**Low-Level Architecture and Data Models**

**<05>:<INSTASHOP>**

**<team member names & ids>**

| **Student ID** | **Name** |
| --- | --- |
| **23100233** | **Muhammad Umair Mohsin** |
| **23100268** | **Farva Talib** |
| **23100091** | **Muhammad Bilal Shahid** |
| **23100254** | **Salman Masood** |
| **23100016** | **Muhammad Affan Ashraf** |

| **Content** | **Totals** | **Obtained** |
| --- | --- | --- |
| Architecture diagram | 30 | 10 |
| Architecture justification | 20 | 5 |
| E/R diagram | 30 | 5 |
| E/R diagram description | 20 | 5 |
| Late submission |  |  |
| **Total** | **100** | **25** |
| **Individual Evaluation** |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**[Do proper formatting of the document.]**

**(Completed)**

**Table of Contents**

[1.](#_heading=h.gjdgxs) Introduction 4

[2.](#_heading=h.30j0zll) System Architecture 5

[2.1 Architecture Diagram—](#_heading=h.1fob9te)As it is in the prototype code 7

[2.2 Architecture Diagram—](#_heading=h.3znysh7)As it should-be 8

[3.](#_heading=h.2et92p0) Data Models 10

[4.](#_heading=h.tyjcwt) Tools and Technologies 13

[5.](#_heading=h.3dy6vkm) Who Did What? 16

[6.](#_heading=h.1t3h5sf) Review checklist 16

# Introduction

<Give an overview of the project here. The overview must highlight the overall objectives of the project and its potential users and customers.>

InstaShop will be a web-based portal which aims to revolutionize influencer marketing. For those who may be unaware, influencer marketing is a recent development in the digital world which involves a brand collaborating with an online influencer to market one of its products or services. These influencers are usually found on social media platforms (Instagram and Snapchat) and have a decent number of followers to whom they market a particular brand’s products or services.

There are two main parties involved in this process, the client and the influencer. For better understanding of the motivation behind this project, we must first walk through the process that is generally followed in influencer marketing from the perspective of both the client and the influencer.

For the client, the first step is perhaps the most tedious which is related to finding the right influencer for your brand. This is usually achieved by manually searching social media platforms or relying on word of mouth from friends/family regarding a particular influencer. The second step is contacting the shortlisted profiles. This is either done through direct messages or emails, both of which are again tedious tasks with no guarantees of a timely response. The final step (assuming the contract has been fulfilled) is the issue of payment. This is perhaps the most troublesome aspect because there is no guarantee that the influencer will produce content that is upto the mark and not run off with any advance payments made.

For the influencer (unless they have a huge following) it is usually hard to find clients for collaborations or for sponsored content. Since these influencers are very active on social media, their inbox and comment sections are almost always flooded which means they tend to miss out on potential business opportunities just because they weren’t able to see the direct message. Some profiles do have designated emails for business inquiries but their response times are in most cases not ideal because they just don’t check their email that often. Finally, the issue of payment also exists. There is no guarantee that the client will pay the influencer the full amount in a timely manner even if the work has been done upto the client’s standards and deadline.

As visible, both of these parties are in dire need of a platform that could automate most of these tasks for them as well as provide payment guarantees. This is where InstaShop comes in. It aims to streamline all the steps involved in this process by providing an easy to use web application thus saving time and effort for both parties involved. For the client, it makes it easier to search relevant influencers courtesy of our database and filtering method, connect with shortlisted influencers (via email or live chat) and have your payment secured (via escrow).

For the influencer, you essentially get access to a marketplace where you can find potential clients, not miss out on potential business opportunities just because your inbox was too cluttered and have a guarantee that the client will pay you for your work.

