SB Works

Freelancing Platform

Complete Project Documentation

MERN Stack

MongoDB · Express · React · Node.js

Version	1.0.0
Stack	MERN (MongoDB, Express, React, Node.js)
Roles	Admin · Client (Owner) · Freelancer
Real-time	Socket.io Chat & Notifications
Auth	OTP Phone Authentication (JWT Cookies)

Table of Contents

1. Project Overview

What is SB Works?

SB Works is a full-stack freelancing marketplace platform built with the MERN stack (MongoDB, Express.js, React.js, Node.js). It connects skilled freelancers with clients who need work done — similar to platforms like Fiverr or Upwork — but built entirely from scratch as a complete, production-ready web application.

The platform supports three distinct user roles, a complete project lifecycle from posting to payment-ready completion, real-time chat and notifications via Socket.io, OTP-based phone authentication without passwords, file upload for work submissions, and a star rating and review system.

Key Features

■ **Project Posting**	Clients post detailed projects with budget, deadline, category and tags
■ **Proposal System**	Freelancers browse open projects and submit competitive bids with cover letters
■ **Hiring System**	Clients review proposals, accept the best bid, and hire freelancers directly
■ **Real-time Chat**	Integrated Socket.io messaging between clients and freelancers
■ **Work Submission**	Freelancers submit completed work with file attachments for client review
■ **Work Review**	Clients approve work or request revisions with detailed feedback
■ **Reviews & Ratings**	1-5 star rating system builds freelancer reputation and trust
■ **Notifications**	Real-time bell notifications for bids, messages, approvals and reviews
■ **Admin Panel**	Administrators manage users, approve accounts, and oversee all activity
■ **OTP Authentication**	Secure phone-based login with one-time passwords — no passwords to remember

2. System Architecture

SB Works follows a classic client-server architecture with a React single-page application on the frontend communicating with an Express REST API on the backend. Real-time features are handled by a Socket.io server running alongside the Express app on the same port.

PRESENTATION LAYER	**React 18 + Vite**	Bootstrap 5 + Material UI	**Port 3000**
API GATEWAY	**Express.js REST API**	JWT Cookie Auth	**Port 5000**
REAL-TIME LAYER	**Socket.io Server**	Events: Chat, Notifications	**Same Port 5000**
DATA LAYER	**MongoDB Atlas / Local**	Mongoose ODM	**8 Collections**

Data Flow

The browser sends HTTP requests to the Vite dev server (port 3000), which proxies all `/api` and `/uploads` requests to the Express backend (port 5000). Authentication uses signed HttpOnly JWT cookies — the access token expires in 1 day, and a refresh token lasts 1 year. Socket.io connections are established directly from the browser to port 5000 using WebSockets.

■ In production, you would put Nginx in front of both servers — serving the React build on port 80 and proxying /api and socket.io to the Node.js backend.

3. Technology Stack

Category	Technology	Version	Purpose
Frontend Framework	React.js	18.2.0	UI component library
Build Tool	Vite	5.2.0	Dev server & bundler
CSS Framework	Bootstrap 5	5.3.2	Responsive layout & utilities
UI Components	Material UI	5.15.10	Advanced components (menus, badges)
HTTP Client	Axios	1.7.2	All API calls from frontend
Routing	React Router DOM	6.23.1	Client-side navigation
Real-time	Socket.io-client	4.7.2	Chat & notifications
Icons	React Icons	5.2.1	MdDashboard, FaBell etc.
Toasts	React Hot Toast	2.4.1	Success/error notifications
Backend Framework	Express.js	4.18.2	REST API server
Database	MongoDB	7.x	NoSQL document database
ODM	Mongoose	7.1.0	MongoDB object modelling
Real-time Server	Socket.io	4.7.2	WebSocket server
Authentication	jsonwebtoken	9.0.0	JWT token generation
File Upload	Multer	1.4.5-lts	Handle multipart/form-data
Validation	Joi	17.9.2	Request body validation
Password Hashing	Cookie-parser	1.4.6	Signed cookie handling
Dev Server	Nodemon	3.1.13	Auto-restart on file changes

