UNIVERSITE ABDELMALEK ESSAADI

FACULTE DES SCIENCES ET TECHNIQUES DE TANGER

End Of Module Project

Web Development

Subject:

Building a Fiverr Clone: MERN Stack and GraphQL Implementation

Prepared by:

Mohcine BOUDENJAL

Hind EL OUAHABI

Oriented by:

Pr. ELAACHAK Lotfi

Table of Contents

I. Introduction:

In this project, we will build a Fiverr clone using the MERN stack (MongoDB, Express.js, React.js, Node.js) and GraphQL.

Fiverr is a popular online marketplace connecting freelancers and clients. By creating this clone, we will gain hands-on experience with backend and frontend development using the MERN stack, while leveraging GraphQL for efficient data retrieval and manipulation.

What is a PWA?

Progressive web applications are the web applications with some practical improvements that allow these app works like a desktop app or mobile app.

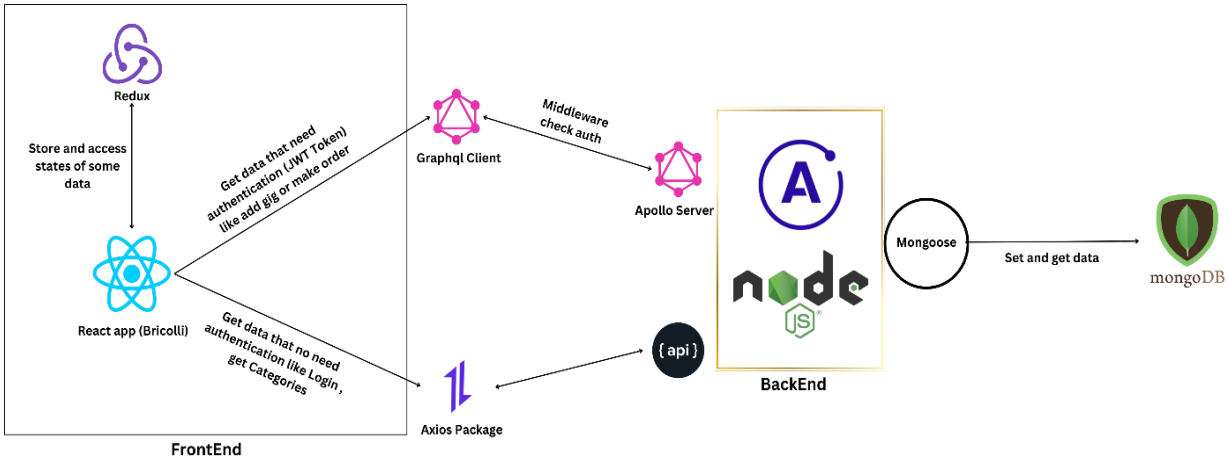
PWAs offer features such as offline functionality, an app-like interface, push notifications, and the ability to be installed on the user's device.

II. Mission:

Create a WPA (Web Progressive Application) clone of Fiverr Platform using React JS, NodeJs / Express, MongoDB and GraphQL /REST API.

III. Project Architecture:

This is our architecture of the project with technologies used.



On the frontend side, we have chosen to utilize the React.js framework along with the Redux package for managing the application state. React.js provides a robust and efficient way to build user interfaces, allowing us to create reusable components and efficiently handle updates.

To handle data retrieval from the server, we have decided to divide the services into two parts.

For RESTful APIs that do not require authentication or tokens, such as fetching gigs and categories, we have opted to use Axios. Axios is a popular JavaScript library that simplifies making HTTP requests from the browser. It provides an intuitive and convenient API for sending asynchronous requests and handling responses. By incorporating Axios into our frontend stack, we can easily retrieve data from the server and seamlessly integrate it into our React.js components. This allows us to efficiently manage and display the necessary information. without the complexity of authentication or token-based authentication for these particular services.

For the second part of our services that require tokens, such as creating, updating, and deleting data, we have chosen to implement GraphQL. GraphQL is a query language for APIs that provides a flexible and efficient way to retrieve and manipulate data.
To handle GraphQL requests and responses on the frontend, we have opted to use the Apollo Client. Apollo Client is a powerful JavaScript library that simplifies working with GraphQL in React applications. It provides an intuitive and declarative API for handling queries and mutations, as well as caching and managing the local application state.

For styling and materials, we opted to utilize the Tailwind CSS and Heroicons packages. These packages offer an extensive range of styling options, components, and utility classes, enabling us to easily customize the appearance of our web application. Tailwind CSS provides a comprehensive collection of pre-built styles and responsive layouts, making it effortless to create visually appealing designs. On the other hand, Heroicons offers a vast library of icons in various styles and sizes.

On the backend side, we have employed Node.js along with the Express framework to build our server infrastructure. Node.js enables us to run JavaScript on the server, providing a fast and efficient runtime environment. Express, being a minimal and flexible framework, allows us to handle HTTP requests, define routes, and implement middleware easily.

To simplify testing our GraphQL queries and mutations, we have incorporated Apollo Server. Apollo Server is a GraphQL server implementation that seamlessly integrates with Express. It provides a powerful set of tools and features for building, testing, and serving GraphQL APIs, making it convenient to test and validate our GraphQL operations.

For defining and working with schemas (models) in our MongoDB database, we have chosen the Mongoose package. Mongoose is an Object Data Modeling (ODM) library for MongoDB, providing a straightforward and intuitive way to define data schemas and interact with the database.

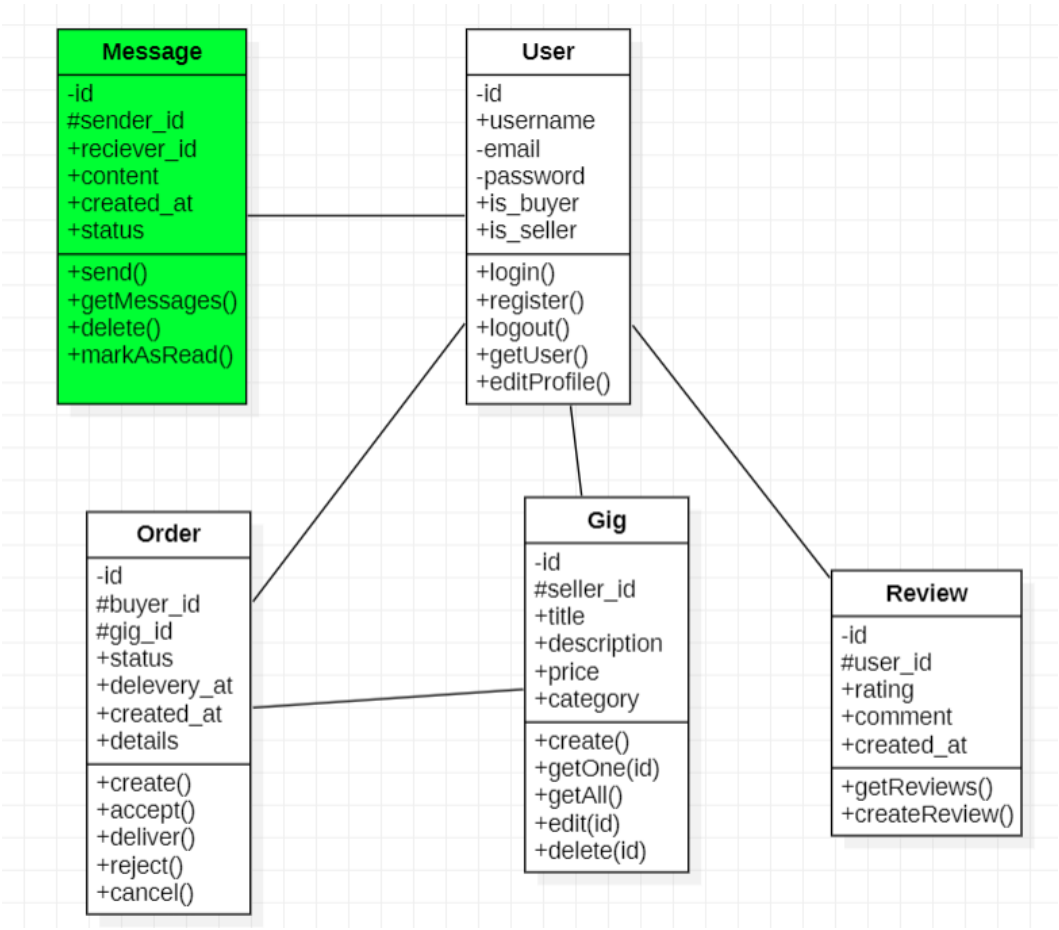
IV. Specifications:

1. The Gig class represents the services offered by sellers
2. Review is feedback left by a buyer on a completed order. When a buyer receives their completed order from a seller, they have the option to leave a review that reflects their experience working with the seller
3. Gigs are categorized into various categories and subcategories to make it easier for buyers to find the services they need (Graphics & Design, Digital Marketing, Writing & Translation, Programming & Tech …)
4. A User is directly a Buyer but he can be a seller as well.
5. A Seller is a User who creates and sells Gigs on the platform.
6. A Buyer can be a seller if he fills the form of Become a seller with his professional information like (full name and skills)
7. A Buyer is a User who purchases Gigs from Sellers on the platform.
8. Message has 3 statuses: sent(default), read and failed
9. An order can have different statuses depending on where it is in the order process:
    a. Active: Order passed, need seller to accept
    b. In progress: that mean the seller accept the order and start his work
    c. Delivered: This status means that the order has been successfully completed, and both the buyer and the seller are satisfied with the work. Payment is released to the seller, and the buyer can leave a review for the seller.
    d. Revision Requested: If the buyer is not satisfied with the work, they can request revisions from the seller.
    e. Completed: This status means that seller finished his work and he submit it, waiting for the confirmation of buyer that if approve it or request revision
    f. Cancelled: If the buyer or the seller needs to cancel the order for any reason
10. A messaging system allows buyers and sellers to communicate with each other directly.
11. If a seller submits his work, the buyer can agree or request a revision for the seller
12. If a gig is linked to an incomplete order, it is not possible to delete the gig.
13. If the gig delivered, the buyer passes to the phase of review that can give to the gig a number of stars and the leave a comment bellow

