# TECHNICAL REPORT

**Instructor**: Thanh Nguyen, Ph.D.,
**Course**: Further Web Programming
COSC2769
**Group**: 5
**Tutorial Session: 1**
**Project manager**: Hau Le (s3741297)
**Group members**:

Nguyen H Vu (s3881101)
Khanh Giang (s3878182)
Nam Nguyen (s3873792)
Vu Pham (s3701522)
Khoa Tran (s3863956)

## Contents

## Abstract

This technical report will delineate on the functional and non-functional requirements, software architecture, UI/UX demonstration, and some other technical specifications relating to the website application BeeLancer, a website where freelancers in the IT fields can find contracts and potential employers. The software is built around the MERN stack structure,

**Keywords:** BeeLancer, RMIT, freelancer, IT fields, web-based application, technical report, Heroku, MERN stack.

## I. Project Description and Functional Block Diagram

### 1. Logo

This project is called “BeeLancer”, a play on the word “Freelancer”. The usage of the bee imagery is due to the industrious and busy nature of the bees, mirroring those of the IT professionals, especially in the freelance world.

### 2. Objectives

Compare to the proposed objectives outlined in the project proposal, those remaining at the end of the project are reduced in scope but remain the same in each objective’s nature. This project aims to provide IT professionals and other freelancers with:

- **Help IT professionals establish their portfolio**: a platform where IT professionals wanting to freelance post and advertise their profile, skills and job aspirations.
- **Personal account management as both freelancers and potential employers**: BeeLancer allows its users to create accounts as either a freelancer looking for a job or an employer offering contracts.
- **Job Seeking and contact:** BeeLancer allows freelancers to view, search, and accept job offers posted on its job list interface. On the other hand, it allows employers to post their contracts for prospective employees to find.

- **Job application via information details:** BeeLancer allows freelancers to view contact details of employers in each contract and contact them personally to arrange for interviews or signing of contracts.

## 3. Notable Product properties

### 3.1. MERN stack application

MERN stack applications are built around the four main technologies of its initialism namesake: MongoDB for database, Express(.js) – a Node(.js) framework, React(.js) – a client-side JavaScript framework, and Node(.js) – the predominant JavaScript web server.

React(.js) is the client-side application which users interact with. Node and Express make up the middle (server-side) application. Finally, MongoDB is the Database to store and pull data from. The following diagram illustrates how the connection between client-side and server-side, as well as the database works:

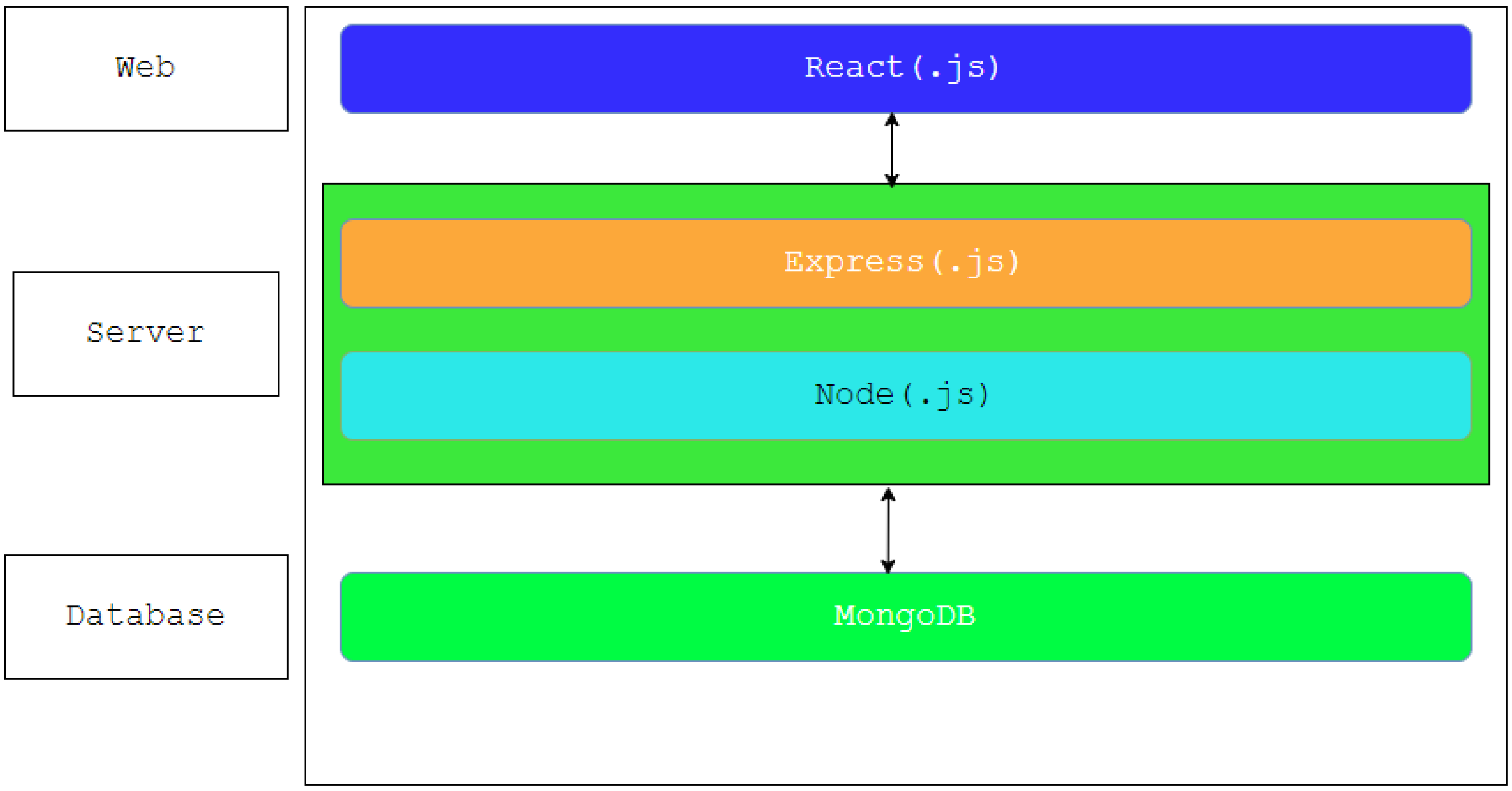


*Figure 1: MERN stack structure*

**React(.js)**: the top tier of MERN stack applications, where users will interact and send and receive requests and responses, is React(.js). React(.js) is the declarative JavaScript framework that creates dynamic client-side websites and applications as HTML. React(.js) breaks down complex interfaces into simple components, connects

BEELANCER to the data on the client-side server via fetch API functions, and render them as HTML.

**Express(.js) and Node(.js)**: The next level after the web stack is the server-side stack, which contains Express(.js) and Node(.js). Express(.js) has many powerful models for URL routing and pathing for RESTful APIs, meaning it matches an incoming URL with a server function. In addition, Express(.js) handles HTTP requests and responses. By creating POSTs and GETs requests from BEELANCER's frontend, React(.js), BEELANCER can connects to Express(.js) functions that power its system. These functions, in turn, use MongoDB's drivers via async and promises, to access and update data in its MongoDB database.

**MongoDB**: since BEELANCER requires storing and access of real data generated by the users and the application, MongoDB is integral to manage said data. JSON documents create React(.js) are sent to Express(.js) server where they are processed and once validated can be stored directly in the MongoDB for further retrieval.

## 3.2. Functional and non-functional requirements

### Functional requirements

Given the extremely unabundant time constraints we have to adhere to with only 9 weeks of available working days (self-development, public holidays and unforeseen obstructions already excluded) as well as our own limitations in knowledge, especially being proficient in the frameworks used in this project like ReactJS, NodeJS, ExpressJS and MongoDB, we have discussed the scope of our web application thoroughly with one another in terms of simplifying things to the minimum and collectively agreed on a most reduced version of a freelancer community app (job-seeking website). In total, our group have identified **5 primary functionalities** within our full-stack system, which are the following:

- **<u>Functional requirement 1 – Manage Account</u>:** Beelancer must allow its users (clients and freelancers) to perform several operations regarding their accounts such as creating, editing, deleting, logging in/out and viewing.

| F.R 1 – Manage Account | Description |
|---|---|
| F.R 1.1 – Create an account | For first time visitors (guest users) of Beelancer, they will be provided an option to register for a new freelancer/client account to gain access to all features using a valid email account along with other personal information (name, date of birth, address, etc.). |
| F.R 1.2 – Log in/Log out of account | Once their account has been successfully created, they are able to sign in to the system as an existing user and to maintain that state throughout the entire UI, we have included a |

| | |
|---|---|
| | unique ID token that the system will commit to memory while their browsing about. |
| F.R 1.3 – Edit account | Users (either freelancer or client) have the ability to update any information present in their account that may have changed since the creation of that respective account, and these modifications will be saved into the database as the new profile. |
| F.R 1.4 – Delete account | If they choose to no longer be a member of the Beelancer community, they have the option to remove their account completely from the database via a button. The system will confirm this request and deactivate their account accordingly. |
| F.R 1.5 – View account | Users can easily navigate to their own profile page, or check out another client/freelancer account via browsing. From there, they can see all the information that the account owner made public for viewing. |

- **Functional requirement 2 – Post/Manage a Job Offer (Clients only):** Companies or individuals that are on the lookout for aspiring talents with high expertise to help them work on a project or complete a task in a remote setting can create a job description page for the position that they need to be filled. This element contains all the necessary briefings that the freelancers require like job title, deadline, budget or secondary details to start making propositions toward the client in question.

| **F.R 2 – Post/Manage a Job Offer** | **Description** |
|---|---|
| F.R 2.1 – Post a new job offer | Clients that have already made an account are given an option to make a job post for the work they want to achieve completion, and this is done by filling out a form containing the necessary information of the job they would like to disclose. These posts are visible to all active Beelancers looking to find work. |
| F.R 2.2 – Manage job offer status | In case the client has received multiple offers from different freelancers/talents, they can perform operations such as denying a proposal (which will leave the job status as open), accepting a proposal (which will change the job status to active) or remove the listing from the system entirely. |

- **Functional Requirement 3 – Browse for available Job Offers (Freelancers only):**

A staple feature for all labor searching websites, Beelancer provides an intuitive and seamless method for its freelancers to find their next gig via a search bar located on the header of each page. Using simple pairs of keywords, a user can retrieve a list of many potential job positions relevant to the query they tried, and these items are organizable through filters/sorts and implement pagination to avoid over-scrolling since each page only displays a maximum of 10 jobs.

| **F.R 3 – Browse for available Job Offers** | **Description** |
|---|---|
| F.R 3.1 – Browse all existing job offers | On the home view of the Beelancer website, freelancers can locate the search bar on the top-left corner and type in the job they want to work in, then the system redirects them to another view indicating the found results based on their keywords. If there are more than 10 items, Beelancer automatically splits it into a new page. |