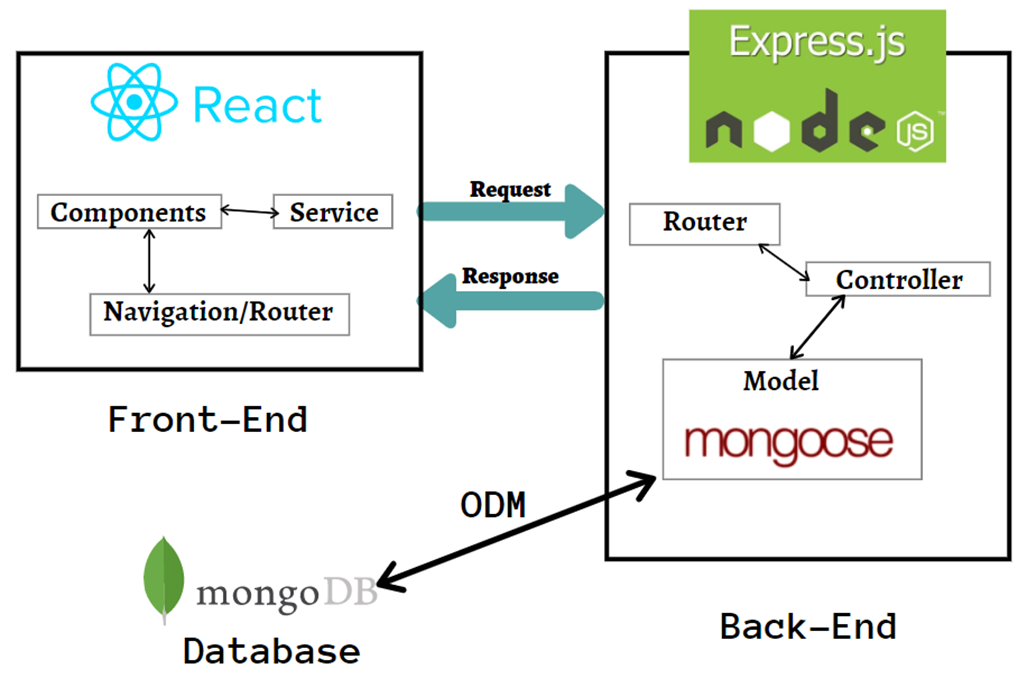
# System Architecture

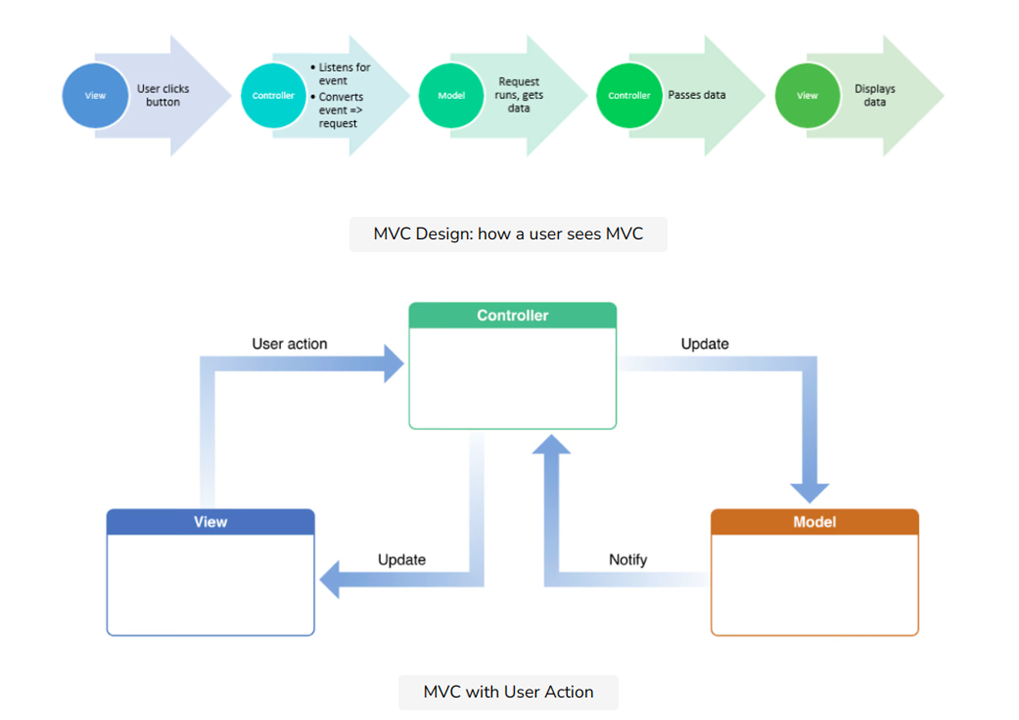
## Architecture Diagram—As it is in the prototype code

