Chandigarh University

Department of Bachelor of Engineering in Computer Science

FreelancerUsingSpringBoot

Project Report

Name: Sayan Pradhan
UID: 23BCS10878
Program: Bachelor of Engineering in Computer Science
University: Chandigarh University

1970-01-01

Introduction & Abstract

FreelancerUsingSpringBoot is a full-stack freelancing marketplace that bridges employers seeking skilled talent and freelancers looking for project-based work. The application delivers an end-to-end workflow for project publication, bidding, hiring, and profile management while ensuring secure handling of user identities and digital assets. The backend is powered by Spring Boot 3.3.3 with Java 17, exposing RESTful services that manage business logic, persistence, and scheduled workflows. A React-based client and a lightweight Node/Express proxy application ensure responsive, role-aware experiences for employers and freelancers, complemented by reusable user interface components and Redux-powered state management. MySQL serves as the durable data store for transactional records, while attachments are persisted through an object storage abstraction. The platform emphasizes streamlined collaboration, transparent bid evaluation, and automation that promotes fair project awards.

Project Objectives

- Design a secure, scalable Spring Boot backend that manages users, projects, bids, and attachments with minimal latency.
- Deliver a responsive React front-end that differentiates between employer and freelancer journeys while preserving shared components.
- Implement end-to-end bid management, including creation, evaluation, auto-award scheduling, and post-award tracking.
- Provide seamless onboarding, authentication, and profile enrichment with encrypted credentials and session-aware access.
- Maintain data integrity via MySQL-backed persistence, validation, and structured error handling.

- Enable extensibility for analytics, payment integration, and real-time collaboration via modular service boundaries.

System Architecture

FreelancerUsingSpringBoot adopts a three-tier architecture underpinned by REST principles.

Frontend (Client Layer): Crafted with React 16 and Redux, the client delivers role-specific dashboards, reusable forms, and responsive layouts. Axios mediates API calls, and SweetAlert-based notifications surface system feedback. The client runs on port 3000.

Proxy/API Gateway (Integration Layer): A Node.js/Express server on port 3001 brokers frontend-to-backend requests, manages client sessions, and configures permissive CORS for local development. The proxy centralizes routing and shields backend endpoints from direct browser exposure.

Backend (Application Layer): The Spring Boot service on port 8080 encapsulates business logic, validation, auditing, and scheduled jobs. Controllers map HTTP requests, services orchestrate workflows, and repositories leverage Spring Data JPA for persistence.

Database (Data Layer): A MySQL 8.3 instance provides durable storage with normalized tables for users, projects, bids, and attachments. JPA entities enforce constraints, while automatic schema updates support iterative development.

High-level architecture of FreelancerUsingSpringBoot

High-level architecture of FreelancerUsingSpringBoot

Technology Stack (Detailed)

Backend (Port 8080)

- Spring Boot 3.3.3, Spring Web, Spring Data JPA, and Bean Validation for RESTful services and persistence.
- MySQL Connector/J 8.3.0 for relational database access and schema evolution.
- Spring Security Crypto with BCrypt for password hashing and credential verification.
- Scheduled tasks (Spring Scheduling) automate bid evaluation and project closure.
- Maven orchestrates dependency management, testing, and packaging into an executable JAR.

Frontend (Port 3000)

- React 16.3 with React Router for single-page navigation and modular components.
- Redux and React-Redux manage global state such as authentication and current projects.
- Axios handles HTTP interactions, while Bootstrap-based styling ensures responsive layouts.
- SweetAlert2 provides consistent, accessible notifications for user feedback.

Proxy & Tooling

- Node.js (Express, body-parser, client-sessions) secures client sessions and enforces CORS policies.
- Foreman (‘nf‘) coordinates concurrent execution of the React client and Node proxy during development.
- Dockerfiles for backend and frontend support containerized builds and deployment pipelines.

Features and Functionality

Authentication & User Profiles

- Secure signup and login flows store hashed passwords and establish HTTP sessions for persistent access.
- Users manage personal data, skills, biographies, and contact information, enabling rich freelancer portfolios.
- Profile images are handled through the attachments service, mapping uploaded assets to persistent links.

Project Lifecycle Management

- Employers publish projects with descriptions, required skills, budget ranges, execution periods, and optional documents.
- The system automatically timestamps postings, tracks bid deadlines, and maintains status transitions (‘OPEN‘, ‘CLOSED‘, ‘CLOSED_NO_BIDS‘).
- Employers review portfolio dashboards splitting open opportunities and work-in-progress engagements, including awarded freelancer details.

Bid Management

- Freelancers submit bids with pricing and delivery periods; validation enforces numeric periods and consistent formatting.

- Average bid values are computed per project to drive employer insights and present market competitiveness.
- Employers evaluate bid details enriched with bidder identities, and hiring workflows mark accepted or rejected offers.

Automated Bid Closure

- A scheduled job inspects projects whose bid deadlines passed, automatically awarding the lowest qualifying bid.
- Projects without bids transition to ‘CLOSED_NO_BIDS‘, and all bid statuses update to reflect the award decision.
- Scheduled execution logs notable events and gracefully handles exceptions to preserve platform stability.

User Experience Enhancements

- Dedicated dashboards for employers and freelancers surface relevant metrics, such as bid histories and awarded work.
- Modular React components (e.g., ‘ProjectItem‘, ‘BidProjectForm‘, ‘Dashboard‘) promote reuse and consistent patterns.
- Responsive CSS themes and contextual call-to-action elements drive discoverability and conversion.