- **Functional Requirement 4 – Apply for an open Job Offer (Freelancers only):** While going through the long list of the various offers they recently found, freelancers have the ability to send a proposal to any particular job of their own choice and along with this provide the required fields that clients need to evaluate the compatibility between themselves and the client they intend to collaborate with. This information may encompass their average working schedules, their quickness of delivery and most importantly their reliability (reflected through ratings).

| **F.R 4 – Apply for an open Job Offer** | **Description** |
|---|---|
| F.R 4.1 – Send a proposal to an open job offer | On finding the idea position to commence working, freelancers have the ability to write up a short document to officially make a proposition to the client, providing all the necessary information from which the hirer can decide on accepting/rejecting their offer. In either scenario, the freelancer will be notified of the status. |

- **Functional requirement 5 – Rate a freelancer on overall performance (Clients only):** Once a job has been finished between a freelancer and a client, then the latter is prompted with a feature to assess how the entire process played out, with criteria namely freelancer responsiveness, quality of work, professionalism, attitude and so on. The main indication of the rating is in the form of stars (out of a potential five) and this is subsequently used to determine whether a freelancer has a better reputation than one another, and thereby by more likely to receive an offer in future endeavors.

| **F.R 5 – Rate a freelancer on performance** | **Description** |
|---|---|

| | |
|---|---|
| F.R 5.1 – Make a rating/review for freelancers | Clients are allowed to leave an in-depth review of their experience with any freelancer that has previously work for them in a job, with the simplest method being a star rating and they are also given the opportunity to write more descriptive remarks or even recommendations if the gig went smoothly. |

### 3.3. Non-Functional Requirements

Working alongside non-technical stakeholders require other aspects than what an application is supposed to do. Besides functionalities that define a software, business aspects are also crucial to the success of a product. Therefore, it is integral that non-functional requirements be met. Compared to the original proposal, the versatility of non-functional requirements of BEELANCER now is much reduced, but exist, nonetheless.

#### 3.3.1. Security

To ensure only registered users can use BEELANCER's features, the registration process a password. Once the guests visiting BEELANCER registration page provide their email address/ username, the system will ask that create a password and validate each field. Once the correct formatted fields are inserted, the verification process is completed. There are two other criteria users must meet for security:

- BEELANCER should allow user to either log in as a freelancer or an employer.

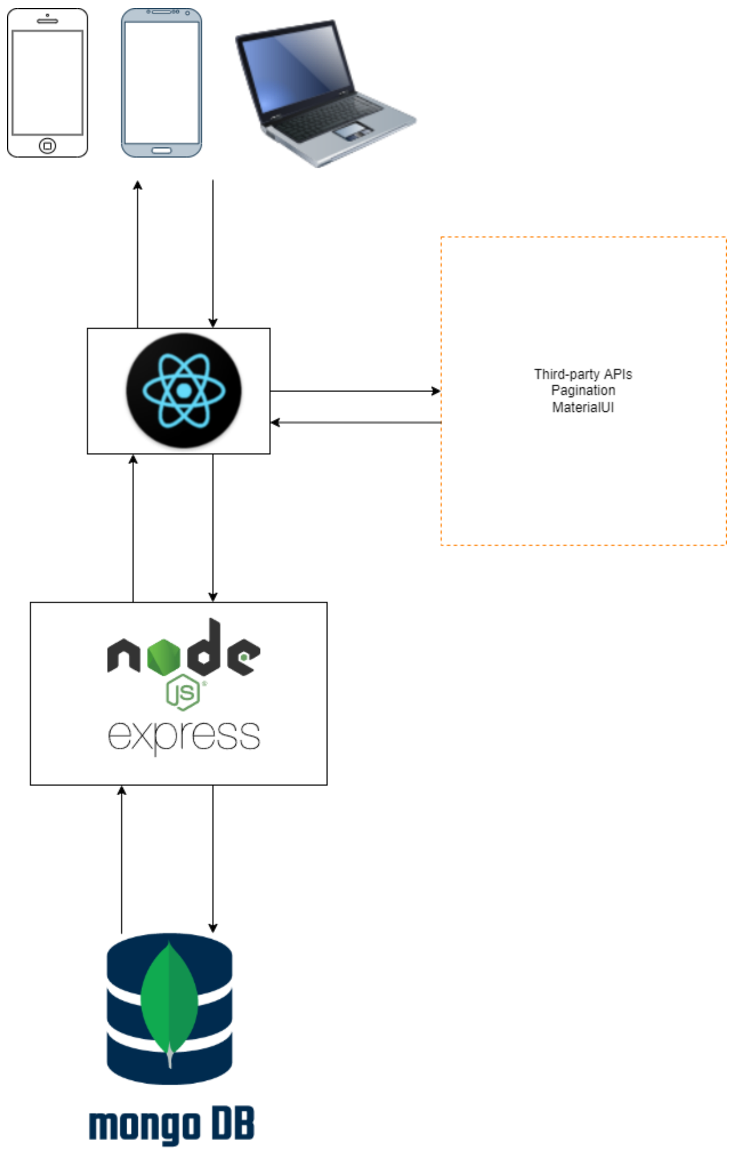
#### 3.3.2. Mobility

BEELANCER is responsive, meaning it can be viewed with adjustable formats on different devices. Moreover, since this is a web-based application with a database, users' information is stored in the cloud so users can access from any of their devices, with the initial verification after registration of account.

Moreover, BEELANCER is developed independent of prerequisites for any operating systems. In other words, on computers, BEELANCER should be accessible web-wise on both Mac OS and Windows. That being said, limitations may apply where Linux is involved since no test has been run on Linux OS.

## 4. Software Architecture

The software Architecture of BEELANCER represents its MERN stack.

