# High Level Document

Hourly Recruitment Application

# Document Version Control

| Date Issued | Version | Description | Author |
|---|---|---|---|
| 01/09/2023 | Initial | HLD — V1.0 | Hardik Arora |
| 07/09/2023 | Updated | HLD — V1.1 | Hardik Arora |

# Contents

## Abstract

The High-Level Design (HLD) outlines a web-based platform connecting clients and freelancers, streamlining the entire process from job posting to contract completion. Users register as clients or freelancers, with clients able to post jobs alongside detailed requirements. Freelancers then bid on these jobs, submitting quotes and proposed timelines.

Upon client selection of a freelancer, both parties enter into a mutual agreement, triggering the commencement of a contract. Unlike traditional secure transaction handling, payments are made directly between the client and freelancer outside the platform. The system focuses on robust security measures in terms of user authentication and data protection.

The platform offers user-friendly interfaces, facilitating efficient workflows for seamless collaboration in the gig economy. Once the job is successfully completed and the client is satisfied, the platform facilitates the conclusion of the contract. This HLD aims to provide a secure, efficient, and user-friendly environment for collaborative interactions between clients and freelancers in the evolving gig economy without direct involvement in transaction handling.

# 1. Introduction

## 1.1 Why this High-Level Design Document?

This High-Level Design Document serves as a foundational guide for the development of a MERN (MongoDB, Express.js, React, Node.js) based web application connecting clients and freelancers. It elucidates the overarching architecture, key functionalities, and design principles essential for the successful implementation of the project. The document aims to provide a comprehensive understanding of the system's structure and rationale, serving as a reference for developers, stakeholders, and other project contributors.

## 1.2 Scope

The HLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

# 2. General Description

## 2.1 Product Perspective

The proposed web application exists within the context of a dynamic online marketplace connecting clients and freelancers. As part of the larger gig economy, it serves as a platform for seamless collaboration, enabling clients to post jobs and freelancers to bid on opportunities.

## 2.2 Problem statement

To create an web solution for collaboration using MERN stack and implement the following use cases

- Designing of a web application “Hourly Recruitment Application” to allow anyone to sign up as a recruiter or as a worker.
- The clients can post a job along with the necessary job requirements.
- The freelancers can see the job posted and can start bidding with their quotes and time period they require to finish the task.
- The client can then see all the bids and select any one bidder and upon mutual agreement the contract will begin.

## 2.3 Proposed Solution

The solution proposed here is a MERN (MongoDB, Express.js, React, Node.js) based web application that addresses the identified challenges in the gig economy. Clients can easily post jobs with detailed requirements, and freelancers can bid on these opportunities. The system ensures transparency, security, and a user-friendly interface throughout the entire process, from job posting to contract conclusion. By integrating the strengths of the MERN stack, the proposed solution aims to provide a comprehensive and reliable platform for clients and freelancers.

## 2.4 Further Improvements

Future iterations could introduce features such as real-time collaboration tools, AI-driven job matching, and enhanced analytics for both clients and freelancers. These improvements are envisioned to elevate the platform's capabilities, providing users with a more tailored and efficient experience within the gig economy.

## 2.5 Technical Requirements

**User Registration and Authentication**:
Users should be able to create accounts with unique credentials. The system must provide secure authentication mechanisms to protect user data.

**Job Posting and Bidding:**
Clients should have the ability to post jobs with detailed descriptions and requirements. Freelancers should be able to browse and bid on posted jobs, providing quotes and estimated timelines.

**Contract Initiation and Management:**
A mechanism for mutual agreement between clients and freelancers, triggering contract initiation. Transparent and accessible contract management features for both parties.

**User Profiles:**
Users must have the capability to create and manage profiles, showcasing relevant skills and experience. Profiles should include a history of past jobs and contracts.

**Security Measures:**
Encryption protocols to safeguard user data and sensitive information.

Protection against common web vulnerabilities, such as SQL injection and cross-site scripting.