4. Project Structure

Backend Structure

```
SBWorks-Fixed/
■■■ backend/
■ ■■■ index.js ← Entry point
■ ■■■ seed.js ← Database seeder
■ ■■■ .env ← Environment variables
■ ■■■ uploads/ ← Uploaded work files
■ ■■■ utils/
■ ■ ■■■ functions.js ← JWT helpers, random generators
■ ■ ■■■ constants.js ← ROLES enum, patterns
■ ■■■ app/
■ ■■■ server.js ← Express + Socket.io setup
■ ■■■ models/ ← 8 Mongoose models
■ ■ ■■■ user.js
■ ■ ■■■ project.js
■ ■ ■■■ proposal.js
■ ■ ■■■ message.js
■ ■ ■■■ submission.js
■ ■ ■■■ review.js
■ ■ ■■■ notification.js
■ ■ ■■■ category.js
■ ■■■ router/ ← Route definitions
■ ■ ■■■ router.js ← Main router (applies middleware)
■ ■ ■■■ userAuth.js
■ ■ ■■■ project.js
■ ■ ■■■ proposal.js
■ ■ ■■■ message.js
■ ■ ■■■ submission.js
■ ■ ■■■ review.js
■ ■ ■■■ notification.js
■ ■ ■■■ category.js
■ ■ ■■■ admin/
■ ■ ■■■ admin.routes.js
■ ■ ■■■ user.js
■ ■ ■■■ category.js
■ ■■■ http/
■ ■■■ controllers/ ← Business logic
■ ■■■ middlewares/ ← Auth & permission guards
■ ■■■ validators/ ← Joi schemas
```

Frontend Structure

```
■■■ frontend/
■■■ vite.config.js ← Proxy config (API → port 5000)
■■■ index.html
■■■ src/
■■■ main.jsx ← App entry, providers setup
■■■ App.jsx ← Routes definition
■■■ index.css ← Global styles + SB Works theme
■■■ context/
■ ■■■ AuthContext.jsx ← Global user state + Socket.io
■■■ services/ ← All Axios API calls
■ ■■■ httpService.js ← Axios instance + interceptors
■ ■■■ authService.js
■ ■■■ projectService.js
■ ■■■ proposalService.js
■ ■■■ messageService.js
■ ■■■ submissionService.js
■ ■■■ reviewService.js
■ ■■■ notificationService.js
■ ■■■ categoryService.js
■■■ layouts/ ← Role-based sidebar layouts
■ ■■■ OwnerLayout.jsx
■ ■■■ FreelancerLayout.jsx
■ ■■■ AdminLayout.jsx
■■■ components/
■ ■■■ Topbar.jsx ← Notifications bell + user menu
■ ■■■ ChatWindow.jsx ← Reusable chat component
■ ■■■ owner/
■ ■■■ CreateProjectModal.jsx
■■■ pages/
■■■ Home.jsx ← Landing page
■■■ Auth.jsx ← OTP Login
■■■ CompleteProfile.jsx
■■■ owner/ ← 4 client pages
■■■ freelancer/ ← 5 freelancer pages
■■■ admin/ ← 4 admin pages
```

5. User Roles & Permissions

SB Works has three distinct roles. Each role gets its own dashboard, sidebar navigation, and set of permissions enforced both on the frontend (route guards) and backend (middleware).

■ ADMIN

- View all users on the platform with search functionality
- Approve users (set status to Approved/Pending/Rejected)
- View all projects posted on the platform
- View all proposals submitted across all projects
- Add and manage project categories
- Access to all API routes without restriction

■ CLIENT / OWNER

- Post new projects with title, description, category, budget, deadline, tags
- Edit and delete own projects (cannot delete if freelancer assigned)
- Open and close projects (toggle OPEN/CLOSED status)
- View all proposals submitted to their projects
- Accept a proposal to hire the freelancer
- Reject unwanted proposals
- View submitted work files and descriptions
- Approve completed work or request revisions with feedback
- Leave star ratings (1–5) and written reviews for freelancers
- Real-time chat with freelancers
- Receive notifications for new proposals and work submissions

■ FREELANCER

- Browse all open projects with search functionality
- Submit proposals with bid amount, delivery time, and cover letter
- View status of all submitted proposals (pending/accepted/rejected)
- View active projects where proposal was accepted
- Submit completed work with description and file attachment
- Resubmit work if client requests revision
- View client feedback on submissions
- Real-time chat with clients
- Receive notifications for proposal decisions and work approvals

- Build reputation through star ratings from clients

6. Authentication System

SB Works uses OTP (One-Time Password) phone authentication — there are no traditional passwords. Users enter their phone number, receive a 6-digit code, verify it, and get logged in. JWTs are stored in signed HttpOnly cookies (not localStorage) for security.