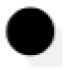


Since BeeLancer is a MERN stack application, it adheres to the basic skeletal structure of client-server-database stacks shown in Figure 1. However, BeeLancer utilizes third-party APIs such as React-Paginate to manage and section its lists display and Material UI to help with frontend design features more expediently. Moreover, Node(.js) and Express(.js) are used to create CRUDs, search and filter, and other APIs for the application's backend. Finally, a free version of MongoDB is utilized to store, fetch, and manage data.

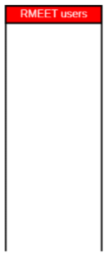
## III. Technical Specifications

### 1. Activity Diagram

After visualizing the use cases denoting the main functionalities of BEELANCER, a clear mental picture of a session of interaction between the user and BEELANCER system is needed. An activity diagram can accentuate the sequential nature of different functionalities of BEELANCER and how user can fully perform them from start to finish. (For better imagery, see Appendix). For BEELANCER's activity diagram, see the follow legends to better understand what the diagram is describing:



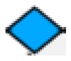
**Initial (start of session)**: this denotes the start of an activity session between user of BEELANCER and its system's components or other actors.



**Swimlane:** this denotes the actor/component of the system where all the actions inside it are carried out by that actor or component.

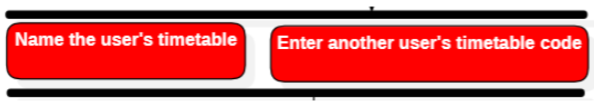
**Action**: an action done within a swimlane which denotes said action is carried out by that swimlane's actor or component.

**Decision node**: this decision node denotes a crossroad where it is followed by an array of actions where the activity flow will only choose to go for one of the options.

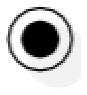


**Merge node**: this merge node denotes a rejoining of different alternate option paths, which all converge on this node to move on to the next common steps.

**Connector:** this connector separates the big diagrams into smaller sections due to space constraints or simply to make it more easy to follow.

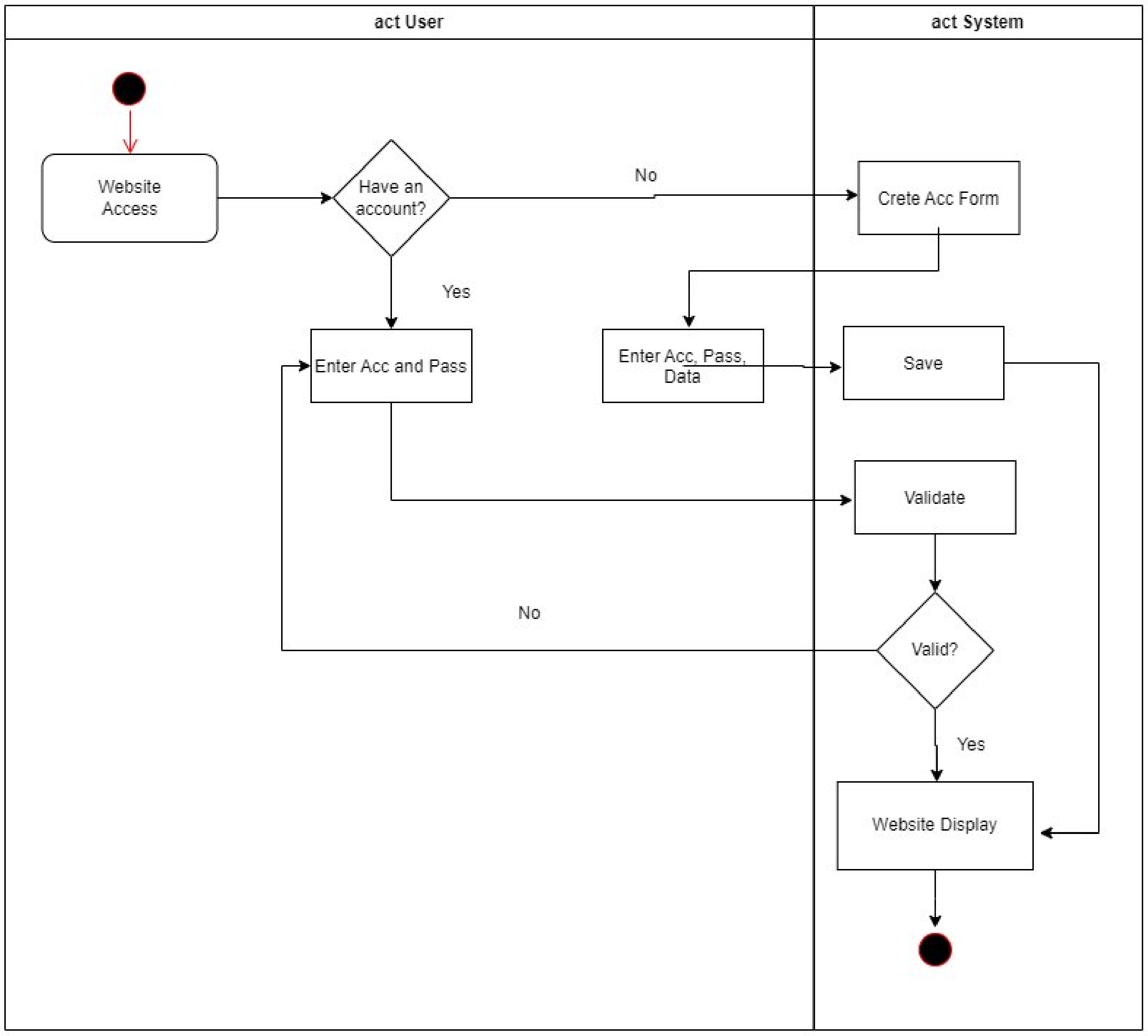

Name the user's timetable
Enter another user's timetable code

**Fork and Join lines**: the fork and Join lines encompass a number of actions within it, which can be done in any order. Unlike the decision node where only one option can be chosen, fork and join lines require their swimlane to perform all of the actions within them before moving on to the next step.