**Responsive User Interface:**
Responsive design to ensure a seamless user experience across various devices and screen sizes.
Intuitive and user-friendly interfaces for both clients and freelancers.

## 2.6 Tools used

The tools used in the development of this website encompass a range of technologies and frameworks. The MERN (MongoDB, Express.js, React, Node.js) stack forms the core foundation:

- **Frontend Framework (React):**
  React for building interactive user interfaces.
- **Backend Framework (Node.js and Express.js):**
  Node.js for server-side JavaScript execution.
  Express.js as the backend framework for handling server-side logic and routing.
- **Database Management (MongoDB):**
  MongoDB as database to store and manage data efficiently.
- **Version Control (Git):**
  Git for version control to track changes, collaborate, and manage the project's source code.
- **Development Environment:**
  Visual Studio Code was used as the IDE.
- **Styling (SCSS):**
  CSS for basic styling and layout.

## 2.7 Constraints

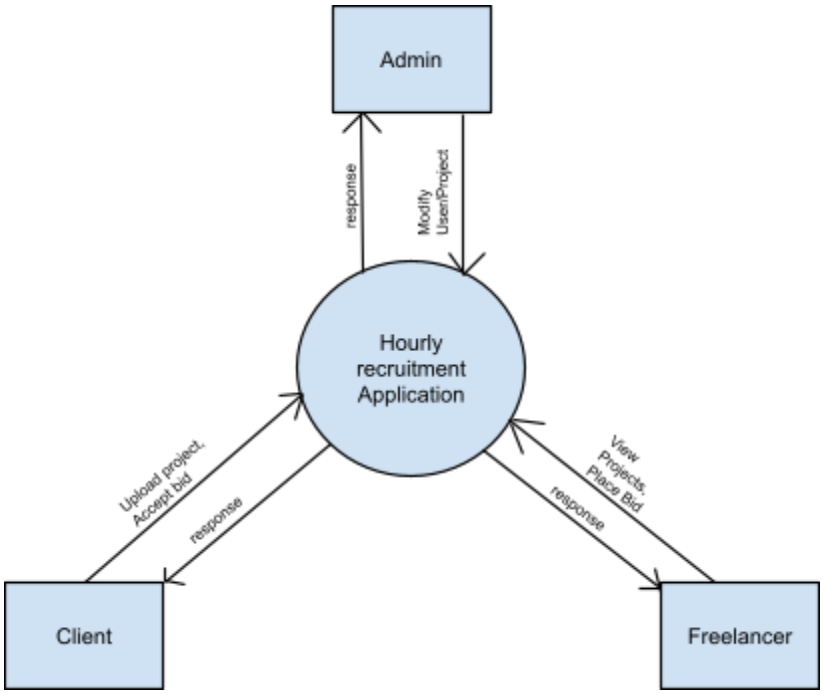
A consistent and intuitive user interface ensures seamless navigation, while performance optimization focuses on quick load times and responsiveness. Cross-browser compatibility and mobile responsiveness guarantee a uniform experience across devices and platforms. Security measures prioritize robust protocols like HTTPS, secure authentication, and data encryption. Scalability is a key consideration in the website's architecture, accommodating potential growth in user traffic. Regulatory compliance, including data protection and privacy regulations, is ensured, and content moderation mechanisms maintain the appropriateness of user-generated content. Adherence to web accessibility standards promotes inclusivity for users with disabilities.