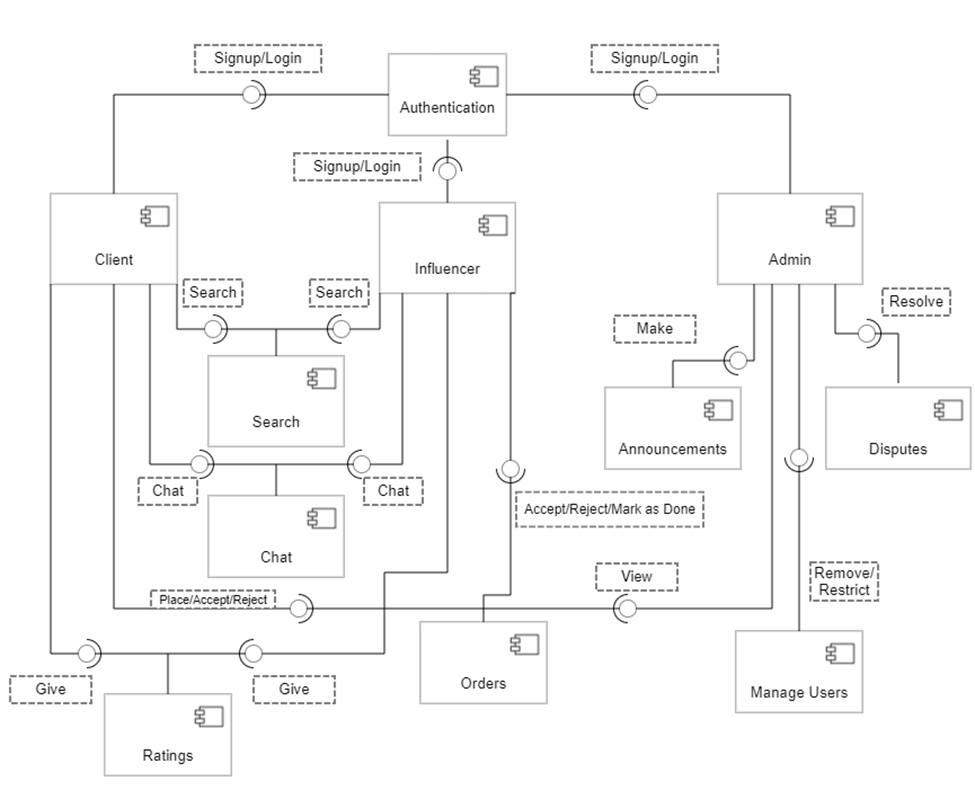
<Draw a diagram of the system architecture. The diagram must reflect the architecture of the code that you have written for the prototype. For instance, if you are following layered architecture, your diagram must distinguish all layers along with classes/modules in each layer. You must mention exact names of classes/modules inside the layers as they are in your prototype code. If there are too many classes, your diagram must show classes/modules involved in at least two use cases. >

[The components that you have identified should also be reflected in the layers, i.e., which layer/subsystem will contain which components. Without this, your system architecture diagram remains general—it should be specific to your system. I gave the same comment earlier as well. Also add description of modules.]

(Sir we had updated it as per your response from last deliverable and our model is mainly the same).







Same as Prototype Model

## Architecture Diagram—As it should-be

<Draw an updated diagram of the above system architecture. The diagram must reflect the architecture of the code as it should be. For instance, you may have learnt during prototyping that you should structure your code in a different way to make it, *maintainable*, for instance. This diagram should reflect this updated understanding. When you rethink your system architecture, keep the following important design goals in mind:

* Maintainability
* Reusability
* Extensibility
* Separation of concerns

In a couple of paragraphs, discuss how your updated architecture will help in achieving design goals listed above.