V. Conception:

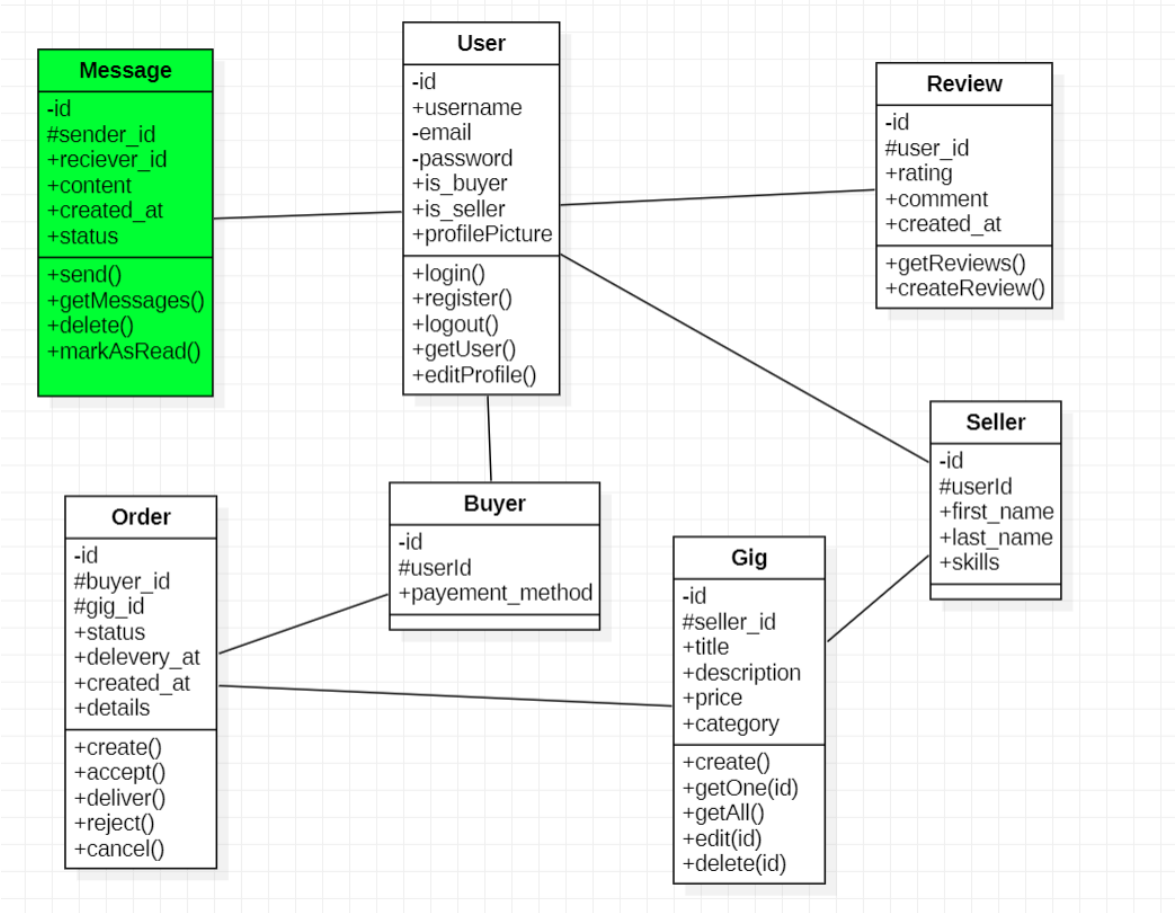
After we understand the system of Fiverr, we are able to make some UML diagrams, the first one is a class diagram with 3 various versions because every meeting we have something to add it.

The first one is after testing some features of Fiverr platform so we suggest the V1:



05/08/2023: update

After that we have some modifications that a user should be linked with his two profiles seller and buyer. So, we create the V2

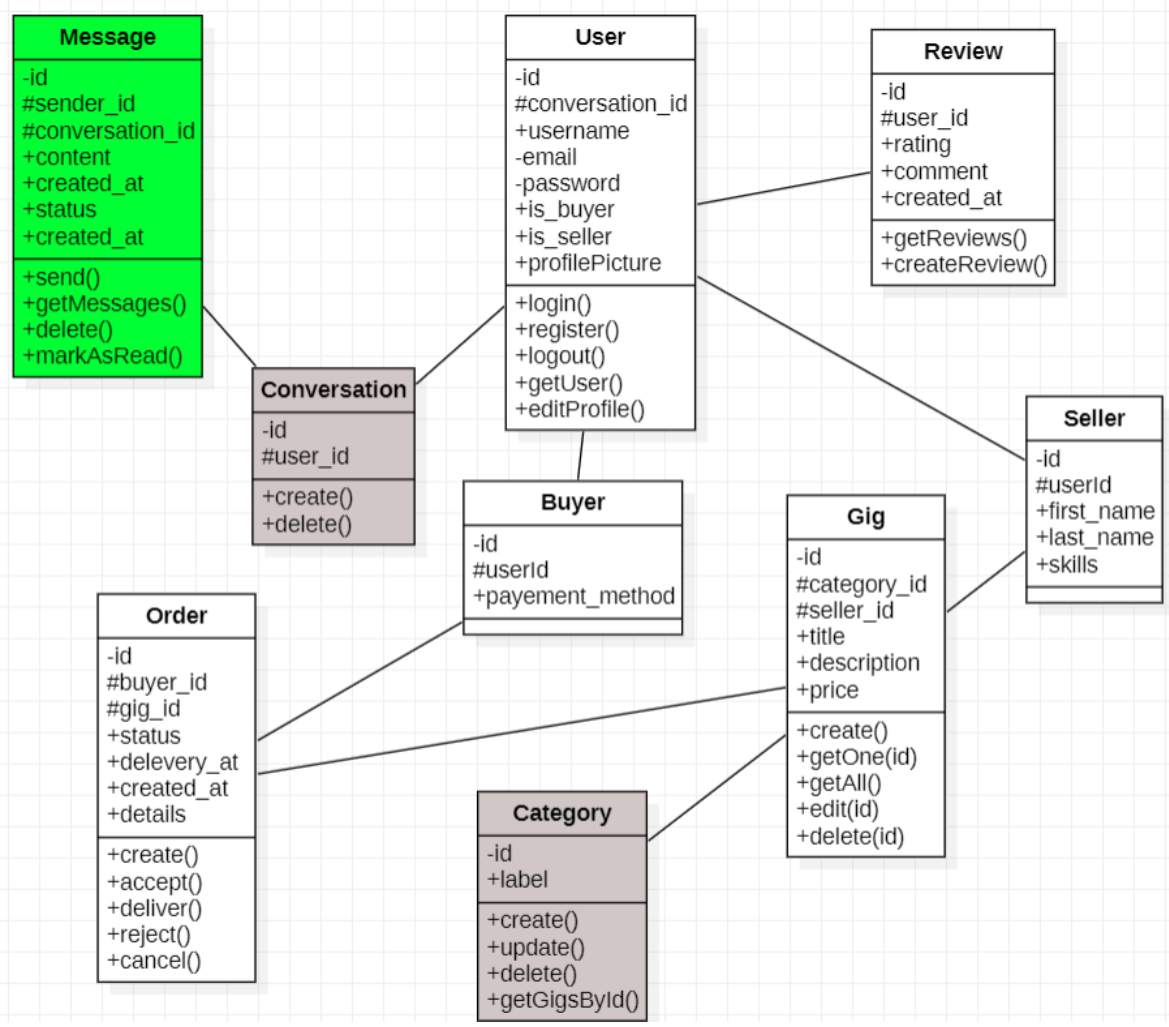


05/11/2023: update

Following our comprehensive comprehension, we decided to add two classes, one for categories that related to the gigs so we can group the gig by categories.

The second class is Conversation, class that can group even the parts of a conversation users with an array and the messages linked to this conversation

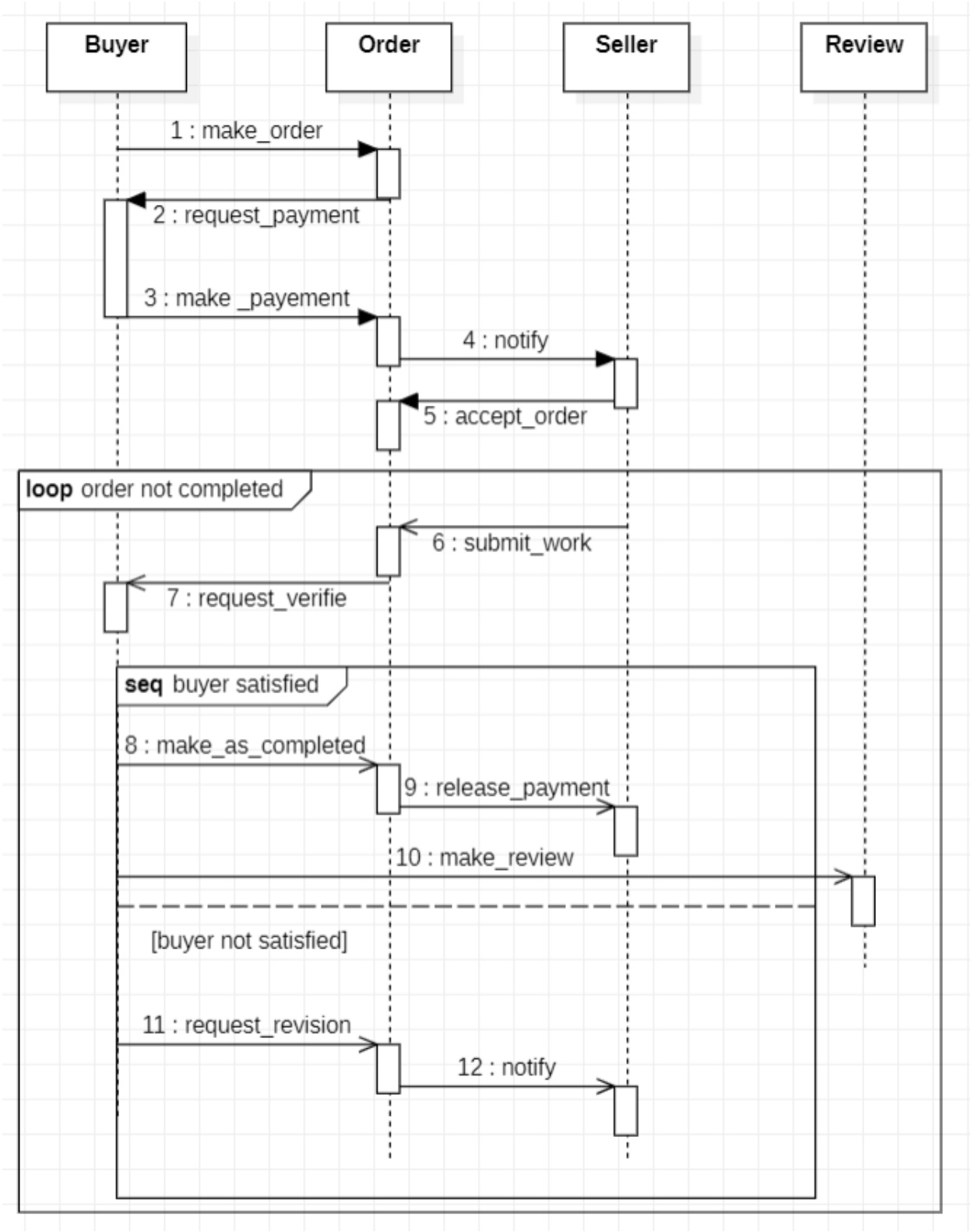
This is our final class diagram V3:



After finalizing the class diagram and understanding the system's classes, we can create a sequence diagram. This diagram illustrates the communication flow between classes, specifically showcasing the process of passing an order from a buyer to the seller for service delivery and make review.

