Low Level Document

Hourly Recruitment Application

Document Version Control

Date Issued	**Version**	**Description**	**Author**
15/09/2023	Initial	Added basic requirements and descriptions from HLD	Hardik Arora
22/09/2023	v2	Added Frontent Details	Hardik Arora
25/09/2023	v2.1	Added Backend and Database Descriptions	Hardik Arora
01/10/2023	v3	Added diagrams	Hardik Arora

Contents

Abstract

The MERN Hourly Recruitment Application, constructed on the MongoDB, Express.js, React, and Node.js stack, orchestrates a seamless interaction between clients and freelancers. Users can sign up, post jobs, and bid on projects, while the system manages contracts and payments. The low-level design emphasizes role-based user management, real-time bidding with WebSocket updates, secure contract and billing systems, payment gateway integration, optimized MongoDB data schemas, robust security measures, and scalability considerations. Thorough testing, documentation, and deployment strategies ensure a reliable system, complemented by an intuitive user interface for an optimal experience.

1. Introduction

1.1 Why this Low-Level Design Document?

The Low-Level Design (LLD) Document serves as a crucial blueprint for the detailed implementation of the system outlined in the architecture. Its primary purpose is to provide a comprehensive guide for developers, offering detailed insights into how each component, module, or function should be constructed. The creation of this Low-Level Design Document (LLD) is imperative for the successful development and implementation of the project. It serves as a detailed guide, providing a comprehensive breakdown of the high-level architecture into actionable steps and specifications. The LLD facilitates effective communication among team members, ensuring a common understanding of the design decisions, coding standards, and implementation details. By offering a detailed roadmap for the development phase, this document enhances collaboration, reduces ambiguity, and acts as a reference point for developers, ultimately contributing to the overall efficiency and success of the project.

1.2 Scope

The scope of this Low-Level Design Document covers the detailed design and implementation guidelines for the following aspects of the project:

- User Authentication and Registration
- Client and Freelancer Dashboards
- Job Posting and Bidding System
- Contract Management
- Billing Calculation
- Database Schema (MongoDB)
- Security Measures
- Error Handling
- Testing Strategies
- Documentation Guidelines
- Scalability Considerations
- Monitoring and Logging
- Deployment Strategies

The document aims to provide a granular view of each component, ensuring that developers have clear instructions on how to implement these features in line with the high-level architecture.

1.3 Constraints

The development of the project is subject to certain constraints that need to be considered throughout the implementation phase. These constraints include:

- Limitations like unavailability of a paid web service like AWS for hosting and a registered domain compromises the security of the software.
- The requirement of integrating a payment gateway could not be satisfied due to lack of resources.

1.4 Risks

Dependencies on third-party providers and potential compatibility issues pose schedule and integration risks. Potential risks for the project include challenges in implementing the payment gateway, which may result in delays or functionality gaps. Mitigation involves rigorous testing of alternatives, early engagement with providers, and contingency plans for possible delays.

1.5 Out of Scope

The following aspects are considered out of scope for this Low-Level Design Document:

- Future Enhancements: Features or functionalities intended for future releases.
- Non-Functional Requirements: Aspects like performance tuning and optimization not explicitly addressed in this document.

2. Technologies Used

React (Frontend):

Employed for building dynamic and responsive user interfaces.

Utilizes React components for client-side functionalities, including job posting, bidding, and dashboards.

Node.js (Backend):

Powers the server-side logic and manages the application flow.
Facilitates the development of robust APIs.

Express.js (Backend):

Used for creating and managing backend APIs.
Handles HTTP requests and responses, ensuring smooth communication between the frontend and backend.

MongoDB (Database):

Chosen as the NoSQL database for its flexibility in handling diverse data structures.
Stores user profiles, job details, bids, contracts, and billing information.

Secure Coding Practices:

Adhered to for ensuring overall application security.
Includes best practices for preventing common vulnerabilities and ensuring data integrity.

HTTPS (Security):

Implemented for secure data transmission between the client and server.
Enhances the confidentiality and integrity of data exchanged within the system.