Authentication Flow

Step	Action	Details
`Step 1`	Enter Phone Number	`User submits phone number → backend generates 6-digit OTP → stores in DB with 90-second expiry`
`Step 2`	Receive OTP	`In dev mode: OTP prints in backend terminal. In production: SMS via Twilio API`
`Step 3`	Verify OTP	`User submits OTP → backend validates code + expiry → marks phone as verified`
`Step 4`	Set Tokens	`Backend sets accessToken cookie (1 day) and refreshToken cookie (1 year) as signed HttpOnly cookies`
`Step 5`	New User Check	`If user.isActive is false → redirect to /complete-profile to set name, email, and role`
`Step 6`	Complete Profile	`User sets name, email, and picks role (FREELANCER or OWNER) → isActive becomes true`
`Step 7`	Role Redirect	`OWNER → /owner/dashboard | FREELANCER → /freelancer/dashboard | ADMIN → /admin/dashboard`
`Step 8`	Auto Refresh	`When access token expires, Axios interceptor calls /api/user/refresh-token automatically`

Token Security

- ■ Access Token: 1-day expiry (JWT)
- ■ Refresh Token: 1-year expiry (JWT)
- ■ Stored as signed HttpOnly cookies
- ■ Not accessible by JavaScript (XSS safe)
- ■ Cookie signed with COOKIE_PARSER_SECRET_KEY
- ■ verifyAccessToken middleware on protected routes
- ■ isVerifiedUser checks account status
- ■ authorize() checks role permissions

■■ Admin Quick Login (Development Only): Phone +10000000000 · OTP 123456 (permanent test OTP set by seed.js)

7. Database Models

The database has 8 Mongoose collections. Each model is described below with its fields, types, and purpose.

User Model

Field	Type	Description
`name`	String	`Full name (set during profile completion)`
`phoneNumber`	String	`Used as login identifier`
`email`	String	`Set during profile completion`
`role`	String	`USER / FREELANCER / OWNER / ADMIN`
`otp.code`	Number	`6-digit OTP code`
`otp.expiresIn`	Date	`OTP expiry timestamp (90 seconds)`
`isVerifiedPhoneNumber`	Boolean	`True after first OTP verification`
`isActive`	Boolean	`True after profile completion`
`status`	Number	`0=Rejected, 1=Pending, 2=Approved`
`rating`	Number	`Average star rating from reviews`
`totalReviews`	Number	`Count of reviews received`
`skills`	[String]	`Array of skill tags`

Project Model

Field	Type	Description
`title`	String	`Project title (min 10 chars)`
`description`	String	`Full project description`
`status`	String	`OPEN (accepting bids) or CLOSED`
`category`	ObjectId → Category	`Project category reference`
`budget`	Number	`Client budget in USD`
`tags`	[String]	`Skill tags for the project`
`deadline`	Date	`Project completion deadline`
`owner`	ObjectId → User	`Client who posted the project`
`freelancer`	ObjectId → User	`Assigned freelancer (null until hired)`
`proposals`	[ObjectId → Proposal]	`Array of submitted proposals`

Proposal Model

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

Submission Model

Field	Type	Description
project	ObjectId → Project	Related project
freelancer	ObjectId → User	Freelancer who submitted
client	ObjectId → User	Client who owns the project
description	String	Work description/notes
fileUrl	String	Path to uploaded file in /uploads/
fileName	String	Original filename
status	String	submitted / approved / revision_requested
clientFeedback	String	Feedback from client on submission

Message Model

Field	Type	Description
sender	ObjectId → User	Message sender
receiver	ObjectId → User	Message recipient
project	ObjectId → Project	Optional project context
content	String	Message text content
isRead	Boolean	Whether recipient has read the message