It provides a visual representation of the interactions and message exchanges.

This sequence diagram helps us analyze and understand the system's functionality and communication patterns in a concise manner.

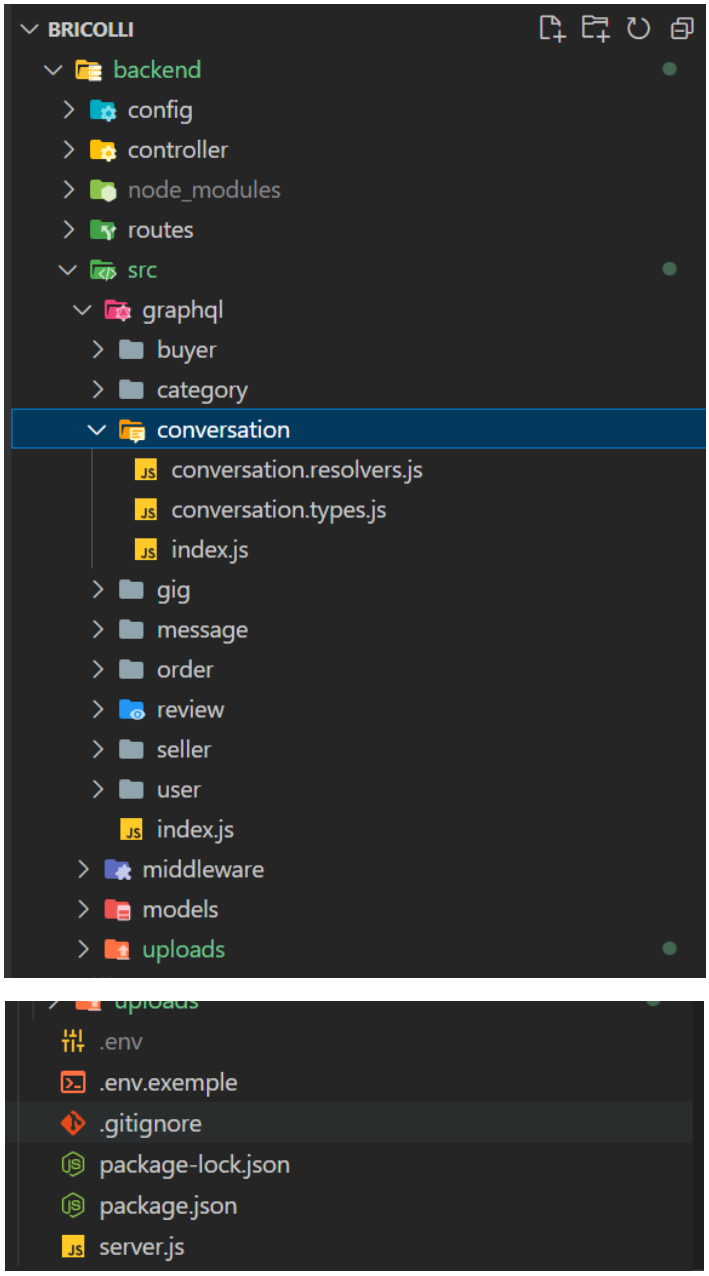

Buyer
Order
Seller
Review
1 : make_order
2 : request_payment
3 : make _payement
4 : notify
5 : accept_order
loop order not completed
6 : submit_work
7 : request_verifie
seq buyer satisfied
8 : make_as_completed
9 : release_payment
10 : make_review
[buyer not satisfied]
11 : request_revision
12 : notify

VI. Code Architecture:

In this part, we will interduce the architecture of our code. The best architecture provides a solid foundation for the development and long-term success of the project.

We start with the backend side:

Backend side:



Config: configuration settings like database connection It typically retrieves these settings from a .env file, which provides a secure way to store sensitive information.

Controller: responsible for handling functionalities that do not require authentication, such as login, registration, and data operations.

GraphQL: Contain the typedefs and resolvers of each model in our database, define query and mutation operations, and the logic for retrieving and manipulating data from the models.

Models: The models component defines the schemas of your models using Mongoose, an ODM library for MongoDB. These schemas specify the structure, validation rules, and relationships of your data models.

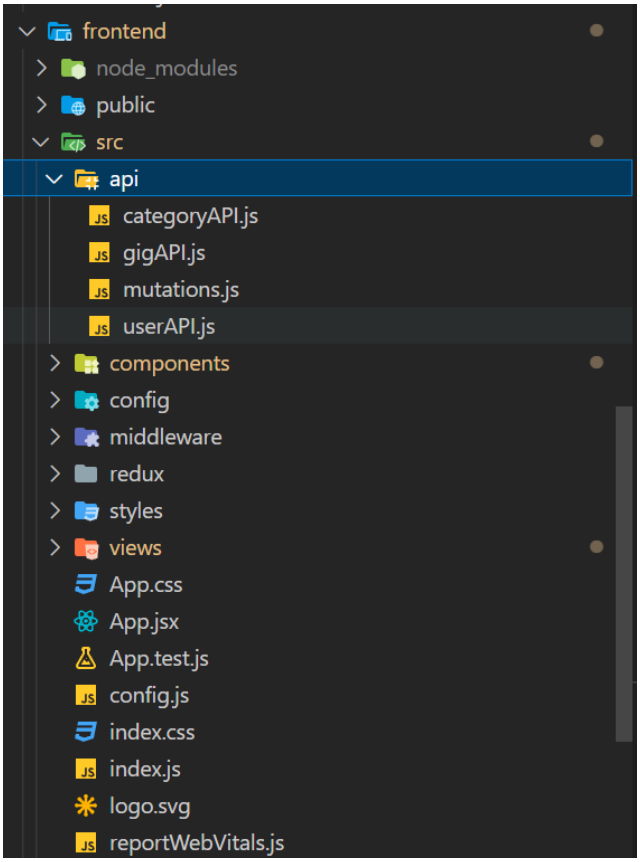
Middleware: includes functions that act as middleware for various purposes. It may contain middleware functions for token verification, authentication, authorization, and other custom operations. Additionally, it can include multer configurations for handling file uploads.

Uploads: for uploading and storing the pictures (profiles and banners)

Server.js: entry point of our backend, here we start the server and apollo server.

Package.json: contains metadata about the project, as well as configuration information for managing dependencies, scripts, and other project-related details.

Frontend side:



Api : contains the services that your frontend application can consume, such as APIs. It likely includes functions or modules for making HTTP requests to your backend server, handling responses, and interacting with data from the server. Additionally, it may include file mutations that define queries and mutations of GraphQL

Components: contains widgets or elements that make up your user interface. Examples include cards, headers, footers, buttons, and other UI elements that can be used across different views

Config: For creation and setup of Axios and Apollo Client, which are commonly used for making HTTP requests and managing GraphQL operations, respectively. It likely includes configuration settings such as base URLs, headers

Redux: Contain the reducers (Slices) and store of application.

Views: Contains the different pages or views that make up your application. Each view represents a specific section or screen of your application, and they use components from the Components directory to build the user interface.

VII. Packages

In our development cycle, we used various of packages provided by npm registry community.

Let's know some of these packages:

Backend:

1. **Express**: A fast and minimalist web framework for building web applications and APIs.
2. **Mongoose**: An Object Data Modeling (ODM) library for MongoDB that provides a higher-level abstraction for interacting with the database.
3. **Apollo Server**: A GraphQL server implementation that integrates with popular Node.js frameworks like Express to provide a GraphQL API.
4. **Bcrypt**: A library for hashing and salting passwords, commonly used for user authentication and security.
5. **JWT (jsonwebtoken):** A package for generating and verifying JSON Web Tokens (JWTs), which are often used for authentication and authorization.
6. **Multer**: A middleware for handling multipart/form-data, primarily used for file uploads in Node.js applications.
7. **Dotenv**: A module for loading environment variables from a .env file, allowing you to store sensitive information or configuration settings outside of your codebase.
8. **Sharp**: popular Node.js package used for image processing and manipulation. It provides a high-performance and efficient way to work with images (resize, rotate …)

Frontend:

1. **Framer Motion**: Framer Motion is a popular animation library for React applications. It provides a simple and declarative way to create smooth and interactive animations
2. **Tailwind CSS**: is a highly customizable and utility-first CSS framework. It provides a set of pre-built CSS classes that can be easily applied to HTML elements to style and design user interfaces.
3. **Heroicons:** is a set of open-source icon packs designed for use in web and mobile applications. It provides a wide range of SVG icons that can be easily integrated into your project.
4. **Apollo Client**: Apollo Client is a library for managing GraphQL data in client-side applications.
5. **React Router DOM:** React Router DOM is a routing library for React applications. It enables you to handle navigation and create dynamic routing within your application.