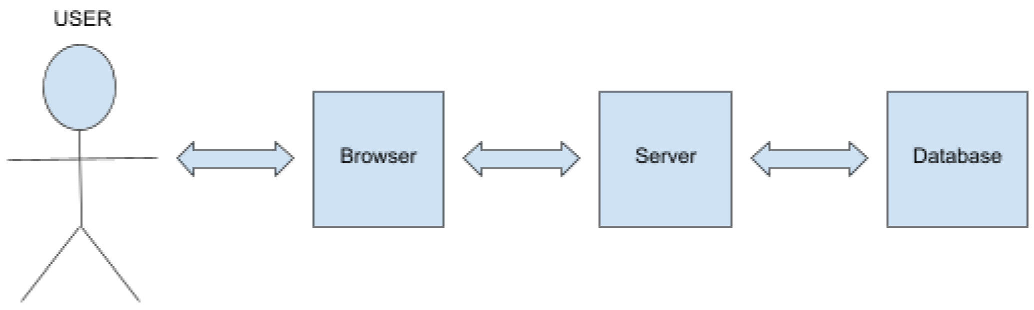
Git (Version Control):

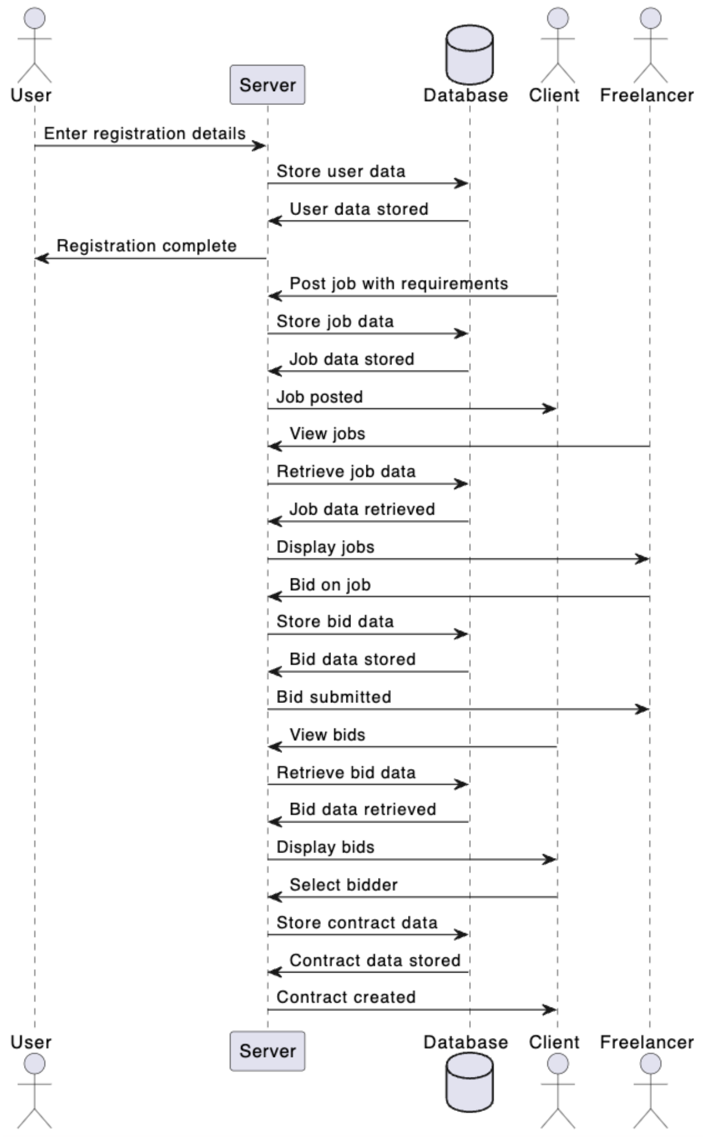
Utilized for version control, enabling collaborative development and tracking changes in the codebase.
Facilitates branching, merging, and code review processes for efficient collaboration among developers.