Review Model

Field	Type	Description
project	ObjectId → Project	Project the review is for
reviewer	ObjectId → User	User leaving the review
reviewee	ObjectId → User	User being reviewed
rating	Number	Star rating 1–5
comment	String	Written review text

Notification Model

Field	Type	Description
user	ObjectId → User	Notification recipient
title	String	Notification headline
message	String	Notification body text
type	String	bid / message / submission / review / admin / general
isRead	Boolean	Whether user has seen the notification
link	String	Optional deep-link URL

Category Model

Field	Type	Description
title	String	Category name (unique)
description	String	Short description
type	String	Always "project" for this platform
parentId	ObjectId → Category	Optional parent category

8. Backend API Reference

Base URL: http://localhost:5000/api | All protected routes require valid accessToken cookie | Content-Type: application/json (except file uploads which use multipart/form-data)

Authentication Routes (/api/user)

Method	Endpoint	Auth	Description
`POST`	/user/get-otp	`None`	Send OTP to phone number
`POST`	/user/check-otp	`None`	Verify OTP and login
`POST`	/user/complete-profile	`Token`	Set name, email, role
`GET`	/user/profile	`Token`	Get current user profile
`GET`	/user/refresh-token	`Cookie`	Refresh access token
`POST`	/user/logout	`None`	Clear auth cookies

Project Routes (/api/project)

Method	Endpoint	Auth	Description
`GET`	/project/list	`Token`	List all projects (public filter)
`GET`	/project/owner-projects	`Owner`	Get my posted projects
`POST`	/project/add	`Owner`	Create new project
`GET`	/project/:id	`Owner`	Get project details + proposals
`PATCH`	/project/update/:id	`Owner`	Edit project details
`PATCH`	/project/:id	`Owner`	Toggle OPEN/CLOSED status
`DELETE`	/project/:id	`Owner`	Delete project (if no freelancer)

Proposal Routes (/api/proposal)

Method	Endpoint	Auth	Description
`GET`	/proposal/list	`Freelancer`	Get my proposals
`POST`	/proposal/add	`Freelancer`	Submit proposal to project
`GET`	/proposal/:id	`Freelancer`	Get single proposal
`PATCH`	/proposal/:id	`Owner`	Accept (2) or Reject (0) proposal

Message Routes (/api/message)

Method	Endpoint	Auth	Description
POST	/message/send	Token	Send a message
GET	/message/conversation/:userId	Token	Get chat with specific user
GET	/message/conversations	Token	Get all conversations
PATCH	/message/read/:senderId	Token	Mark messages as read

Submission Routes (/api/submission)

Method	Endpoint	Auth	Description
POST	/submission/submit	Freelancer	Submit work (multipart/form-data)
GET	/submission/project/:projectId	Token	Get submission for project
GET	/submission/my	Freelancer	Get all my submissions
PATCH	/submission/review/:id	Owner	Approve or request revision

Review Routes (/api/review)

Method	Endpoint	Auth	Description
POST	/review/add	Owner	Submit star rating + review
GET	/review/user/:userId	Token	Get all reviews for a user
GET	/review/project/:projectId	Token	Get reviews for a project

Notification Routes (/api/notification)

Method	Endpoint	Auth	Description
GET	/notification	Token	Get my notifications
PATCH	/notification/read/:id	Token	Mark one as read
PATCH	/notification/read-all	Token	Mark all as read
DELETE	/notification/:id	Token	Delete notification

Admin Routes (/api/admin)

Method	Endpoint	Auth	Description
GET	/admin/user/list	Admin	List all users (with search)
PATCH	/admin/user/verify/:userId	Admin	Change user status (0/1/2)
GET	/admin/user/profile/:userId	Admin	View user + their activity

`POST`	/admin/category/add	`Admin`	Add project category
`PATCH`	/admin/category/update/:id	`Admin`	Edit category
`DELETE`	/admin/category/remove/:id	`Admin`	Delete category