6. **React Redux:** React Redux is the official state management library for React applications. It is built on top of Redux and provides a way to connect your React components with the Redux store

VIII. Description of pages

Based of our conception and the Fiverr platform design, we put our planification of pages without any design provider like Figma. This is what we decided to do:

Component:

Navbar: Logo with research bar and list horizontal of categories with 3 cases:

- If not login display buttons of register and login
- If login and not seller display buttons of become seller and profile with submenu (messages, orders and logout)
- If login and seller just profile with submenu added gigs

Footer: Copyright

GigCard: This is the card from Fiverr so we will do something like that

Pages:

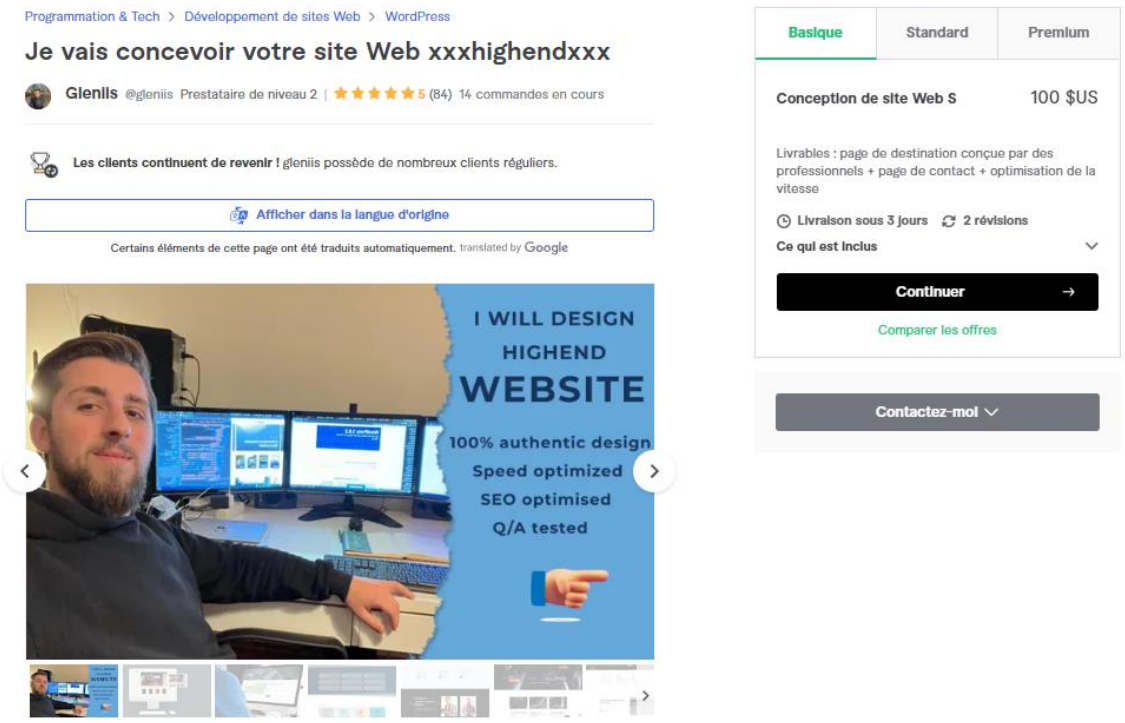
Login: page login standard (email+password)

Register**:** page register standard (username+email+password+confirm password+photo profil)

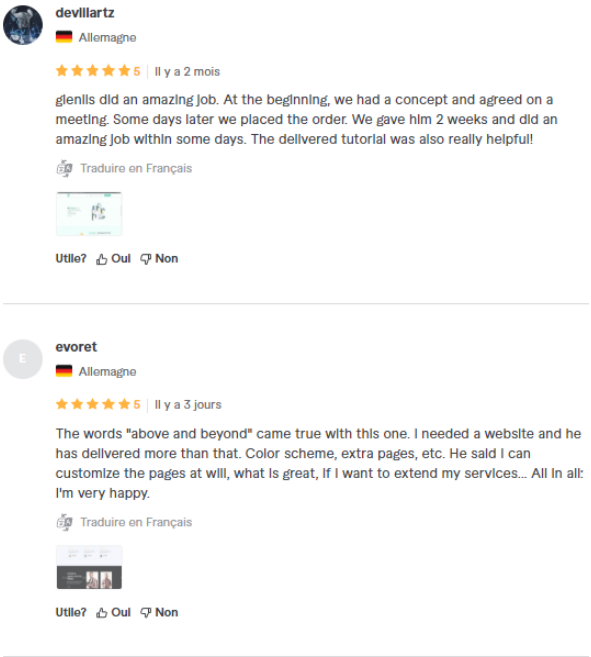
Home: list of gigs 4 in every line

Gigs details: We will divide this page to 3 parts like in Fiverr:

- Part for gig informations
- Part for make order



- Part for reviews in bottom

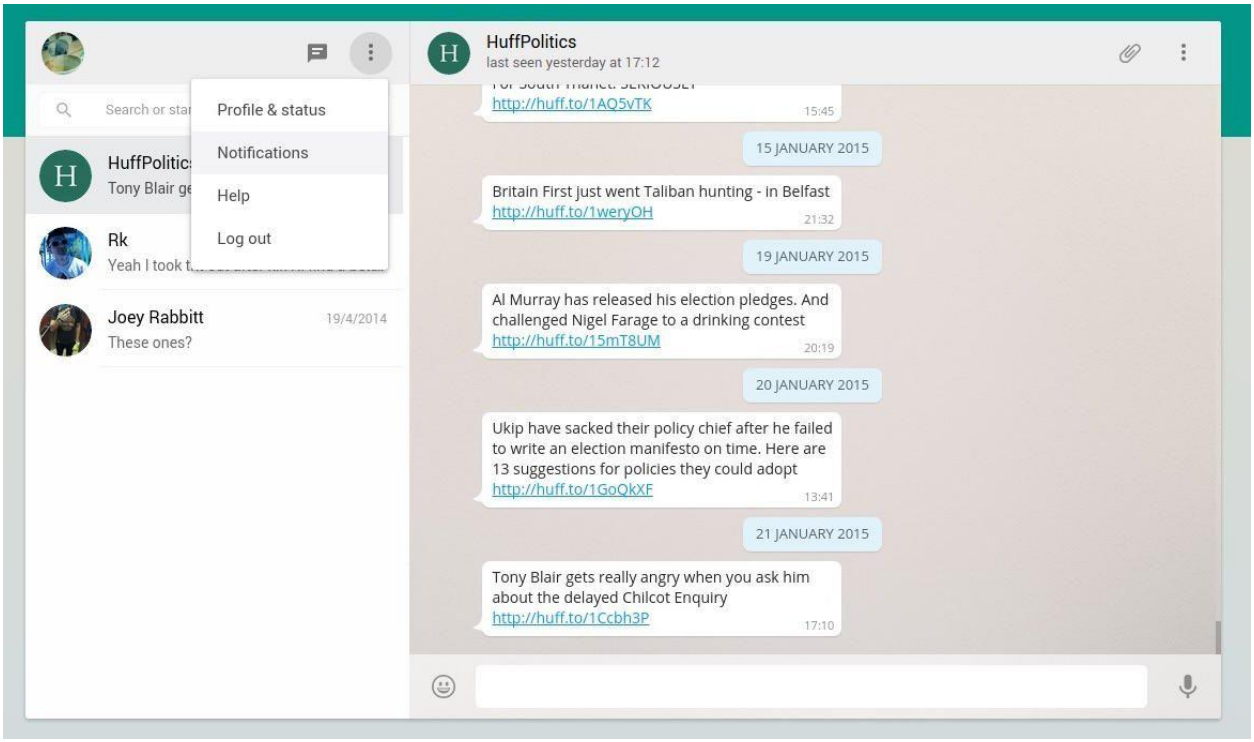


Buyer Orders: List of orders with status and actions like this:

- If status completed: two buttons required revision or approve it
- If status delivered: appear form to add review (stars hover and comment)

Gig Orders: view of orders for gig, two tables one for pending orders with information of order (delivered time, buyer, details …) and one for current orders that can submit order and make revision

Conversations: Something like WhatsApp web with conversations in left side and messages in right side



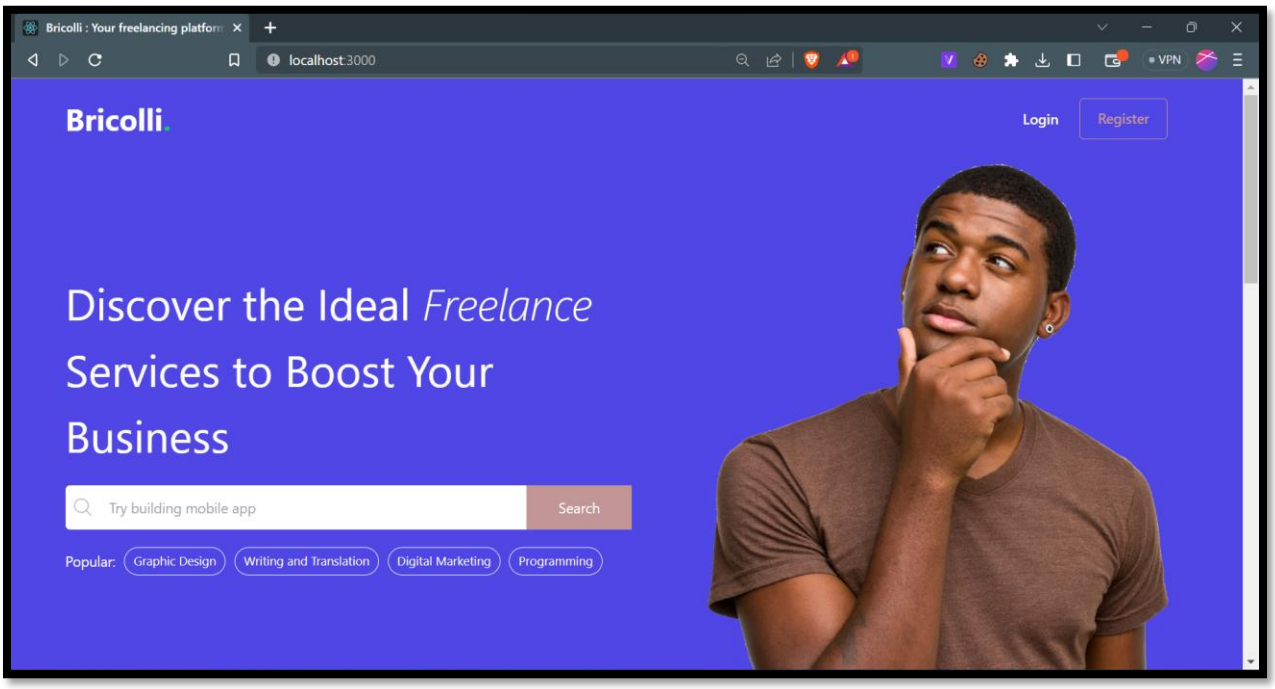
Become A seller form: form that compose of inputs of first name last name and skills

This is in general our planification of essential pages that compose our application, the next part we will show the result of all the previous parts.