**Final (end of session) node:** this final node denotes the end of the activity session.

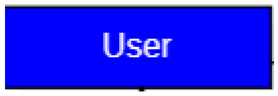
With the legends, the following activity diagram can be interpreted as follows:



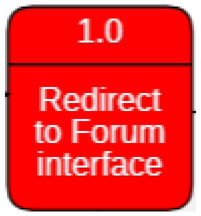
The abovementioned diagram showcases an example activity diagram of a login session. BeeLancer users access the website domain, if they have an account they can login. If not, they can create account, wait for the system to validate and update, and finally be directed to the logged in landing page.

## 2. Data Flow diagrams

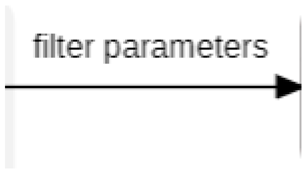
One of the most important aspects of understanding how data is handled throughout the application process is by visualizing it through dataflow diagrams. To better navigate the components and flows between them, a legends is provided below:



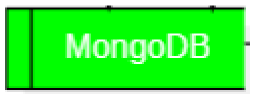
**External Entity:** also known as actors, sources, or sinks, and terminators, these produce and consume data flows between the entity and the system. In other words, external entities are the input and output of the diagram.



**Process:** this square divided into two rows here signifies a process that transforms or changes data flows. Since they transform data flows, they must have an input and output flow. Moreover, since this is a process, it is usually denoted with a verb phrase. The order of the process is denoted as 1.0, 2.0, etc.



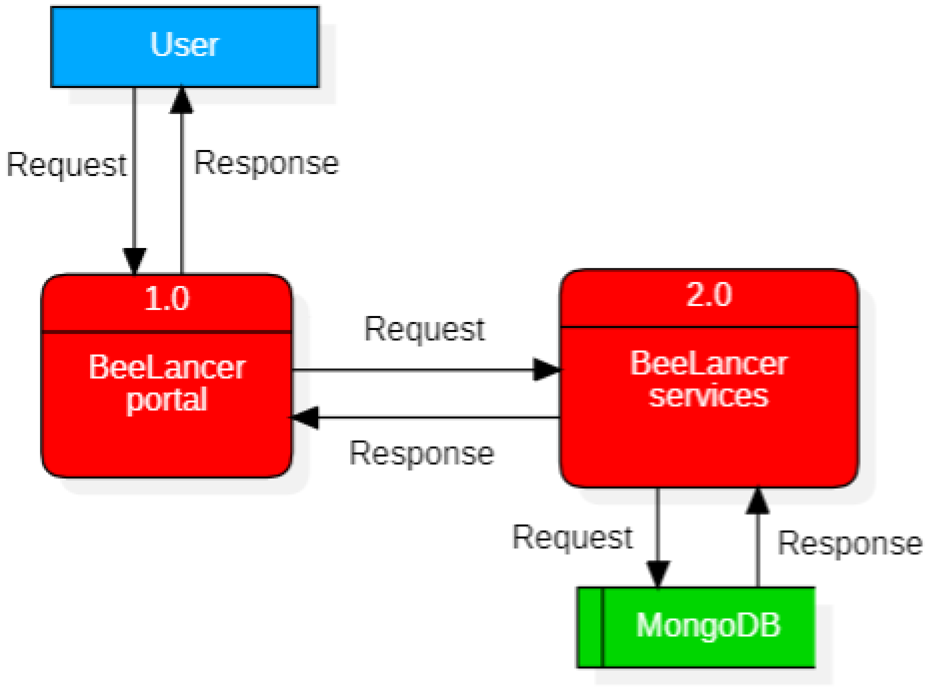
**Data flow**: an arrow with a label attached to it denotes the flow of data flows in the dataflow diagram. The arrow symbol represents the direction of flow while the label, in the form of noun phrases, denotes types and name of data being transported.



**Data store**: data stores does not generate any operation, but they often retain data for later use.

The following Dataflow diagrams are only level 0, meaning they showcase only the upmost generic flows of data and their associating processes. Before diving into the level 0s, it is important to take a look at the **context diagram**, which is a form of generalized dataflow diagram that gives context to the entire system as a whole and what primary information is being flown around.