9. Frontend Pages & Components

Page / Component	Route	Role	Description
Home	/	Public	Landing page with hero, features, CTA and footer
Auth	/auth	Public	Two-step OTP login: phone number → OTP verification
CompleteProfile	/complete-profile	New User	Set name, email, pick FREELANCER or OWNER role
OwnerDashboard	/owner/dashboard	Owner	Stats cards + recent projects + quick actions
OwnerProjects	/owner/projects	Owner	Full project list with create/edit/delete/toggle
OwnerProject	/owner/projects/:id	Owner	Project detail: proposals, submission review, star rating
OwnerChat	/owner/chat/:userId?	Owner	Real-time chat with freelancers
FreelancerDashboard	/freelancer/dashboard	Freelancer	Stats + recent proposals + profile card
BrowseProjects	/freelancer/browse	Freelancer	Browse open projects + submit proposals modal
FreelancerProposals	/freelancer/proposals	Freelancer	Table of all submitted proposals + status
FreelancerProjects	/freelancer/my-projects	Freelancer	Active projects + submit/resubmit work modal
FreelancerChat	/freelancer/chat/:userId?	Freelancer	Real-time chat with clients
AdminDashboard	/admin/dashboard	Admin	Platform stats + recent users overview
AdminUsers	/admin/users	Admin	User list with search + change status modal
AdminProjects	/admin/projects	Admin	View all projects across platform
AdminProposals	/admin/proposals	Admin	View all proposals across platform
Topbar	Component	All roles	Notification bell (MUI Badge) + user menu + logout
ChatWindow	Component	All roles	Reusable chat: conversation list + message window
CreateProjectModal	Component	Owner	Bootstrap modal form: title, desc, category, budget

CSS Architecture

All styles are in `src/index.css` using Bootstrap 5 as the base. Custom SB Works styles are layered on top using CSS variables for the brand color palette.

```
:root {
--sb-primary: #2563eb; /* Blue — main brand color */
--sb-secondary: #7c3aed; /* Purple — accent */
--sb-success: #16a34a; /* Green — approved/open */
--sb-warning: #d97706; /* Amber — pending */
--sb-danger: #dc2626; /* Red — rejected/closed */
--sb-sidebar-bg: #1e293b; /* Dark navy sidebar */
}
.sb-sidebar { width:260px; position:fixed; background:var(--sb-sidebar-bg); }
.sb-main { margin-left:260px; min-height:100vh; }
.sb-card { background:#fff; border-radius:12px; box-shadow:0 4px 6px
rgba(0,0,0,.1); }
.badge-open { background:#dcfce7; color:#16a34a; }
.badge-closed{ background:#fee2e2; color:#dc2626; }
.star-rating .star { font-size:1.4rem; cursor:pointer; color:#d1d5db; }
.star-rating .star.filled { color:#f59e0b; }
```

10. Real-time Features (Socket.io)

Socket.io runs on the same port as Express (5000) using the Node.js http.createServer() pattern. The frontend connects directly to http://localhost:5000 with credentials.

Socket Events Reference

Event Name	Direction	Payload	Description
`join`	Client → Server	`{ userId }`	User joins their room after login
`sendMessage`	Client → Server	`{ receiverId, message }`	Relay message to recipient
`receiveMessage`	Server → Client	`Message object`	New message received in real-time
`receiveNotification`	Server → Client	`{ title, message }`	Push notification to user
`sendNotification`	Client → Server	`{ receiverId, notification }`	Manual notification trigger
`disconnect`	Auto	–	User removed from onlineUsers map

When are Notifications Sent?

Trigger	Who Gets Notified
`New Proposal Submitted`	Owner receives notification when a freelancer submits a proposal
`Proposal Accepted`	Freelancer notified when their proposal is accepted — hired!
`Proposal Rejected`	Freelancer notified when their proposal is rejected
`New Message`	Recipient notified when a new chat message arrives
`Work Submitted`	Owner notified when freelancer submits completed work
`Work Approved`	Freelancer notified when client approves their submission
`Revision Requested`	Freelancer notified when client requests changes with feedback
`New Review`	User notified when they receive a star rating and review

11. Complete User Workflows

■ Workflow 1: Post a Project (Client)

1. Login as Owner → go to My Projects
2. Click "+ Post Project" button
3. Fill in: Title (min 10 chars), Description, Category, Budget ($), Deadline, Tags
4. Click "Post Project" → project appears with OPEN status
5. Share project URL or wait for freelancers to find it via Browse

■ Workflow 2: Submit a Proposal (Freelancer)