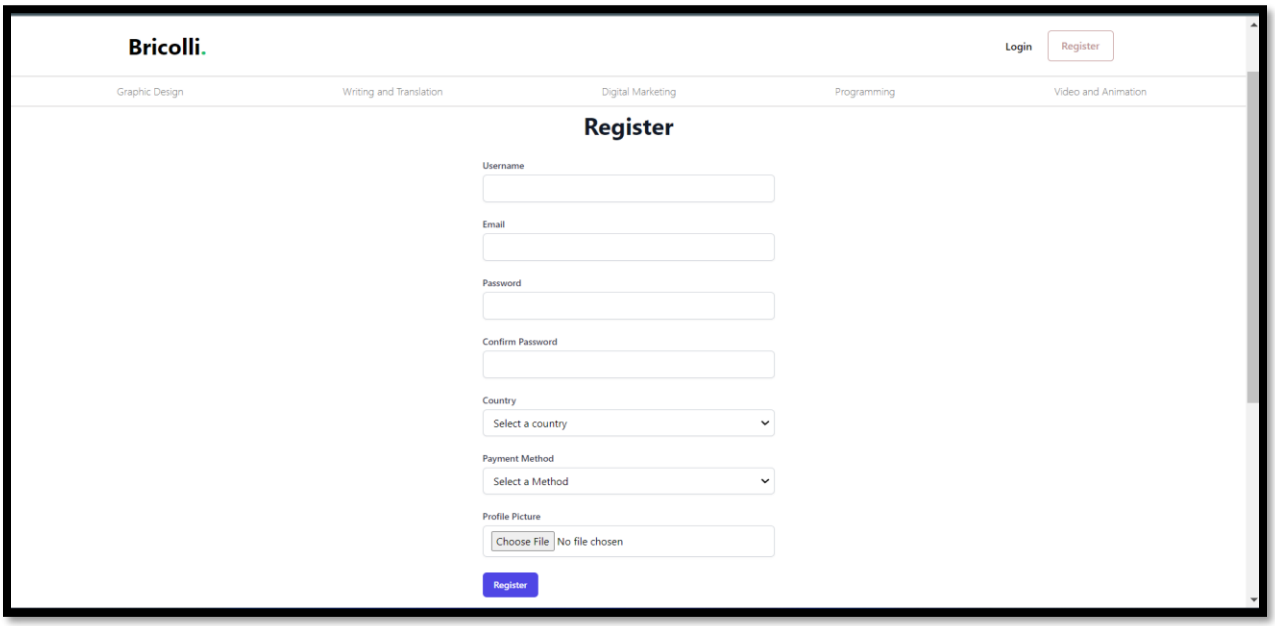
IX. Pages of Application

In this part, we will visualize the result of all the previous parts including conception technologies and packages. We based our design directly from Fiverr platform without using any design planification like Figma or something else.

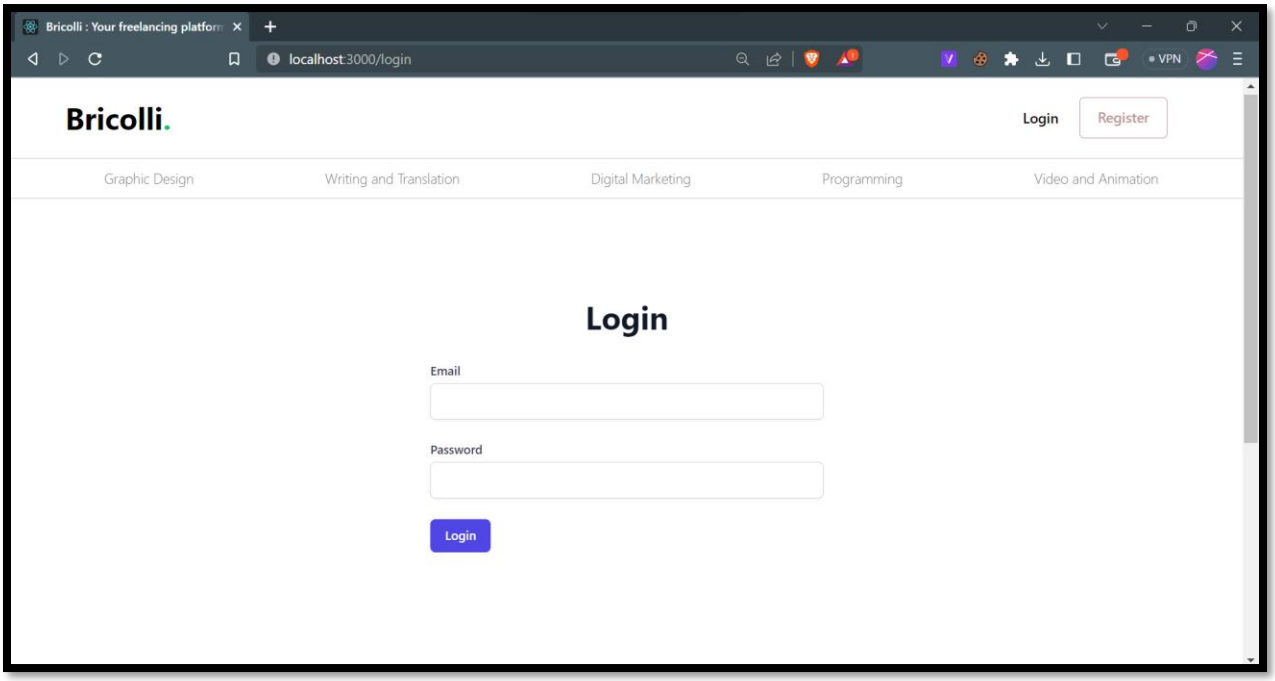
The first page home appears like this with buttons login and register and some popular categories of gigs



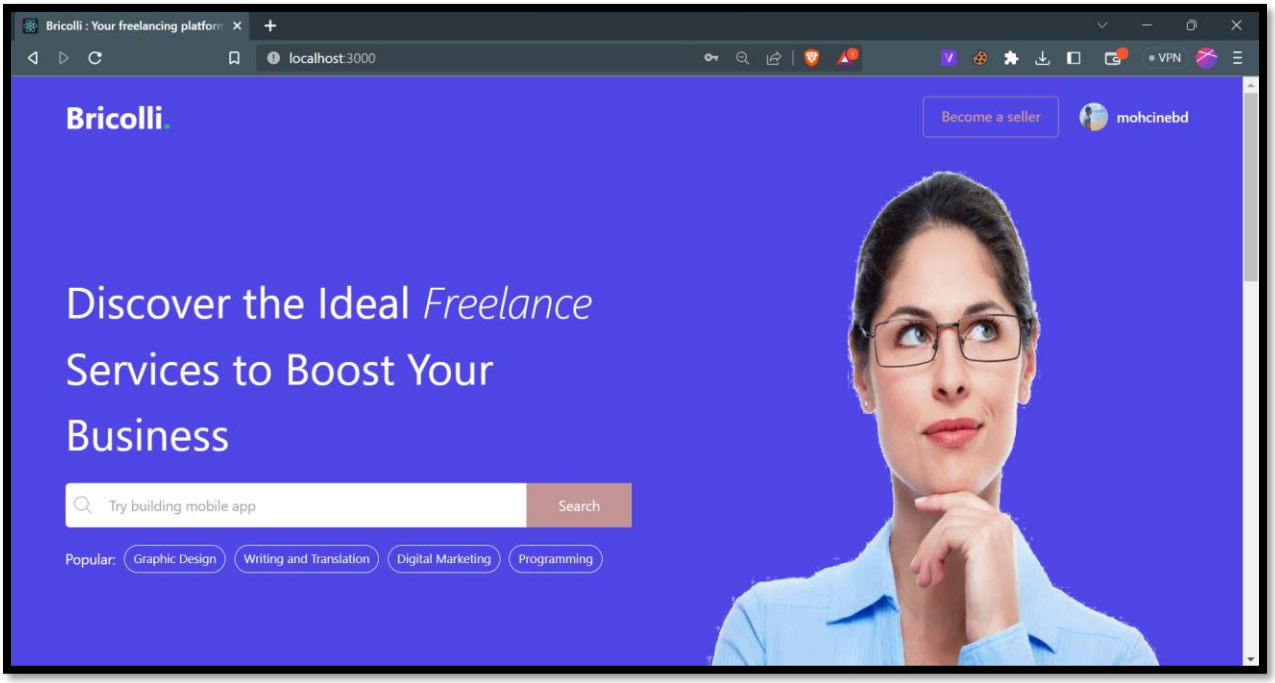
Registering page for sign up the first time, the navbar change his format when he is not in home page