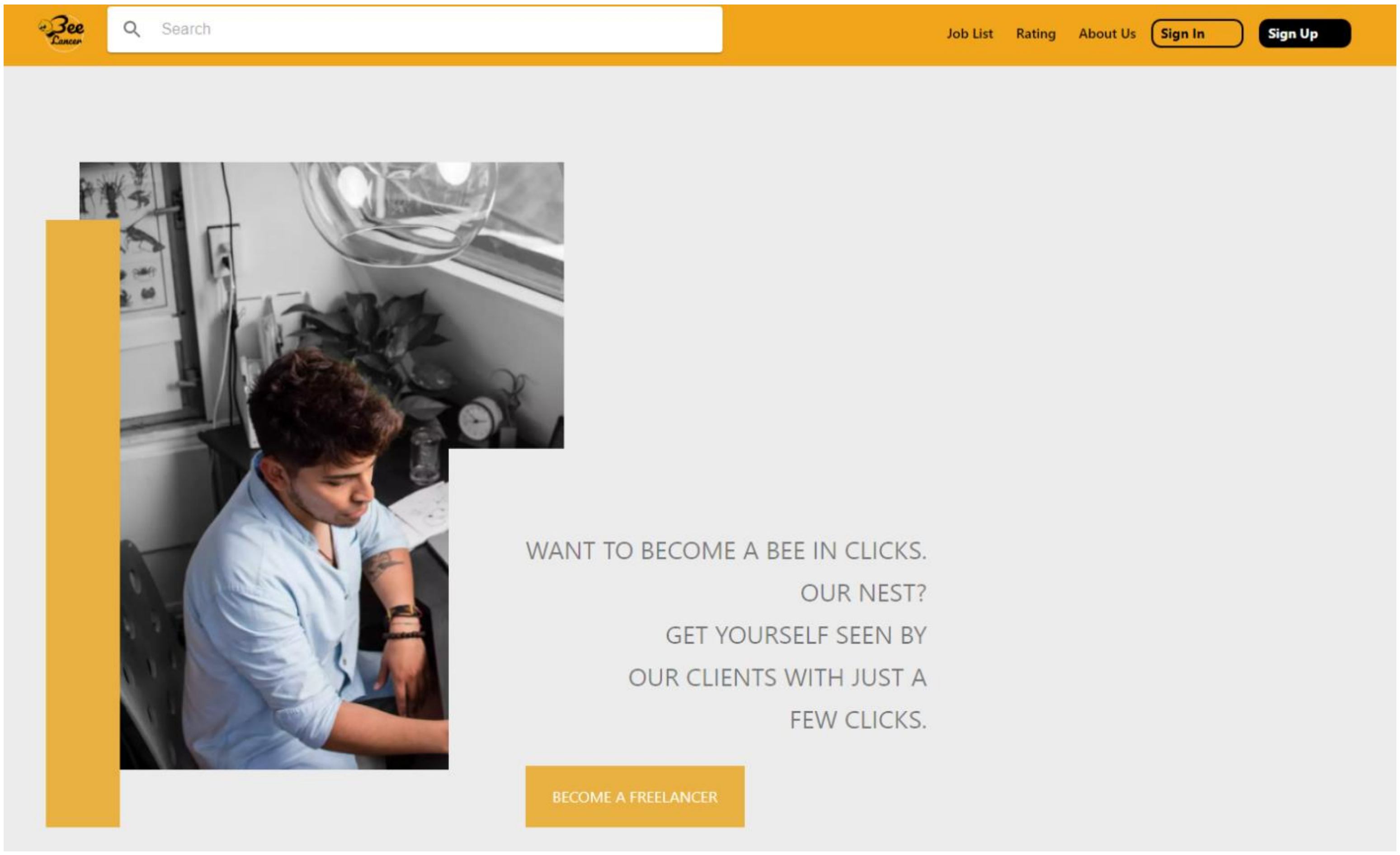
The context diagram below provides the outermost structure of how data move between different entities and processes of BEELANCER. Adhering to the three-layer structure shown in the software architecture, whenever the user entity wants to use a function, a request is sent through the client-side interfaces, or BEELANCER portal services, which, in turn, send another request to the server-side BEELANCER services. If the request involves fetching data, then another request moves to the MongoDB database to fetch said data and a response is generated, moving all the way back to the user.



The above data flow diagram above is the **context diagram,** which is the utmost generic diagram showcasing what the entire system will handle the flows of data. BeeLancer will take in request inputs from user, transmit it to its system backend and to the database and a response is invoked all the way back to be displayed to the users

