

> ■■ Design Principle used: MVC (Model-View-Controller) pattern. Models handle data, Controllers handle business logic, and Views (React components) handle display. This separation makes the code easier to understand, test, and maintain.

1.2 System Architecture Overview

SB Works uses a three-tier client-server architecture:

Layer	Technology	Port	Responsibility
Presentation (Frontend)	React 18 + Vite	3000	User interface, routing, state management
API (Backend)	Express.js + Node.js	5000	Business logic, authentication, file handling
Real-time Layer	Socket.io	5000	Live chat messages and push notifications
Data Layer	MongoDB + Mongoose	27017	Data persistence, queries, schema validation

Data Flow: The user's browser talks to the React app (port 3000). React sends HTTP requests to the Vite dev server which proxies all /api calls to Express on port 5000. Express queries MongoDB and returns JSON. Socket.io connections go directly from browser to port 5000 for real-time features.

2. Database Design

2.1 Database Schema Overview

SB Works uses 8 MongoDB collections. Each is described below with its key fields and how it relates to other collections:

Collection: users

Stores all user accounts — admins, clients, and freelancers

Field	Type	Description
name	String	Full name (set during profile completion)
phoneNumber	String	Primary identifier, used for OTP login
email	String	Email address
role	String	USER / OWNER / FREELANCER / ADMIN
otp.code	Number	6-digit OTP, expires in 90 seconds
status	Number	0=Rejected, 1=Pending, 2=Approved
isActive	Boolean	True after profile is completed
rating	Number	Average star rating from reviews
totalReviews	Number	Count of reviews received

Collection: projects

Stores all freelance projects posted by clients

Field	Type	Description
title	String	Project title, min 10 characters
description	String	Detailed project description
status	String	OPEN (accepting bids) or CLOSED
category	ObjectId	Reference to categories collection
budget	Number	Project budget in USD
deadline	Date	Project deadline
owner	ObjectId	Reference to the client user who posted
freelancer	ObjectId	Reference to hired freelancer (null if not hired yet)
proposals	[ObjectId]	Array of proposal references

Collection: proposals

Stores freelancer bids on projects

Field	Type	Description
description	String	Cover letter from freelancer
price	Number	Bid amount in USD
duration	Number	Promised delivery time in days
user	ObjectId	Freelancer who submitted the bid
project	ObjectId	Project being bid on
status	Number	0=Rejected, 1=Pending, 2=Accepted

Collection: submissions

Stores work submitted by freelancers for review

Field	Type	Description
project	ObjectId	Related project
freelancer	ObjectId	Freelancer who submitted
description	String	Work description
fileUrl	String	Path to uploaded file in /uploads/
status	String	submitted / approved / revision_requested
clientFeedback	String	Feedback when revision is requested

2. Database Design (Continued)

Collection: messages

Stores all chat messages between users

Field	Type	Description
sender	ObjectId	User who sent the message
receiver	ObjectId	User who receives the message
content	String	Text content of the message
isRead	Boolean	Whether recipient has read it
project	ObjectId	Optional — links message to a project

Collection: reviews

Stores star ratings and reviews

Field	Type	Description
project	ObjectId	Project the review relates to
reviewer	ObjectId	User who wrote the review
reviewee	ObjectId	User being reviewed
rating	Number	Star rating from 1 to 5
comment	String	Written review text

Collection: notifications

Stores all user notifications

Field	Type	Description
user	ObjectId	Who the notification is for
title	String	Short notification headline
message	String	Full notification message
type	String	bid / message / submission / review / general
isRead	Boolean	Whether user has seen it
link	String	Deep-link to relevant page

Collection: categories

Stores project category options

Field	Type	Description
title	String	Category name, must be unique
description	String	Short description of the category
type	String	Always "project" in this system

3. API Design and Folder Structure

3.1 REST API Design Principles

The API follows REST (Representational State Transfer) conventions. Each resource has its own URL path, and actions are determined by the HTTP method:

HTTP Method	Action	Example Endpoint	What It Does
GET	Read data	/api/project/list	Fetch list of open projects
POST	Create new data	/api/project/add	Create a new project
PATCH	Update part of data	/api/proposal/:id	Accept or reject a proposal
DELETE	Remove data	/api/project/:id	Delete a project

3.2 API Route Groups

Route Group	Base Path	Number of Endpoints	Who Uses It
Authentication	/api/user	6 endpoints	All users
Projects	/api/project	7 endpoints	Owners + Freelancers
Proposals	/api/proposal	4 endpoints	Owners + Freelancers
Messages	/api/message	4 endpoints	All users
Submissions	/api/submission	4 endpoints	Freelancers + Owners
Reviews	/api/review	3 endpoints	Owners
Notifications	/api/notification	4 endpoints	All users
Categories	/api/category	1 endpoint	All users
Admin	/api/admin	6 endpoints	Admin only

3.3 Backend Folder Structure Design

The backend was designed with a clean, logical folder structure:

Folder / File	Purpose
index.js	Entry point — starts the server
seed.js	Seeds database with categories and admin account
.env	All configuration variables (never commit to Git)
app/models/	8 Mongoose schemas defining database structure

app/http/controllers/	9 controller files — all business logic lives here
app/router/	Route definitions — maps URLs to controller functions
app/http/middlewares/	Auth checks, permission guards, request validation
app/http/validators/	Joi validation schemas for request body checking
app/server.js	Express + Socket.io setup and configuration
utils/functions.js	Helper functions: JWT creation, OTP generation
utils/constants.js	Shared constants: role names, status codes
uploads/	Physical storage for uploaded work submission files

4. Frontend Design and UI Architecture

4.1 Frontend Technology Choices

The frontend uses React 18 with Vite as the build tool. Pages are organized into role-based layouts that share a common sidebar and topbar, but show different navigation items depending on the user's role.

Component Type	Description	Technologies Used
Layouts	Sidebar + topbar wrappers for each role	React, Bootstrap CSS
Pages	Full page content components	React, Axios, Socket.io-client
Services	All API call functions organized by feature	Axios with interceptors
Context	Global state: current user, socket connection	React Context + Socket.io
Components	Reusable pieces: ChatWindow, Topbar, Modals	React, Material UI

4.2 Page Inventory

Page	Route	Role	Key Features
Home	/	Public	Hero section, stats, feature cards, call-to-action buttons
Auth	/auth	Public	2-step OTP login: phone entry then OTP verification
CompleteProfile	/complete-profile	New User	Name, email, role selection (Owner or Freelancer)
OwnerDashboard	/owner/dashboard	Owner	Stats cards, recent projects table, quick actions
OwnerProjects	/owner/projects	Owner	Project list, create/edit/delete/toggle status
OwnerProject	/owner/projects/:id	Owner	Proposals list, accept/reject, submission review, review form
OwnerChat	/owner/chat	Owner	Conversation list, real-time message thread
FreelancerDashboard	/freelancer/dashboard	Freelancer	Stats, profile card, recent proposals
BrowseProjects	/freelancer/browse	Freelancer	Project cards grid, search, submit proposal modal
FreelancerProposals	/freelancer/proposals	Freelancer	Proposal status table
FreelancerProjects	/freelancer/my-projects	Freelancer	Active projects, submit work modal
FreelancerChat	/freelancer/chat	Freelancer	Chat with clients
AdminDashboard	/admin/dashboard	Admin	Platform stats, recent activity

AdminUsers	/admin/users	Admin	User table, search, change status
AdminProjects	/admin/projects	Admin	All projects across platform

■ Design Outcome: By the end of Phase 3, the team had complete database schemas for all 8 collections, a full API route map with 39 endpoints, a frontend page inventory with 15 pages, and a folder structure ready to be implemented in Phase 4.