Database Design

The relational schema balances normalization and query efficiency. Core tables include ‘user‘, ‘project‘, ‘bid‘, and ‘attachments‘, linked by foreign keys that enforce referential integrity. Projects store employer and freelancer identifiers, budgets, bid deadlines, and statuses; bids record amounts, durations, and lifecycle states. Attachment metadata references stored artifacts, decoupling binary storage from transactional tables. Auditing fields such as ‘createdAt‘ and ‘datePosted‘ support reporting and historical analysis. Indexing on frequently filtered columns (project status, employer ID, user ID) ensures performant dashboards and API responses.

API Endpoints

Backend endpoints follow a RESTful pattern under the Spring Boot service:

- **Authentication:** ‘POST /signup‘, ‘POST /login‘, ‘POST /logout‘, ‘GET /isLoggedIn‘, ‘POST /getprofile‘, ‘POST /updateProfile‘.

- **Projects:** ‘GET /allProjects‘, ‘POST /postProject‘, ‘POST /getProjectById‘, ‘POST /getOpenProjects‘, ‘POST /getProjectDetail‘, ‘POST /getUserProjects‘, ‘POST /hireFreelancer‘.
- **Bids:** ‘GET /allBids‘, ‘POST /postBid‘, ‘POST /getProjectBids/id‘, ‘POST /getUserBidProjects‘.
- **Attachments:** Upload, retrieval, and metadata queries that associate digital assets with projects and user profiles.

All endpoints return uniform ‘ResultObject‘ payloads conveying success messages, error descriptions, and data payloads to the client.

Security Implementation

Security spans multiple layers:

- Passwords are encoded with BCrypt before persistence, and login checks rely on secure hash comparisons.
- Client sessions managed via Express and Spring preserve authenticated state while supporting logout and inactivity invalidation.
- Role-sensitive operations (posting projects, hiring freelancers) validate authenticated identity and enforce server-side checks.
- Cross-Origin Resource Sharing (CORS) rules restrict browser access to trusted origins, mitigating CSRF vectors during development.
- Serialized responses sanitize sensitive fields, and validation annotations prevent malicious payload injection.

Frontend Architecture

The React client adopts a component-driven architecture that promotes reuse and separation of concerns. Container components orchestrate API calls and state transitions, while presentational components render data-rich cards, tables, and forms. Redux reducers manage slices such as authentication status and project collections, and Redux actions encapsulate asynchronous API calls. CSS modules derived from Bootstrap and custom styles tailor the experience for landing pages, dashboards, and modals. SweetAlert-driven feedback loops inform users about bid submissions, authentication status, and system errors.

Backend Architecture

The backend follows a layered MVC pattern:

- **Entities:** ‘User‘, ‘Project‘, ‘Bid‘, and ‘Attachments‘ map to database tables with validation and serialization annotations.
- **Repositories:** Spring Data interfaces (‘UserRepository‘, ‘ProjectRepository‘, ‘BidRepository‘, ‘AttachmentsRepository‘) abstract CRUD operations.
- **Services:** Business logic resides in ‘UserService‘, ‘ProjectService‘, ‘BidService‘, and ‘AttachmentsService‘, ensuring transactional safety and composability.
- **Controllers:** REST controllers coordinate request parsing, validation, and response assembly while logging key events.
- **Utilities:** Shared response structures (‘ResultObject‘) and POJOs (‘ProjectDetail‘, ‘BidDetail‘) standardize data transfer to the client.

User Interface Design

The user interface balances clarity and productivity:

- Landing pages highlight project discovery for freelancers and posting flows for employers.
- Dashboards categorize projects into open opportunities, work-in-progress engagements, and historical records.
- Forms leverage validation feedback, guided placeholders, and inline hints to reduce user error.
- Responsive breakpoints ensure accessibility across desktops, tablets, and mobile devices.

Testing Considerations

A multi-layer testing strategy underpins quality assurance:

- **Unit Tests:** Service-layer tests (JUnit 4, Mockito) verify password hashing, bid aggregation, and scheduler edge cases.
- **Integration Tests:** Spring Boot test slices validate controller endpoints, data persistence, and validation flows.
- **Frontend Tests:** Mocha-based suites (planned) validate component rendering and Redux reducer logic.
- **Manual QA:** Exploratory testing covers project posting, bidding, hiring, and profile updates under varied roles.

Deployment Considerations

- Backend packaging via the Spring Boot Maven plugin yields a self-contained JAR deployable on cloud hosts (AWS EC2, Azure App Service).
- Frontend builds (‘npm run build‘) generate static assets suitable for CDNs or containerized Nginx hosting.
- Environment variables manage database credentials, session secrets, and storage endpoints across environments.
- Docker Compose descriptions orchestrate multi-container local stacks spanning backend, frontend, and database services.

Future Enhancements

- Integrate in-app messaging and milestone tracking to strengthen collaboration and transparency.
- Add payment gateway support for milestone releases, escrow, and invoicing.
- Introduce analytics dashboards summarizing bid trends, freelancer success rates, and project throughput.
- Implement real-time notifications via WebSockets or server-sent events for bid updates and project awards.
- Expand automated testing coverage with Cypress end-to-end scenarios and load testing for peak bid periods.

Conclusion

FreelancerUsingSpringBoot fulfills the core requirements of a modern freelancing marketplace by combining secure authentication, collaborative tooling, and automated bid governance within an extensible architecture. The modular separation of concerns, reliance on industry-standard frameworks, and emphasis on user-centered design position the platform for incremental enhancements such as payments, analytics, and real-time communication. The project demonstrates proficiency in full-stack development, showcases enterprise-grade best practices, and provides a strong foundation for production deployment.