SCSS (Styling):

Implemented for styling purposes, extending the capabilities of CSS.
Enables the use of variables, mixins, and nested styles for more maintainable and modular stylesheets.
Enhances the efficiency of styling workflows and promotes a consistent and scalable design approach.

3. User I/O workflow





4. Presentation Tier (Frontend):

4.1 User Authentication and Registration

Frontend components for user authentication and registration forms.
React for dynamic user interfaces, allowing users to sign up as clients or freelancers.

4.2 Responsive client dashboard using React components

Enabling clients to post jobs, manage contracts, and view billing information.

4.3 Dedicated freelancer dashboard for viewing available jobs and managing bids.

Displaying jobs using React components with detailed requirements.

4.4 Bidding System

Bidding system with React components for freelancers to submit bids.

4.5 Contract Agreement and Billing:
Contract agreement and billing calculations for each project.

5. Application Tier (Backend):

5.1 User Management:

Node.js APIs(passport) for user authentication, registration, and profile management.
Express.js for handling HTTP requests and responses.

5.2 Job and Bid Management:

APIs for posting, updating, and deleting jobs.
Bid management APIs for freelancers to submit bids and clients to review them.

5.3 Contract Management:
APIs for finalizing contracts.

5.4 Billing System:

Billing system APIs to calculate charges based on agreed-upon rates and timeframes.

6. Data Tier (Database):

MongoDB Schema:

MongoDB database schema for storing user profiles, job details, bids, contracts, and billing information.

1. User:
   first_name, last_name, type(client,freelancer/admin), username, password, createdAt, skills
2. Project:
   title, description, technologies, clientId, duration, status, freelancerId, accepted_bid, approved_submission, submission_url
3. Submission:
   clientId, freelancerId, projectId, sourceCodeUrl, demoUrl, hostedUrl, accepted_at
4. Bid:
   clientId, freelancerId, projectId, hourly_rate, accepted_at

7. Deployment

The deployment will be accomplished separately for frontend and backend using Netlify and Railway. Netlify, chosen for its robust hosting capabilities, employs automated CI/CD pipelines for efficient deployment upon code changes. Its global CDN ensures optimal user experience worldwide. Railway, on the other hand, serves as a reliable platform for backend deployment, particularly for managing databases. This dual deployment strategy provides scalability, reliability, and a user-friendly experience for both frontend and backend components. Regular monitoring and updates contribute to the ongoing stability and performance of the deployed web application.

8. Test Cases

S. No.	Test Case Description	Test Input	Expected Output
1	Successful User Login	Valid username and password	Successful login
2	Unauthorized Access	Invalid or expired JWT token	Access denied
3	Successful Job Posting	Valid job details and requirements	Job posted successfully, visible in the job listing.
4	Empty Job Post	Posting a job without specifying details	Validation error with a request to provide necessary details.
5	Successful Bid Submission	Valid bid details, including quote and time period	Bid submitted successfully and visible in the list of bids.
6	Bid with Invalid Quote	Bid with a non-numeric quote	Validation error and request for valid numeric quote.
7	Project Completion	Post repository link	Request successful, Final cost is displayed
8	Project completion with Empty Link	Clicking on complete without link	Request Failed, and request for link.
9	Admin User Deletion	Admin deletes an existing user	User is removed from the system
10	Admin Project Deletion	Admin deletes an existing project	Project is removed from the system

9. Key performance indicators (KPI)

Evaluation Focus:

The success of the web-based platform will be assessed through key performance indicators (KPIs).

User Engagement Metrics:

Monthly Active Users (MAU) and Daily Active Users (DAU) will be tracked to gauge user interaction frequency.

User Experience Efficiency:

Response time, latency, and error rates will be monitored to ensure a seamless user experience.

User Retention and Reliability:

User retention rate will indicate the platform's long-term appeal.
Platform uptime will reflect the reliability of the system.

Ensuring Sustained Success:

The comprehensive evaluation of user engagement, transaction efficiency, and overall satisfaction will ensure sustained success and user contentment.