1. Login as Freelancer → go to Browse Projects
2. Search or scroll to find a suitable open project
3. Click "Submit Proposal" on any project card
4. Enter: Your Bid Amount ($), Delivery time (days), Cover Letter
5. Click "Submit Proposal" — client gets a real-time notification
6. Track proposal status in My Proposals page (Pending → Accepted/Rejected)

■ Workflow 3: Hire a Freelancer (Client)

1. Login as Owner → My Projects → click "View Bids" on a project
2. Review all submitted proposals (bid amount, delivery time, cover letter)
3. Click "■ Message" to chat with a freelancer before deciding
4. Click "✓ Accept & Hire" on the best proposal
5. All other proposals are automatically rejected
6. Freelancer gets a real-time "Proposal Accepted!" notification
7. Project now shows the assigned freelancer in the sidebar

■ Workflow 4: Submit Work (Freelancer)

1. Login as Freelancer → My Projects page
2. Find the active project (accepted proposal) in the list
3. Click "■ Submit Work" button
4. Write a description of what was completed
5. Optionally attach a file (zip, pdf, images, etc.)

6. Click "Submit Work" — client gets a real-time notification
7. Submission status shows as "Under Review"

■ Workflow 5: Review & Approve Work (Client)

1. Login as Owner → My Projects → View Bids on the project
2. Scroll to "Submitted Work" section
3. Read the description and download the file if attached
4. Option A: Click "✓ Approve Work" — project closes, freelancer notified
5. Option B: Write feedback and click "■ Request Revision" — freelancer notified
6. If revision requested: freelancer can resubmit from My Projects page
7. After approval: "■ Leave a Review" button appears

■ Workflow 6: Leave a Review (Client)

1. After approving work, click "■ Leave a Review"
2. Click on stars to set rating (1–5)
3. Write a review comment about the freelancer
4. Click "Submit Review" — freelancer profile rating updates automatically
5. Freelancer receives a real-time review notification

■ Workflow 7: Admin User Management

1. Login as Admin (Phone: +10000000000, OTP: 123456)
2. Go to Users → see all registered users
3. Search by name, email or phone number
4. Click "Change Status" on any user
5. Options: ✓ Approve User | ■ Set Pending | ✗ Reject User
6. Approved (status=2): user can access all features
7. Pending (status=1): user is awaiting approval
8. Rejected (status=0): user is blocked from the platform

12. Environment Configuration

The backend reads all configuration from the .env file in the backend folder. Never commit this file to version control.

```
# backend/.env

PORT=5000 # Express server port

# MongoDB Connection (choose one):
MONGO_URI=mongodb://localhost:27017/sbworks # Local MongoDB
# MONGO_URI=mongodb+srv://user:pass@cluster.mongodb.net/sbworks # Atlas

# JWT Secret Keys (change these in production!)
ACCESS_TOKEN_SECRET_KEY=sbworks_access_secret_key_2024_secure
REFRESH_TOKEN_SECRET_KEY=sbworks_refresh_secret_key_2024_secure
COOKIE_PARSER_SECRET_KEY=sbworks_cookie_secret_key_2024

NODE_ENV=development # Change to "production" when deploying
DOMAIN=localhost # Cookie domain
SERVER_URL=http://localhost:5000

# Twilio SMS (OPTIONAL - leave blank for dev mode)
# In dev mode, OTP is printed to terminal instead of SMS
TWILIO_ACCOUNT_SID=your_twilio_account_sid
TWILIO_AUTH_TOKEN=your_twilio_auth_token
TWILIO_PHONE_NUMBER=+1XXXXXXXXXX # Your Twilio phone number
```

Variable	Required	Description
PORT	Yes	Server port (default 5000)
MONGO_URI	Yes	MongoDB connection string — local or Atlas
ACCESS_TOKEN_SECRET_KEY	Yes	Secret for signing access JWTs — make it long and random
REFRESH_TOKEN_SECRET_KEY	Yes	Secret for signing refresh JWTs — different from above
COOKIE_PARSER_SECRET_KEY	Yes	Secret for signing cookies
NODE_ENV	Yes	development or production (affects cookie security)
DOMAIN	Yes	Cookie domain — localhost in development
TWILIO_ACCOUNT_SID	No	Twilio Account SID — skip for dev (OTP prints to console)
TWILIO_AUTH_TOKEN	No	Twilio Auth Token — skip for dev
TWILIO_PHONE_NUMBER	No	Twilio phone number to send SMS from