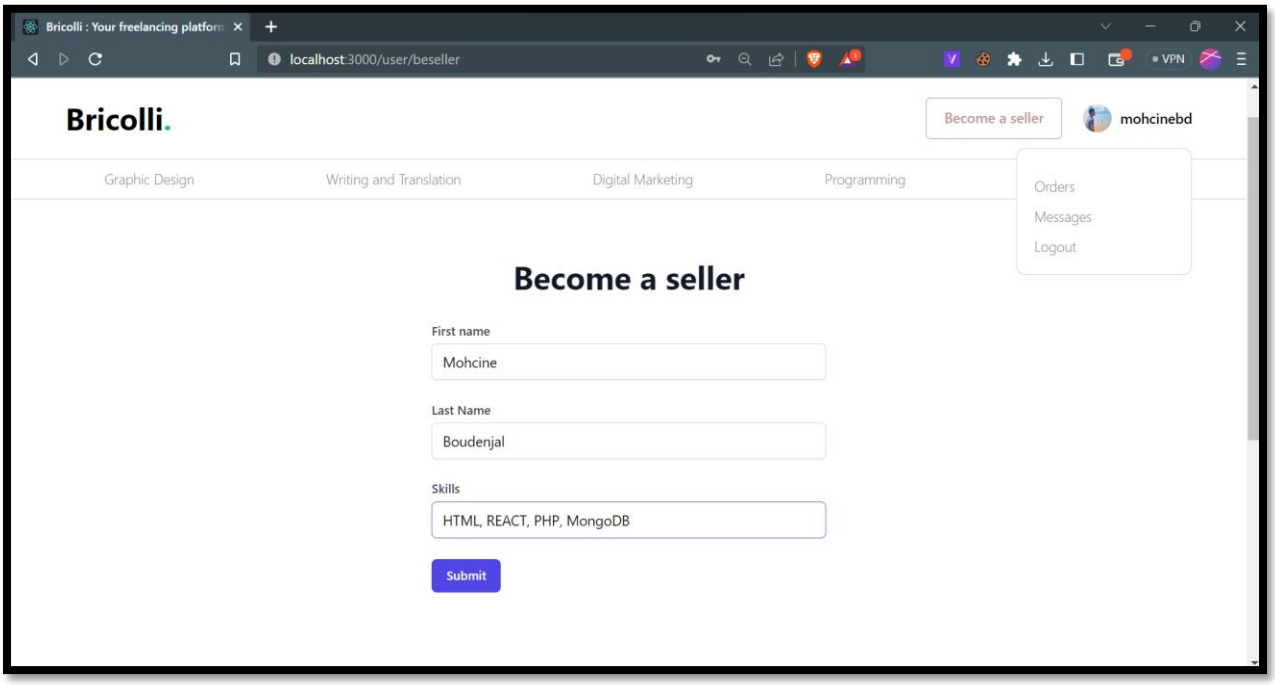
Login page appear like this



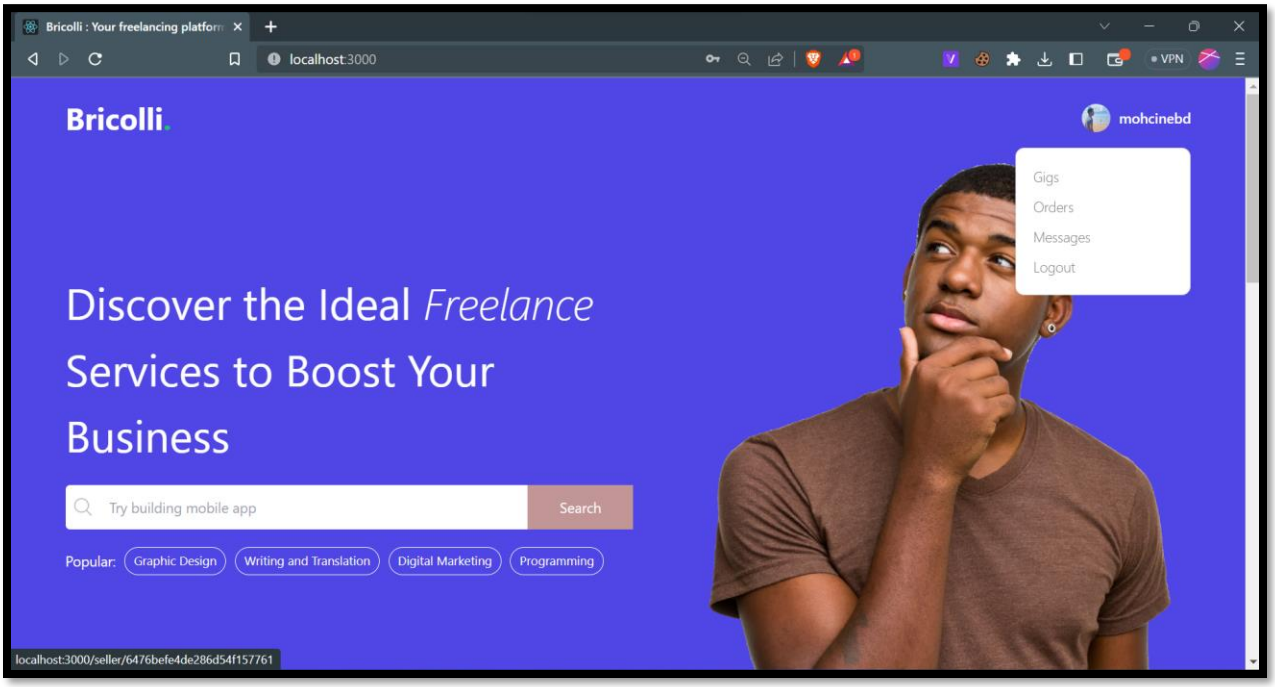
This is the navbar after the login, but the user still not a seller so he will fill the form to do it



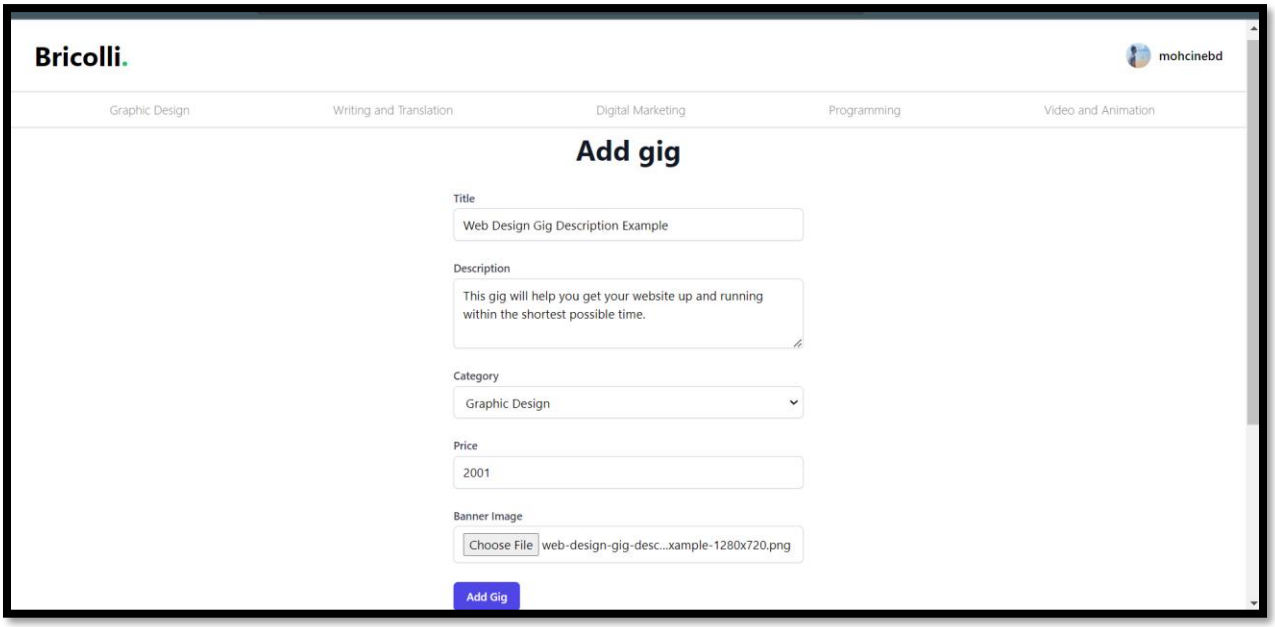
The user should fill this form



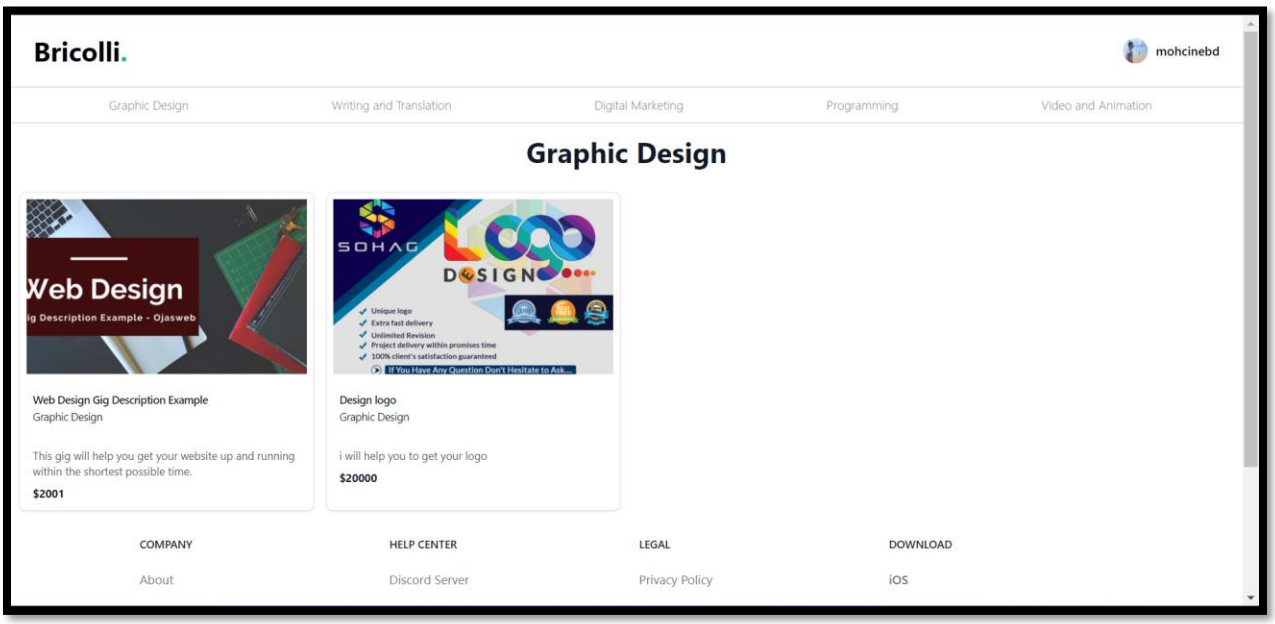
So, he is now a seller and he can manage his gigs