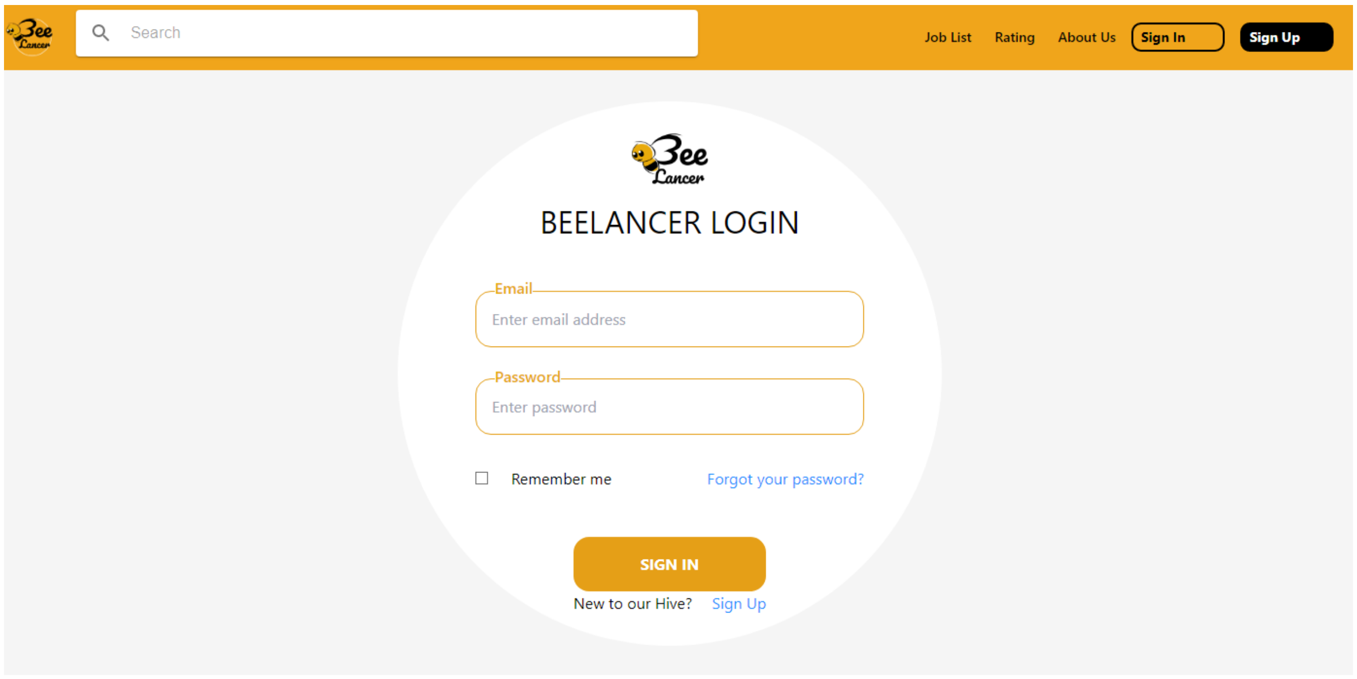
## 3. UI/ UX result

### Landing page



**This** is the landing page when new guests visit BeeLancer domain. It showcases a message “become a freelancer” in the form of a button. The navigation bar has the options of either logging in or signing up.

### Login and sign up:



BEELANCER LOGIN

Email: Enter email address

Password: Enter password

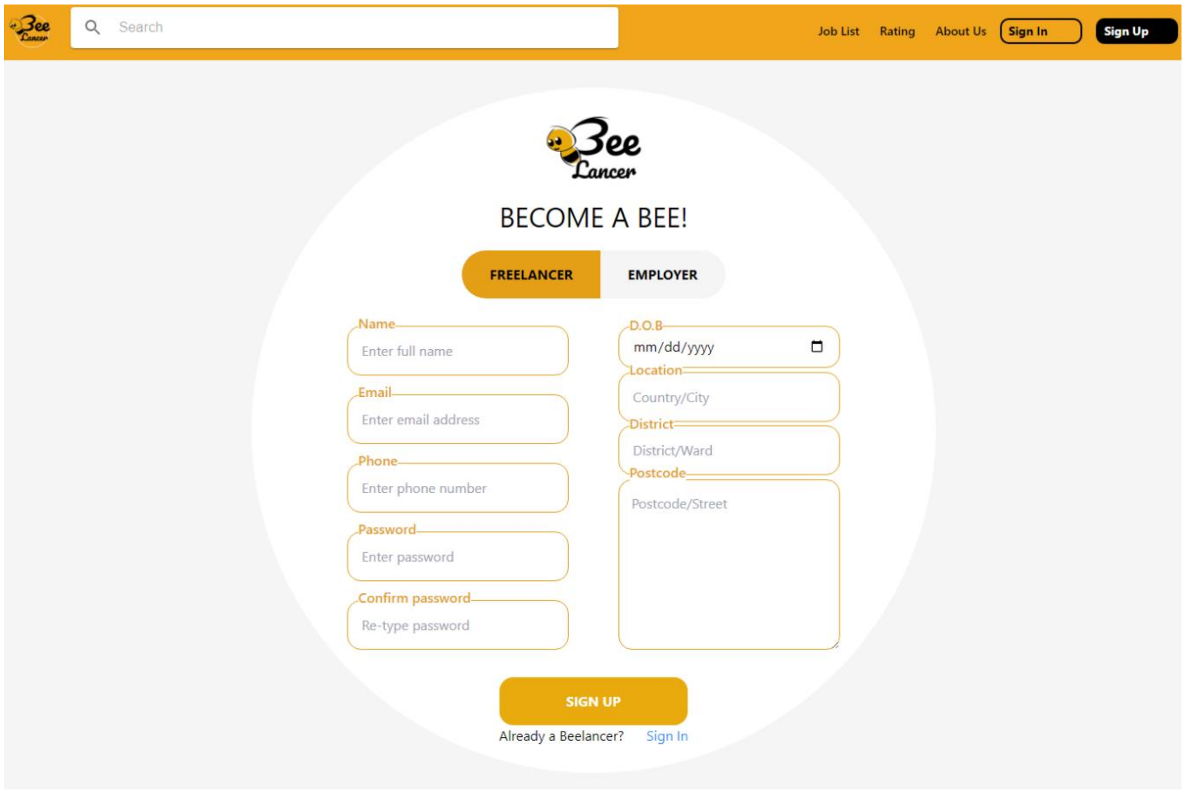
☐ Remember me

Forgot your password?

SIGN IN

New to our Hive? Sign Up

The login page is encircled to resemble the circled outline of the logo. There are two fields: email and password. The user can choose to remember the fields' details the next time they log in by clicking on "Remember me"



BECOME A BEE!