If you are following layered architecture, your diagram must distinguish all layers along with classes/modules in each layer. You must mention exact names of classes/modules inside the layers as they are in your prototype code. Similar statements hold if you are following another architectural pattern. If there are too many classes, your diagram must show classes/modules involved in at least two use cases.

Add a brief description of the classes/modules in the architecture diagram.

[Most of the modules that you have listed here are related to user interface only. Modules for other subsystems must be shown and described as well.]

[Discuss architecture justification separately.]

Since we are following MVC architecture each and every class/module that we have developed is completely independent from each other and at the same time they are all well connected in their entirety with the help of routes. The usage of rest api has helped us develop a modular code that can be reused in the future if required.

Some of the modules that we developed are as follows:

esignup,osignup: These module helps collecting data from the user and sending it to the backend(server) where the module dealing with its requests stores the incoming data to the database. We have further divided the signup based on the role of the user (i.e endorser and organization respectively).

login: this collects the user data and sends it to the module in the backend where it verifies the user's identity and grants access if verified.

ChangePass: this module shows the user an option to change their password. First the user needs to enter their current password and then enter a new password. The data entered by the user is sent to the module in the backend where the current password is verified and then new password is set.

Similarly, we have completely separated the modules that deal with the admin’s use cases. For example the “announce” and “EditAdmin” modules are exclusively used by the admin to send announcements and edit their profile if required respectively.

Moreover, from endorsee’s view, they are also given a separate window to edit their profiles as well from the “EditEndorsee” module

And to change organization details, an “EditOrganization” module is created.

Elaborating the modules developed in the server/backend, they are also developed to ensure that the code is as modular as possible. As mentioned before the use of rest apis helped us to make independent functions for every use case. For example adminProfile takes the request from the backend, checks whether the role of the user is admin and checks the admin details in the database and then responds back by sending the admin’s role,name, username, and password. Similarly under admin’s use cases, the createAnnouncement receives the json object coming from the frontend and saves that data in the database and sends an acknowledgement to the admin that the announcement has been successfully created. In the same manner, the getAnnouncement fetches the announcement from the database and responds back to the admin’s request.

Similarly, separate modules have been developed for clients and influencer’s end as well. clientProfile and InfluencerProfile fetches the user’s data after checking their role (based on their unique email ID) and responds back to the front end with their details that are stored in the database.

There are some common modules that are applicable to all the users. For them we have created a single module and within that module it segregates them based on certain checks (like their role). For example, since all the users need to sign up to the application, a single signup module(Singup) has been created which checks the role of the user (which is mentioned by the user in the frontend) and based on their role and the remaining data provided by the user stores them in the database. Furthermore, changePasswordGeneral finds the user from their email and updates their password and responds back to the frontend.

Besides the rest api, we have also designed the database in a modular manner. createAdmin, createAnnouncment, createClient and createInfluencer modules each have functions that set the defined attributes respectively so that these functions are triggered every time a user signs up or creates an announcement. Similarly, Admin, Announcement, Client, Influencer functions are designed to developed the schema of the database and all the attributes are defined in these functions.

In brief, the tasks are divided into modules based on the user’s role which in turn are further subdivided to ensure that the code is as modular as possible. This would help us to make any changes easier if required in the future and also would be easier to develop applications on different platforms(presently we are developing web app, we can also expand them to mobile apps) because of the use of routes and api calls.

So we have decided that our model of architecture diagram will remain the same after the prototype phase so we don’t need to update our architecture because we followed the same coding standard which is modular