## 2.8 Assumptions

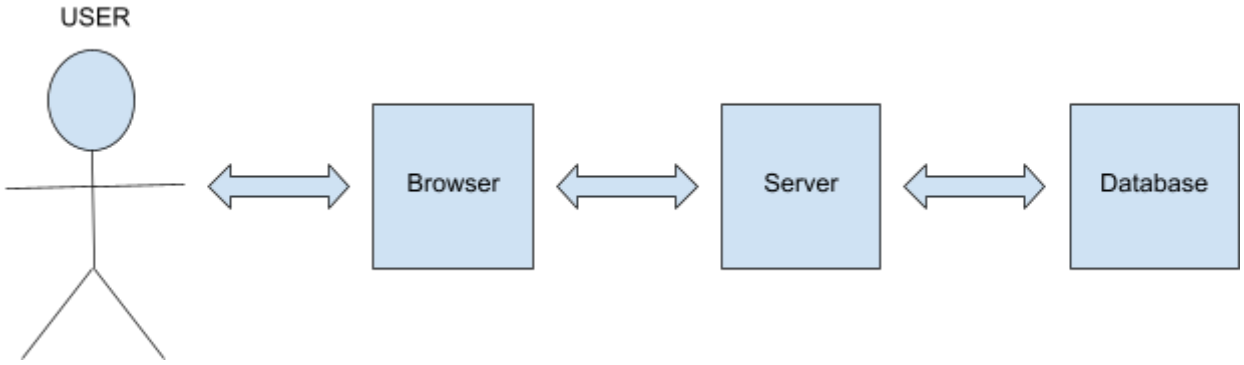
Technologically, we anticipate compatibility across various browsers and devices, ensuring consistent performance. Users are assumed to consistently provide accurate content, adopt new features promptly, and manage privacy settings responsibly.

# 3. Design Details

## 3.1 Process Flow



### 3.2 Data Flow




### 3.3 Error Handling

The user will be able to see logs on the browser console providing clear error messages, guiding users on corrective actions. The error handling system is designed to minimize disruptions, enhance user satisfaction, and streamline troubleshooting processes during development and production.

## 4. Performance

The application employs techniques such as code optimization, caching strategies, and content delivery network (CDN) integration to enhance speed and overall responsiveness. Regular performance monitoring ensures proactive identification of bottlenecks, allowing for timely optimizations and a consistently high-performing user experience.

### 4.1 Reusability

The code written in frontend or any specific react component should have the ability to be reused with no problems.

### 4.2 Application Compatibility

The Application will be compatible with any modern web browser with javascript support and active internet connection.

### 4.3 Resource Utilization

CPU resources are provisioned by the deploying service(in this case railway/render/vercel) and those resources will be used anytime when the API is called. API calls are triggered by the frontend whose resources are provisioned by the frontend

content distribution network(in this case Netlify). All access to the site will trigger the use of frontend resources, but only API calls will trigger the use of backend resources.

### 4.4 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.

## 5. Key Performance Indicators

The success of the web-based platform will be evaluated through key performance indicators (KPIs) covering user engagement, transaction efficiency, and overall user satisfaction. Metrics such as Monthly Active Users (MAU) and Daily Active Users (DAU) will gauge user interaction frequency, while the conversion rate from bids to contracts will assess the platform's effectiveness. Transaction success rate and a Customer Satisfaction Score (CSAT) will measure payment system reliability and user contentment.

Efficiency indicators, including response time, latency, and error rates, will ensure a seamless user experience. The conversion funnel metrics will identify and optimize potential bottlenecks in the user journey. User retention rate and platform uptime will reflect the platform's long-term appeal and reliability. Regular analysis of these KPIs will inform strategic decisions and continuous improvement efforts, ensuring sustained success and user satisfaction.

## 6. Conclusion

In summary, this high-level design document charts the course for a dynamic web-based platform catering to clients and freelancers in the gig economy. Rooted in the MERN stack, the project promises a seamless user experience, emphasizing efficient error handling and optimal performance.

Strategic deployment on Netlify and Railway ensures scalability and efficiency for both

frontend and backend components. Key performance indicators (KPIs) have been defined to assess user engagement, transaction efficiency, and overall satisfaction.

As we transition to implementation, this design provides a clear roadmap, emphasizing reusability and resource optimization. With a focus on adaptability and continuous improvement, the platform is poised to meet the dynamic demands of the gig economy, offering a reliable and user-centric solution. The outlined KPIs will guide ongoing success and user satisfaction throughout the development lifecycle.