FREELANCER | EMPLOYER

Name: Enter full name

Email: Enter email address

Phone: Enter phone number

Password: Enter password

Confirm password: Re-type password

D.O.B: mm/dd/yyyy

Location: Country/City

District: District/Ward

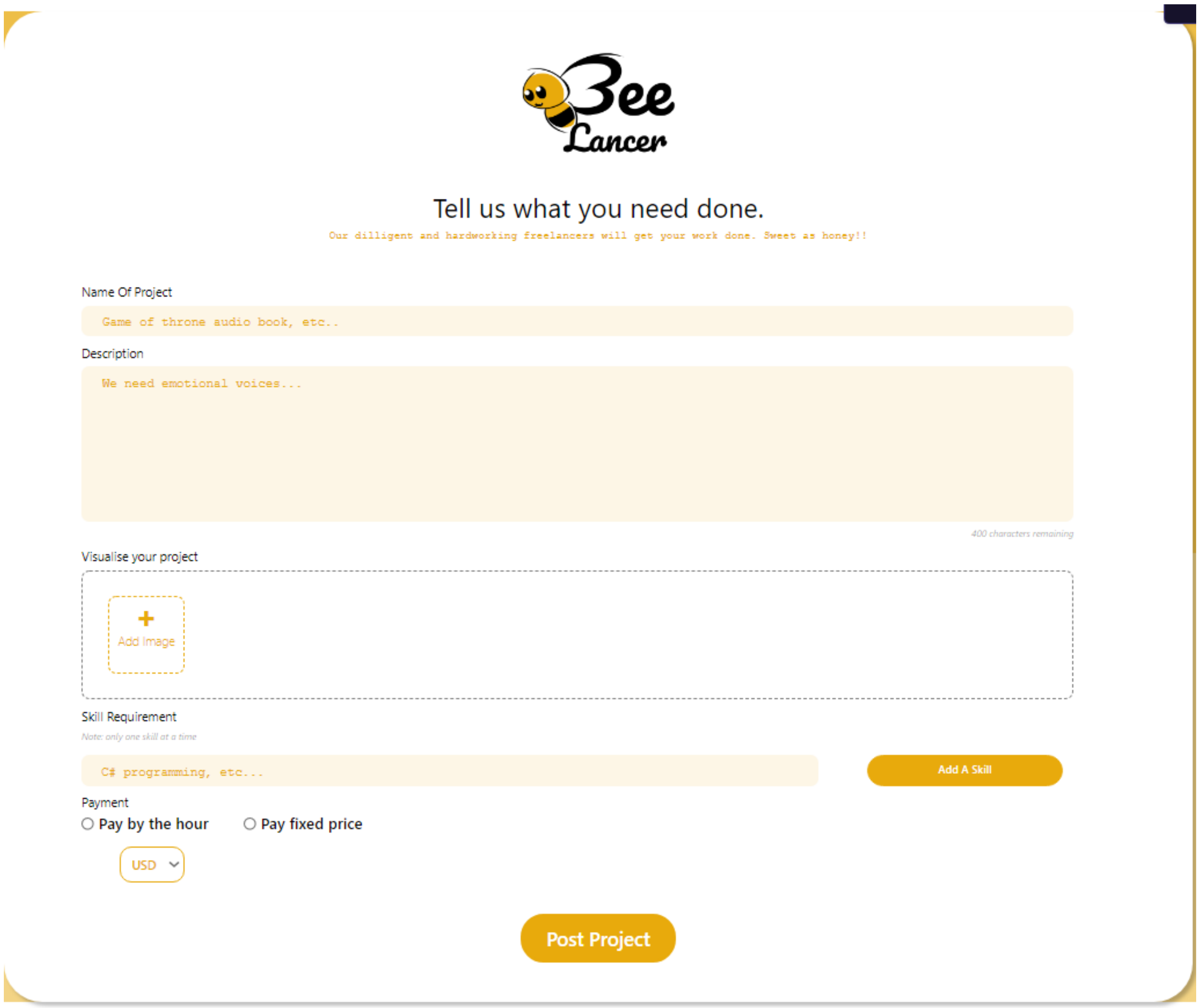
Postcode: Postcode/Street

SIGN UP

Already a Beelancer? Sign In

BeeLancer user can register either as a freelancer or an employer. There are several fields to be filled out. Once they are, clicking the sign-up button at the bottom will post the new user's details the server-side and the database.

## Contract posting



Tell us what you need done.

Our dilligent and hardworking freelancers will get your work done. Sweet as honey!!

Name Of Project

Game of throne audio book, etc..

Description

We need emotional voices...

400 characters remaining

Visualise your project

Skill Requirement

Note: only one skill at a time

C# programming, etc...

Payment

○ Pay by the hour ○ Pay fixed price

USD

Employers looking for freelancers to fill out their contract can post their project details on the web via this interface. The employers need to fill out the name of the project, its detailed description, and some visuals to facilitate comprehension and attract freelancers.

### 4. Limitations and Bugs

Due to several external circumstances, in addition to the time constraints and some individual performances within the development team, the application is **severely reduced** in scopes. The original proposal fleshed out a BeeLancer where there is payment methods, precise searching of posts of projects, and different portal interfaces for freelancers and employees. However, all of that were scratched out.

Moreover, the current final product bears many flaws, including lacking a complete search UI for the users.

Finally, due to time constraints, there were no quality Assurance via Unit Testing, or any form of Alpha or beta Testing as outlined in the proposal.

## IV. CONTRIBUTION

| **Name and student ID** | **Role(s)** | **Responsibilities** |
|---|---|---|
| **Le Trung Hau (s3741297)** | Project Manager, Backend Developer, Task Coordinator | Oversee Development Cycle, SCRUM master, entire backend schema, CRUD APIs, frontend-backend handshakes, Design Figma pages. |
| **Vu Le Hoang Nguyen (s3881101)** | Backend Developer, Report author | Design Figma pages of about us, responsible for the entire report, search API for jobs |
| **Giang Nhat Khanh (s3878182)** | Frontend developer, Meeting Minute Secretary | Monitor meeting minutes. Develop Sign In, Sign Up, Forget Password, Client Detail Page. Figma designer. |
| **Pham Nguyen Vu (s3701522)** | Frontend developer | Develop Home, About Us, Job Post Detail page. Figma designer. |
| **Nguyen Phan Nam (s3873792)** | Frontend developer | Develop Header component, Job List, Client Detail Page, Rating page. Figma designer. |
| **Tran Nguyen Anh Khoa (s3863956)** | Frontend developer | Develop Job Post, Contact Us, Freelancer Profile, Sign In, Sign up, Profile Setting pages and Footer component. Figma designer. Develop image upload for backend. |