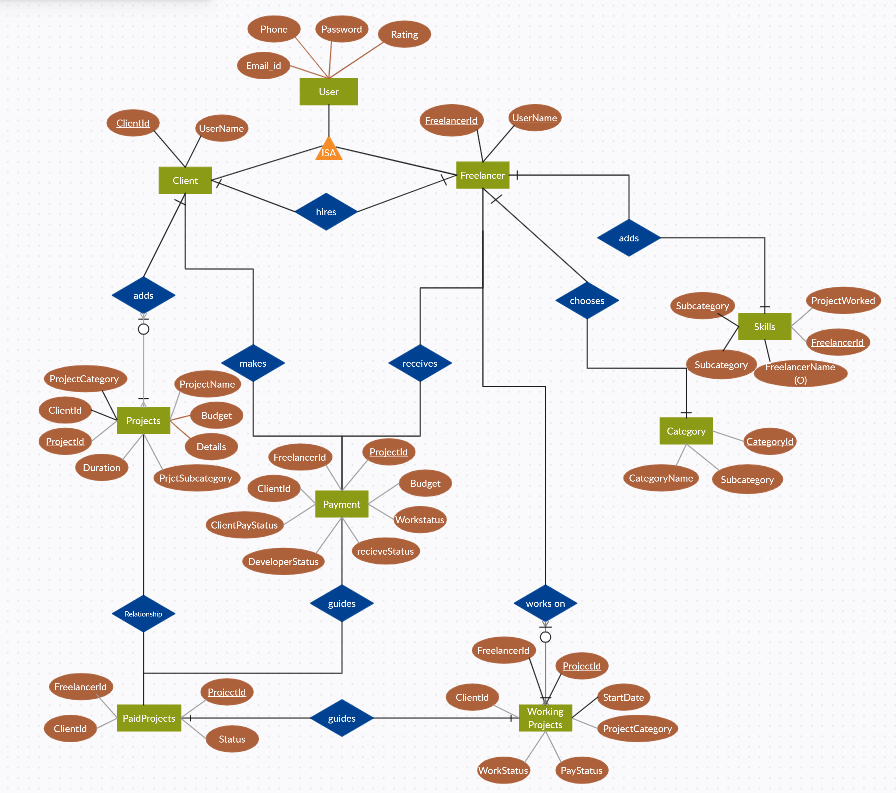
# Data Models

<Create data model of your system. For example, E/R diagram. Give brief description of entities and data fields.>

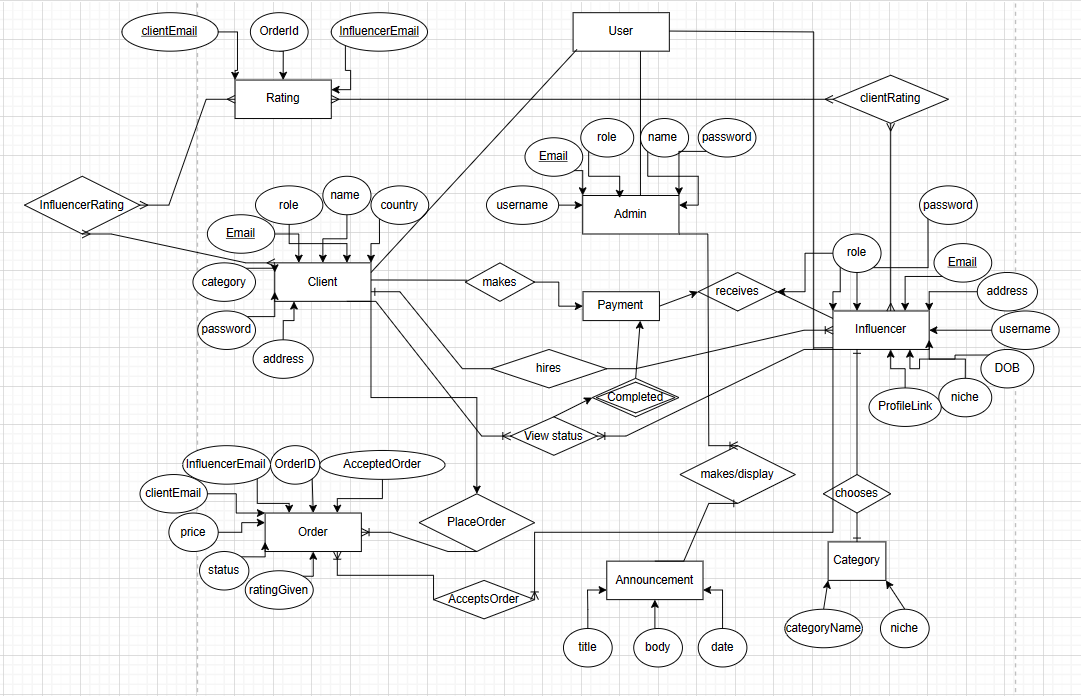
[Format this section properly, remove unnecessary details. Diagram should be given first and then details of entities. Moreover, draw data model properly. Data model is not merely a copy of the class model.]

