Awesome! Now let's move on to the **Low-Level Design (LLD)**. This document focuses on how individual components, classes, services, and data structures are implemented in your backend using **Spring Boot**, and how frontend components interact with them.

**📄 6. Low-Level Design (LLD)**

**Project:** *FreelancerConnect*  
**Technology Focus:** Spring Boot (Backend), React (Frontend)

**🧩 1. Backend Class Design (Java + Spring Boot)**

**🔸 Entity Classes (Models)**

**✅ User**

@Entity

public class User {

@Id @GeneratedValue

private Long id;

private String name;

private String email;

private String password;

private String role; // CLIENT or FREELANCER

private boolean isBlocked;

@OneToOne(mappedBy = "user", cascade = CascadeType.ALL)

private FreelancerProfile freelancerProfile;

@OneToOne(mappedBy = "user", cascade = CascadeType.ALL)

private ClientProfile clientProfile;

}

**✅ FreelancerProfile**

@Entity

public class FreelancerProfile {

@Id @GeneratedValue

private Long id;

private String skills;

private Double hourlyRate;

@OneToOne

private User user;

}

**✅ ClientProfile**

@Entity

public class ClientProfile {

@Id @GeneratedValue

private Long id;

private String companyName;

@OneToOne

private User user;

}

**✅ Project**

@Entity

public class Project {

@Id @GeneratedValue

private Long id;

private String title;

private String description;

private Double budget;

private String status; // OPEN, IN\_PROGRESS, COMPLETED

@ManyToOne

private User client;

@OneToMany(mappedBy = "project", cascade = CascadeType.ALL)

private List<Proposal> proposals;

}

**✅ Proposal**

@Entity

public class Proposal {

@Id @GeneratedValue

private Long id;

private String message;

private Double bidAmount;

private String status; // PENDING, ACCEPTED, REJECTED

@ManyToOne

private Project project;

@ManyToOne

private User freelancer;

}

**✅ Review**

@Entity

public class Review {

@Id @GeneratedValue

private Long id;

private Integer rating;

private String comment;

@ManyToOne

private User fromUser;

@ManyToOne

private User toUser;

@ManyToOne

private Project project;

}

**🧰 2. Service Layer Example**

**✅ UserService.java**

public interface UserService {

User register(RegisterDto dto);

User login(LoginDto dto);

User getProfile(Long userId);

void updateProfile(Long userId, UpdateProfileDto dto);

}

**✅ ProjectService.java**

public interface ProjectService {

Project createProject(Long clientId, ProjectDto dto);

List<Project> getAllProjects();

Project getProjectById(Long id);

}

**📦 3. DTOs (Data Transfer Objects)**

**✅ RegisterDto**

public class RegisterDto {

private String name;

private String email;

private String password;

private String role;

}

**✅ ProposalDto**

public class ProposalDto {

private Long projectId;

private Double bidAmount;

private String message;

}

**✅ ProjectDto**

public class ProjectDto {

private String title;

private String description;

private Double budget;

}

**🕹️ 4. Controller Layer (Sample)**

**✅ AuthController.java**

@RestController

@RequestMapping("/api/auth")

public class AuthController {

@PostMapping("/register")

public ResponseEntity<?> register(@RequestBody RegisterDto dto) {

return ResponseEntity.ok(userService.register(dto));

}

@PostMapping("/login")

public ResponseEntity<?> login(@RequestBody LoginDto dto) {

return ResponseEntity.ok(userService.login(dto));

}

}

**✅ ProjectController.java**

@RestController

@RequestMapping("/api/projects")

public class ProjectController {

@PostMapping

public ResponseEntity<?> postProject(@RequestBody ProjectDto dto, @AuthenticationPrincipal User user) {

return ResponseEntity.ok(projectService.createProject(user.getId(), dto));

}

@GetMapping

public List<Project> getAllProjects() {

return projectService.getAllProjects();

}

}

**⚙️ 5. Frontend Component Structure (React)**

src/

│

├── components/

│ ├── Auth/

│ │ └── LoginForm.jsx, RegisterForm.jsx

│ ├── Dashboard/

│ │ └── FreelancerDashboard.jsx, ClientDashboard.jsx

│ ├── Projects/

│ │ └── ProjectCard.jsx, PostProjectForm.jsx

│ └── Proposals/

│ └── ProposalList.jsx, SubmitProposalForm.jsx

│

├── pages/

│ ├── Home.jsx

│ ├── Login.jsx

│ ├── Register.jsx

│ ├── ProjectDetails.jsx

│ └── Dashboard.jsx

│

├── services/

│ ├── api.js (Axios instance)

│ └── authService.js, projectService.js, proposalService.js

This **LLD** ensures:

* Code modularity
* Clean class relationships
* Easily testable and scalable architecture

Next up:  
➡️ **Test Plan & Test Cases** — includes how you’ll test each feature (unit, integration, and manual/UAT).

Reply **continue** to proceed.