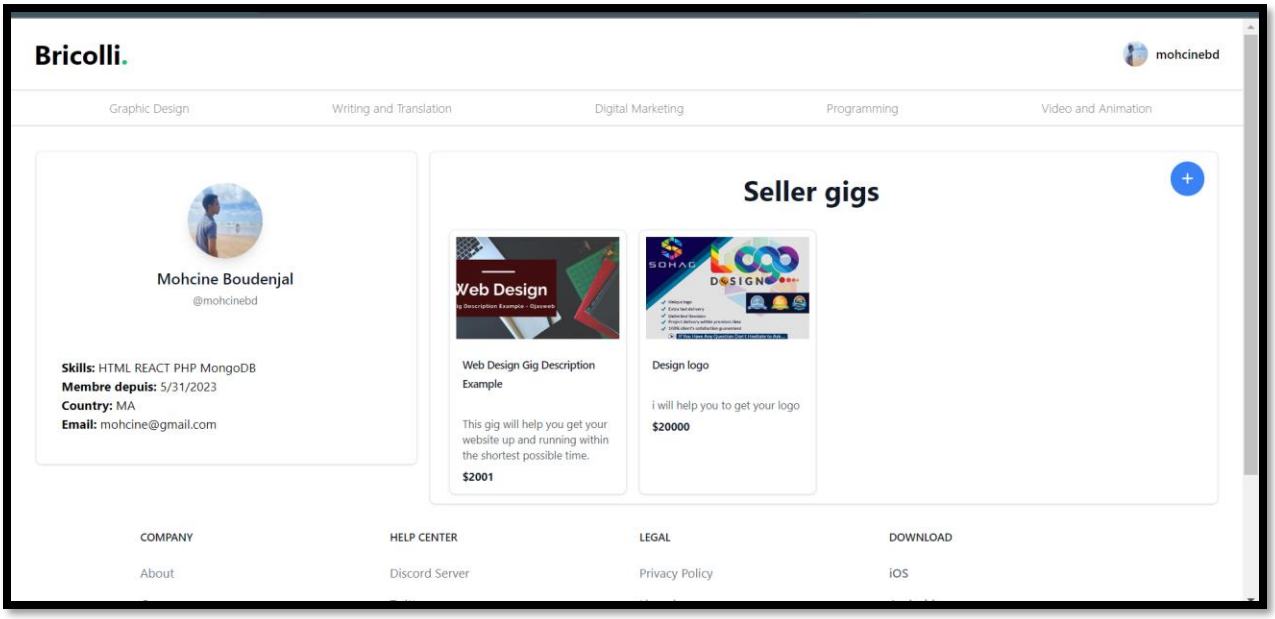
Add new gig



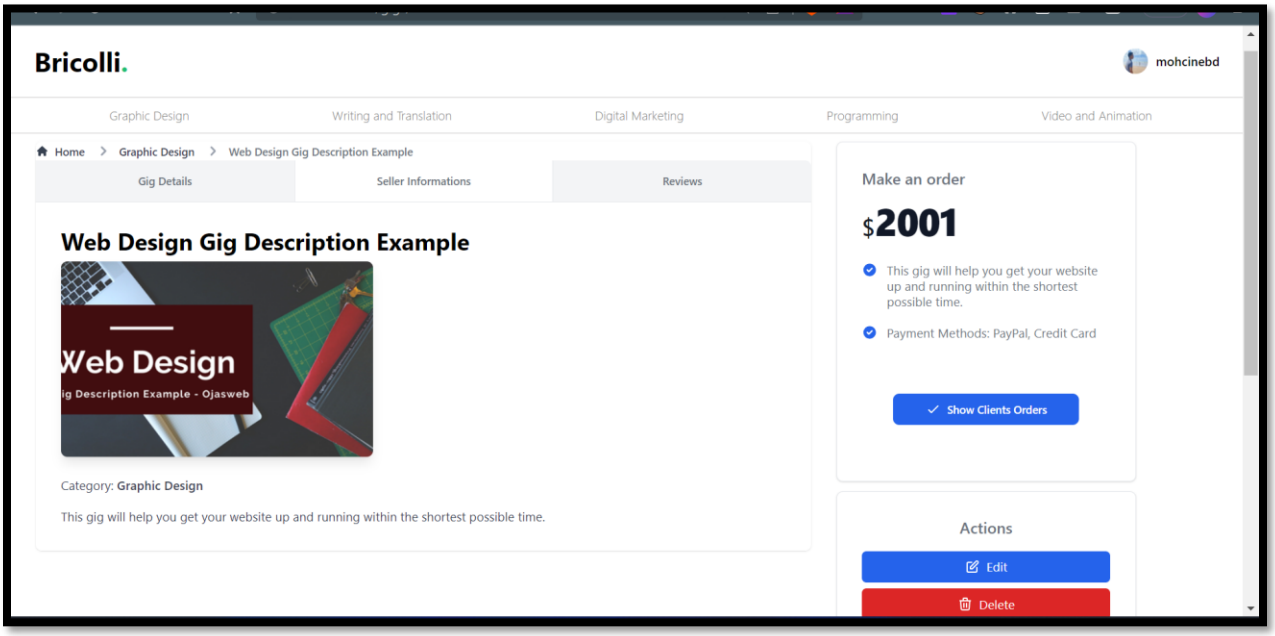
The gig created successfully, here we can show the gigs by category



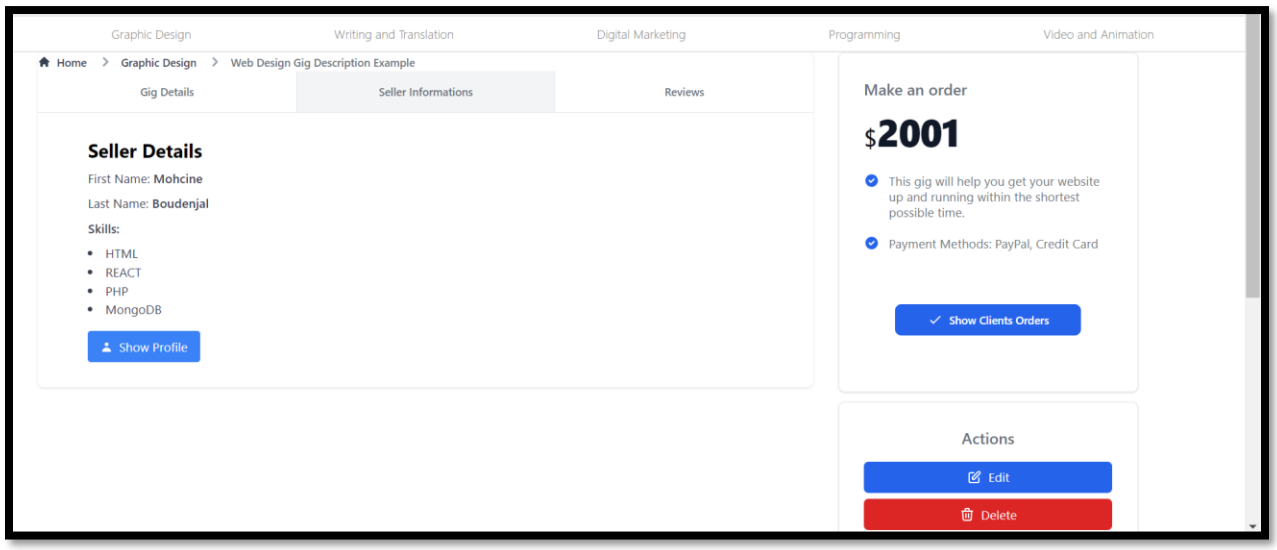
Profile of a seller



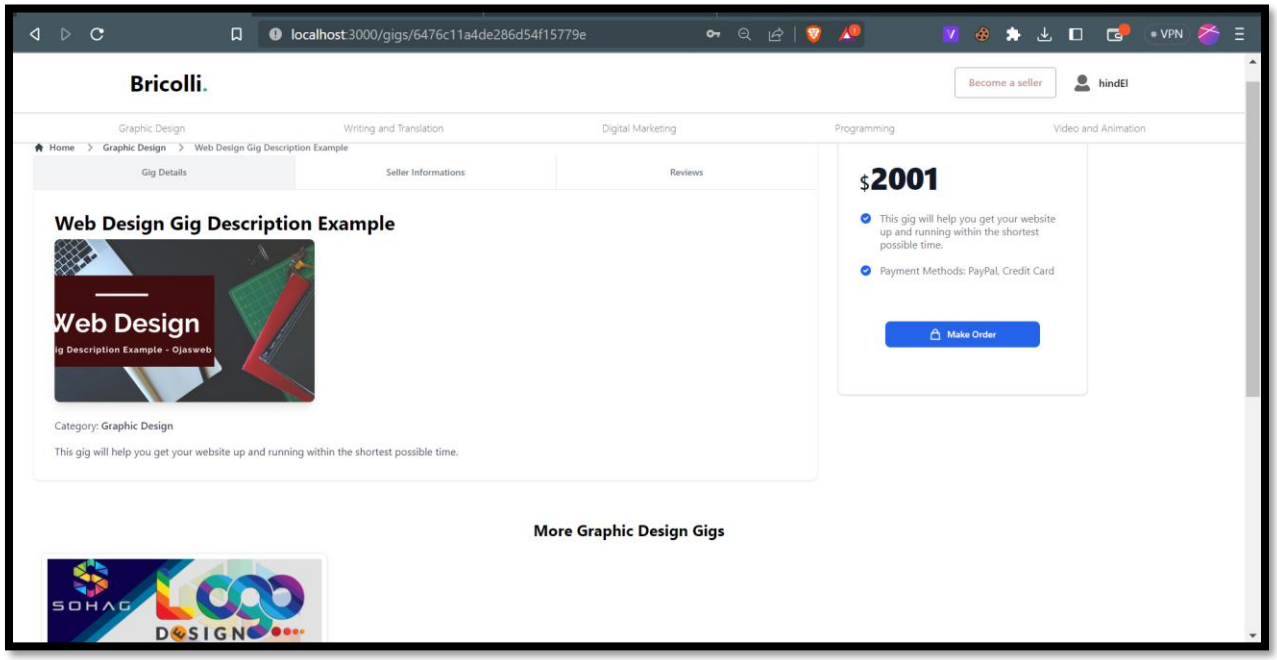
Gig details:



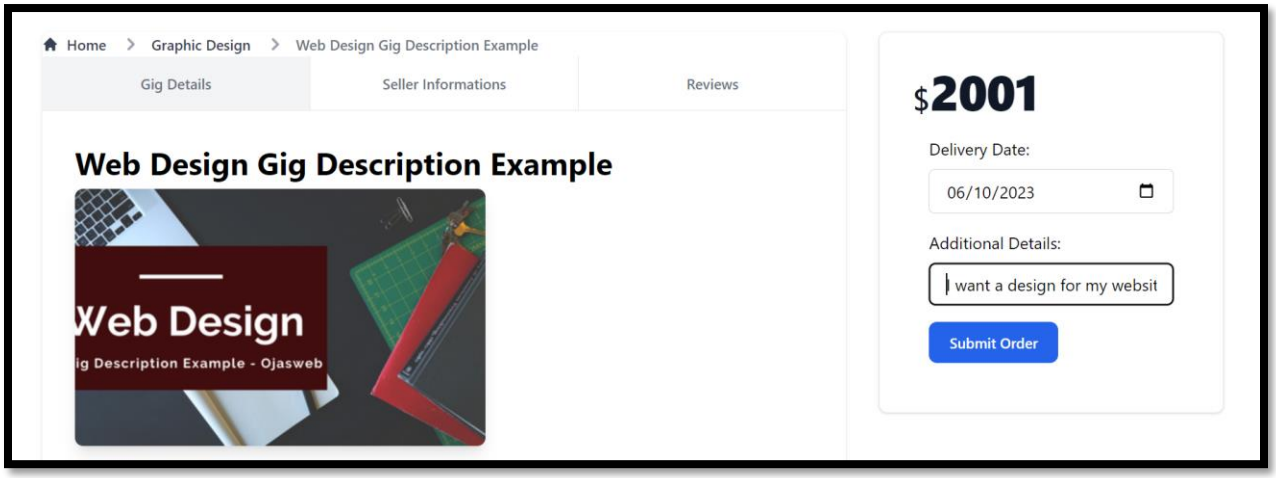
He can edit or delete it or visualize the client orders



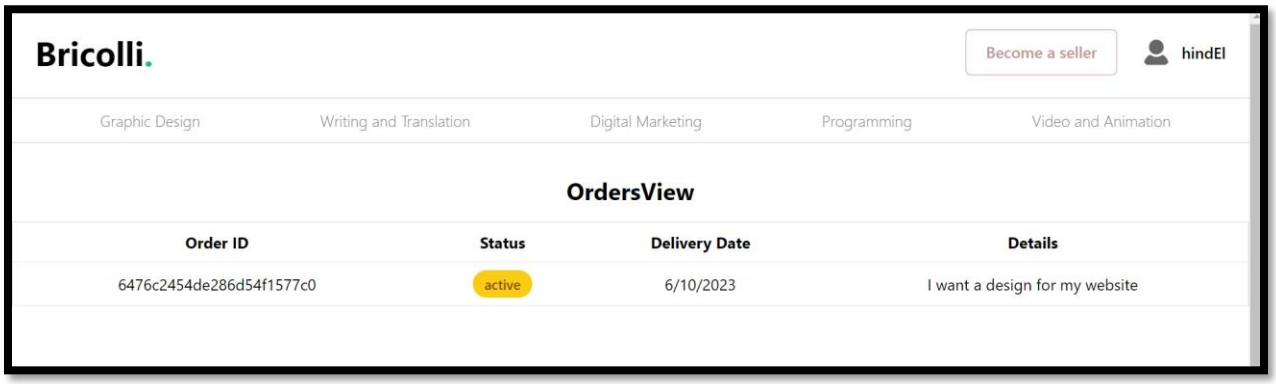
Let’s connect with other account (Hind account) and make order to this gig



