Great! Let's move on to the **System Design Document (SDD)** — this outlines how the system will be architected from both a high-level and low-level perspective.

**📄 4. System Design Document (SDD)**

**Project Name:** *FreelancerConnect*

**📘 1. Introduction**

**1.1 Purpose**

This document defines the architecture, system components, and data flow of the FreelancerConnect platform to guide the development team.

**1.2 Scope**

This SDD covers frontend, backend, database, API structure, key components, and how different modules interact to support business use cases.

**🧱 2. High-Level Architecture**

**2.1 Architecture Style:**

**Client-Server Architecture** using **RESTful APIs**

**2.2 Layers:**

User (Web Client)

↓

Frontend (React.js)

↓

Backend API (Spring Boot - REST)

↓

Database (MySQL/PostgreSQL)

↓

External Services (Payment Gateway, Email, Notifications)

**🧩 3. Components**

**🔹 3.1 Frontend (React.js)**

* Role-based authentication
* Forms: Registration, Project posting, Proposal submission
* Dashboards for Freelancer, Client, and Admin
* Chat interface (WebSocket/Firebase)

**🔹 3.2 Backend (Spring Boot)**

* User authentication & authorization (JWT)
* Role-based access
* REST APIs for all operations
* Data validation & error handling
* Payment service integration
* Notification service

**🔹 3.3 Database (MySQL/PostgreSQL)**

* Normalized relational schema
* Entity relationships for users, projects, proposals, reviews, messages
* Secure storage of sensitive data (password hashing, token encryption)

**🔹 3.4 External Integrations**

* **Razorpay/Stripe**: Payments & escrow
* **Firebase/WebSocket**: Real-time chat
* **SendGrid/Mailgun**: Emails for notifications

**🛤️ 4. Data Flow Diagrams**

**🔸 4.1 User Registration Flow**

1. User submits form (React)
2. API: /api/auth/register
3. Backend validates and stores user
4. JWT token is issued and sent to client

**🔸 4.2 Project Posting (Client)**

1. Client fills project form
2. API: /api/projects/create
3. Project saved with client ID
4. Visible to all freelancers

**🔸 4.3 Proposal Submission (Freelancer)**

1. Freelancer clicks "Apply" on a project
2. API: /api/proposals/submit
3. Proposal is stored and linked to freelancer/project

**🔸 4.4 Messaging**

1. WebSocket opens on login
2. Real-time messages are exchanged between users
3. Messages also stored via REST API /api/messages

**🔸 4.5 Payment & Withdrawals**

1. Client deposits funds via Razorpay
2. Admin holds in escrow
3. On job completion, system transfers to freelancer's wallet
4. Freelancer requests withdrawal → Razorpay disburses

**📦 5. Entity Relationship Diagram (Simplified)**

User (user\_id, name, email, role)

├── Freelancer Profile (skills, hourly\_rate)

└── Client Profile (company\_name)

Project (project\_id, client\_id, title, description, budget)

└── Proposal (proposal\_id, project\_id, freelancer\_id, bid, status)

└── ChatMessage (message\_id, sender\_id, receiver\_id, content)

Payment (payment\_id, user\_id, amount, status)

Review (review\_id, project\_id, from\_user\_id, to\_user\_id, rating, comment)

**🧰 6. Technology Stack**

| **Layer** | **Technology** |
| --- | --- |
| Frontend | React.js, Tailwind CSS |
| Backend | Spring Boot, Spring Security |
| Database | MySQL or PostgreSQL |
| Auth | JWT (JSON Web Tokens) |
| Chat | Firebase or WebSocket |
| Payments | Razorpay / Stripe |
| Hosting | Vercel (frontend), AWS/Render (backend) |
| CI/CD | GitHub Actions / Jenkins |
| Docs | Swagger / Postman for API Testing |

**🔐 7. Security Considerations**

* All sensitive routes protected via role-based access
* Passwords hashed using bcrypt
* JWT expiration + refresh tokens
* API rate limiting to prevent abuse
* HTTPS everywhere

**🧪 8. Monitoring & Logging**

* Use **ELK stack** or **Prometheus + Grafana** for logs and performance monitoring.
* Alert system for failures or downtimes.

This gives a clear structure and blueprint for your entire development process.

Next up:  
➡️ **Architecture Diagram with endpoints**  
Ready? Just reply: **continue**.