ER diagram for our Model :

For an overall view of the model is shown below : Freelancer is same as Influencer in our platform



Along with our Project cases and entities as well :



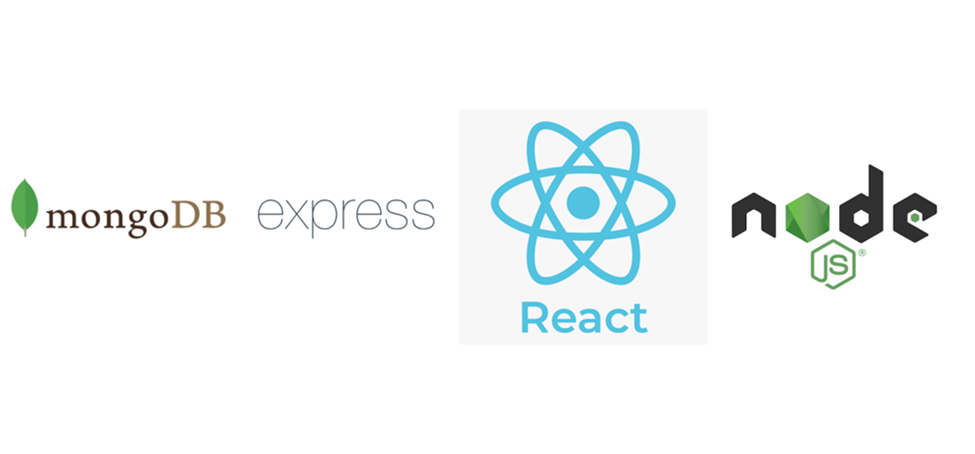
To start with description of ER relationships:

1. Firstly A User has a choice to be an Admin , client or an influencer when he signs up for an account and how he wants to register himself so the relation is single 1-1 as each entity is unique so a user can become one of them only.
2. Client Rating system (client-influencer Rating) is Many -to-Many because many several clients registered can give rating to any influencer for the orders/project they completed so this is the relationship.
3. Similarly Influencer Rating system (influencer- clientRating) is also Many-to-Many as well because many of the influencers who complete the order placed by clients are possible so multiple ratings of clients can be given by influencers.
4. Client and Order relationship (client-PlaceOrder) is Many -to- Many because a registered clients can place more than 1 orders against an influencer for different types of categories/niches and set the price , status , and email respectively against that particular order.
5. Admin to Announcement (One -to- Many) as an admin has the option of adding an announcement which is displayed to the users side. Which include a title and body of text shown to them.
6. Influencer- Category (One-to-One) is because an influencer has a choice of choosing one type of niche/category only.
7. Client - View Status ( One- to- Many) as clients have placed an order (multiple orders possible) it can view in the progress tab if the order is pending or completed yet or not by the influencer.
8. Influencer - View Status (One -to-Many) as influencer has received an order (multiple orders possible) it can view in the progress tab if the order is pending (if not accepted yet) or completed yet by the influencer against the OrderID.
9. Influencer - AcceptOrder( One -to-Many) relationship is there as an influencer can accept as many orders as possible according to his needs or price preference as well.

# Tools and Technologies

<List down tools and technologies that you are using for development and deployment. Make sure that you mention name and version of the tools.>

We'll be using the MERN (MongoDB, ExpressJS, ReactJS, NodeJS) stack for this project's development. The MERN stack is among the most widely used technology stacks for creating web applications.



Prior to choosing this particular technological stack, several factors were taken into account.

The following is a list of some of the causes:

● Because MERN Stack only supports one language, the team is better able to work together and support one another.

● ReactJS renders and performs UI elements more quickly than other UI frameworks and libraries.

● The community has a lot of support for Javascript because it is one of the most widely used languages.

● Supports the Model View Controller protocol, which enhances the development phase's efficiency.

● The MERN stack has built-in testing libraries, which provide testing for the application.

● There are various libraries available, which makes it easier to implement many functions.