13. Setup & Installation Guide

Prerequisites

■ Node.js 18+ (download from nodejs.org)	■ MongoDB (local) OR MongoDB Atlas account (free)
■ npm (comes with Node.js)	■ Web browser (Chrome/Firefox)
■ Git (optional)	■ Two terminal windows

Step 1 — Extract & Setup MongoDB

- Extract SBWorks-Fixed.zip to a folder on your computer
- Option A (Cloud): Create free account at cloud.mongodb.com → create cluster → get connection string
- Option B (Local): Install MongoDB Community from mongodb.com/try/download/community

Step 2 — Configure Environment

- Open SBWorks-Fixed\backend\.env in Notepad
- Set MONGO_URI to your MongoDB connection string
- For Atlas: mongodb+srv://username:password@cluster.mongodb.net/sbworks
- For Local: mongodb://localhost:27017/sbworks (already set as default)

Step 3 — Install & Seed Backend

```
cd SBWorks-Fixed\backend
npm install
npm run seed ← Creates categories + admin account
npm run server ← Starts on port 5000
```

■ You should see: ■ SB Works server running on port 5000 ■ MongoDB Connected (after seed: ■ 10 categories + admin created)

Step 4 — Install & Start Frontend

```
cd SBWorks-Fixed\frontend
npm install
npm run dev ← Starts on port 3000
```

Step 5 — Open & Login

- Open http://localhost:3000 in your browser

- Admin login: Phone +10000000000 → OTP 123456
- New users: Enter any phone → check backend terminal for OTP code
- Complete profile: set name, email, pick role (Freelancer or Owner)
- Admin must approve new users before they can fully use the platform

14. Troubleshooting Guide

■ MongoDB connection failed (ECONNREFUSED)

MongoDB is not running on your machine.

Fix: Install MongoDB from mongodb.com/try/download/community

Or: Use MongoDB Atlas (free cloud) — see Section 12 for setup

Windows: Check if MongoDB service is running in Task Manager → Services

■ bad auth: Authentication failed (Atlas)

Username or password in MONGO_URI is wrong.

Fix: Go to Atlas → Security → Database Access → Edit user → Reset password

Use a simple password with only letters and numbers (no special chars)

If username contains @ or # characters, encode them: @ becomes %40

Easiest fix: Create a brand new user with username "sbadmin" and simple password

■ mongodb+srv URI cannot have port number

Your Atlas connection string has :27017 in it.

Fix: Remove the :27017 from your MONGO_URI string

Atlas SRV URIs never use a port number — it is resolved automatically

■ npm is not recognized (PowerShell)

PowerShell execution policy is blocking npm.

Fix Option A: Use Command Prompt (cmd) instead of PowerShell

Fix Option B: Run PowerShell as Administrator and type:

Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser

■ Missing script: dev or server

You are running npm from the wrong folder.

For backend commands: cd must be in SBWorks-Fixed\backend

For frontend commands: cd must be in SBWorks-Fixed\frontend

Run "npm run" to see available scripts in current folder

■ OTP not arriving via SMS

Twilio is not configured — this is expected in development.

The OTP is always printed in your backend terminal window.

Look for the line: "OTP for +XXXXX : 123456" in Terminal 1

To enable real SMS: add real Twilio credentials to .env file

■ 403 Forbidden on API calls

Your user account has not been approved by admin.

Fix: Login as Admin → Users → find your user → Click Change Status → Approve

Admin login: Phone +10000000000, OTP 123456

Also check that your profile is complete (name + email + role set)

■ Frontend shows blank page or crashes

Check if backend is running on port 5000 first.

Open browser DevTools (F12) → Console tab for errors.

Make sure npm install was run in the frontend folder.

Try clearing browser cache and hard refresh (Ctrl+Shift+R)

SB Works — Full Stack Freelancing Platform | Built with MERN Stack | MongoDB · Express.js · React.js · Node.js · Socket.io · Bootstrap 5 · Material UI