● Asynchronous programming is supported by JavaScript, which enhances performance.

MongoDB:

Using a horizontal scale-out architecture, MongoDB is an open-source document database. People use this database because it has a quick technique for retrieving data and can hold a lot of data.

ExpressJS:

ExpressJS is a JavaScript web application framework that is available as open source. ExpressJS makes it simple and rapid to construct APIs. It helps organize web applications on the server side into a more structured MVC design and is lightweight. We can effectively route our traffic using ExpressJS, and we can also build middleware to handle requests for traffic in the right way.

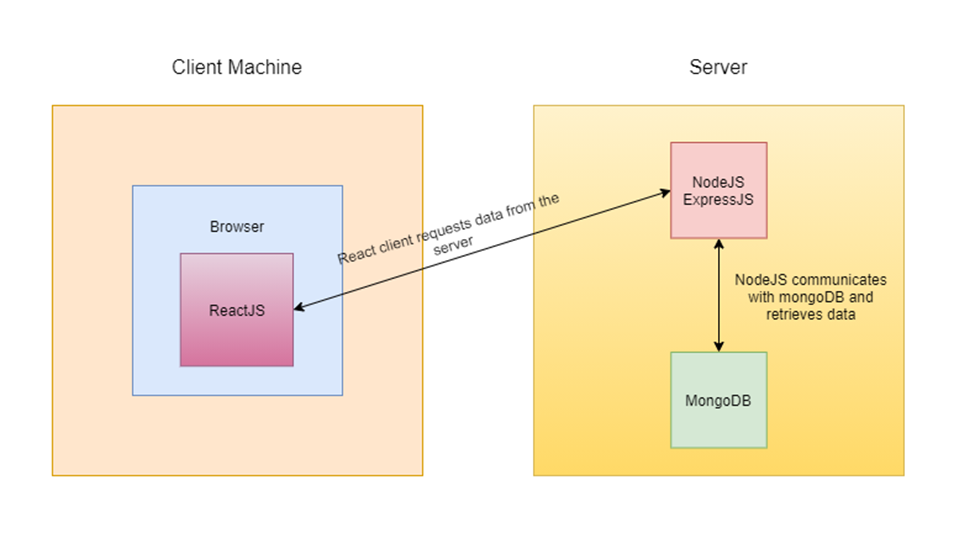
ReactJS:

For building interactive user interfaces, ReactJS is a free and open source JavaScript front end library. Facebook currently looks after it. ReactJS makes it simpler to create the straightforward views that our online application needs. Furthermore, since each component is capable of maintaining its own state, we may add new code without having it impact existing code.

NodeJS:

Performance, efficacy, and the speed at which your online application loads are all factors that a developer must consider when choosing coding languages or frameworks. As we all know, JavaScript is the most popular programming language for creating client-side web applications. Since NodeJS was introduced, JavaScript can now also be used to create server-side apps. Our application's effectiveness and efficiency are two advantages. An open source run-time javascript environment called NodeJS is based on Chrome's V8 engine. It is advantageous because it is incredibly rapid and effective, event-driven, and provides non-blocking I/O. It can complete multiple tasks at once without hindering or interfering with other operations. As a result, the performance of the websites is greatly enhanced.

This figure is only for illustration purposes:



Deployment:



We used Heroku for the deployment of our application. Heroku enables programmers to easily and quickly deploy an application to a web server. Additionally, it offers a large number of plugins that you can include in your application. You can always move more quickly with a PaaS solution than you can with a VPS solution where you have to configure everything from scratch.

# Who Did What?

| **Name of the Team Member** | **Tasks done** |
| --- | --- |
| Muhammad Umair Mohsin | Intro and Architecture , ER diagram |
| Farva Talib | Tools and technologies. |
| Muhammad Bilal Shahid | System Architecture |
| Salman | 2.2 |

# Review checklist

Before submission of this deliverable, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

| **Section** **Title** | **Reviewer Name(s)** |
| --- | --- |
| Muhammad Umair Mohsin | Tools and technologies. |
| Farva Talib | 2.2 |
| Affan Ashraf | ER and description |
| Salman | System Architecture |