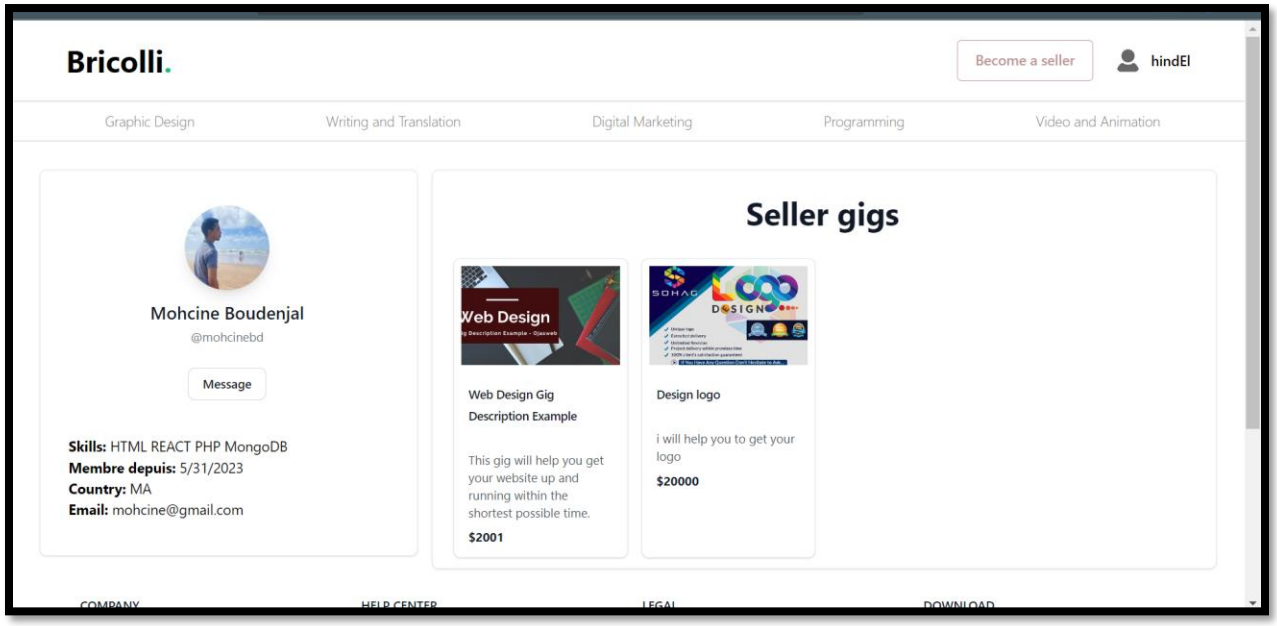
The buyer should fix the deadline and additional details



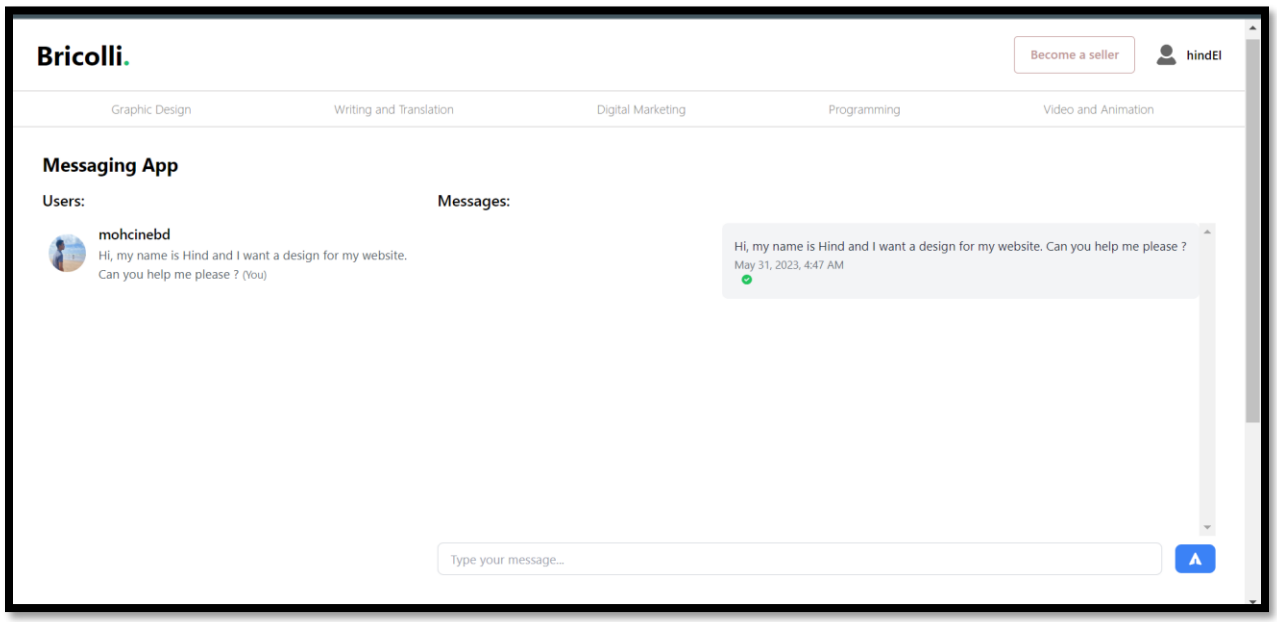
We can imagine that payment done and this is the orders of buyer, it's pending now and the seller should approve or cancel it



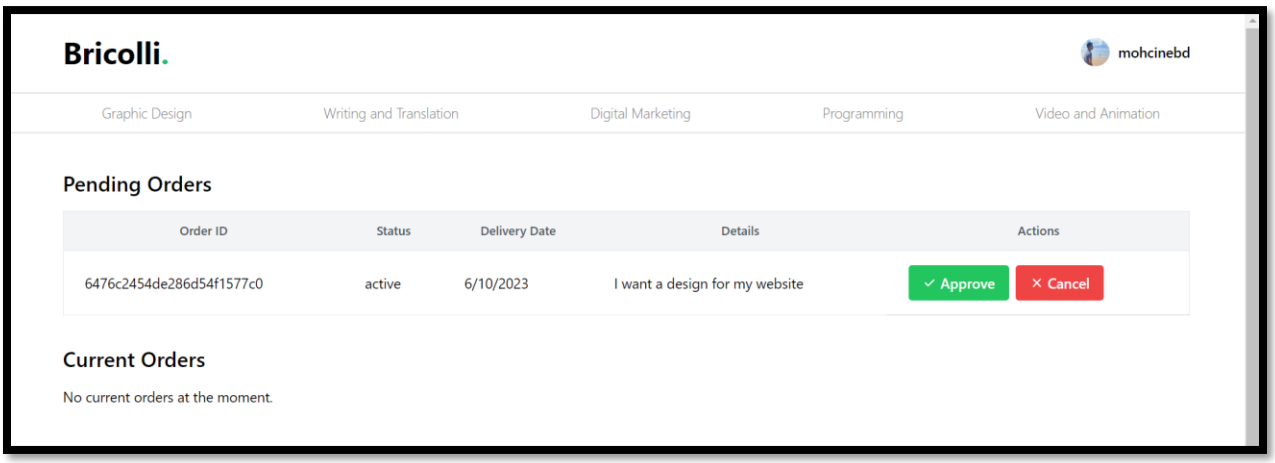
This is the view of another profile, look at the message section that create a conversation



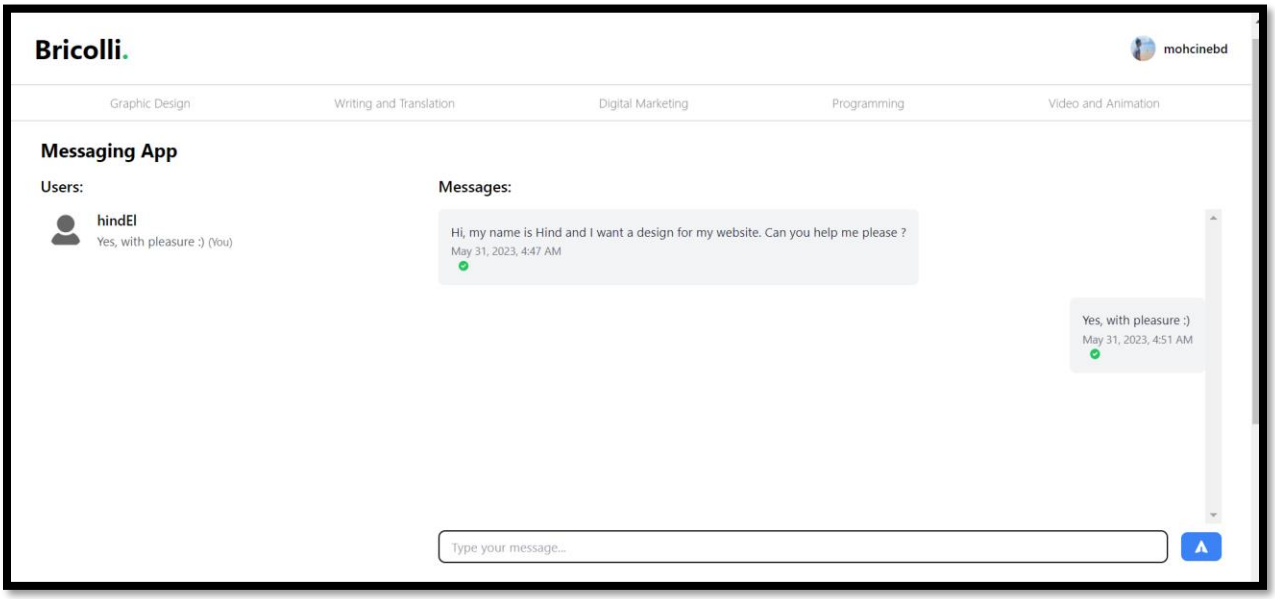
The buyer sends a message to a seller and the conversation began



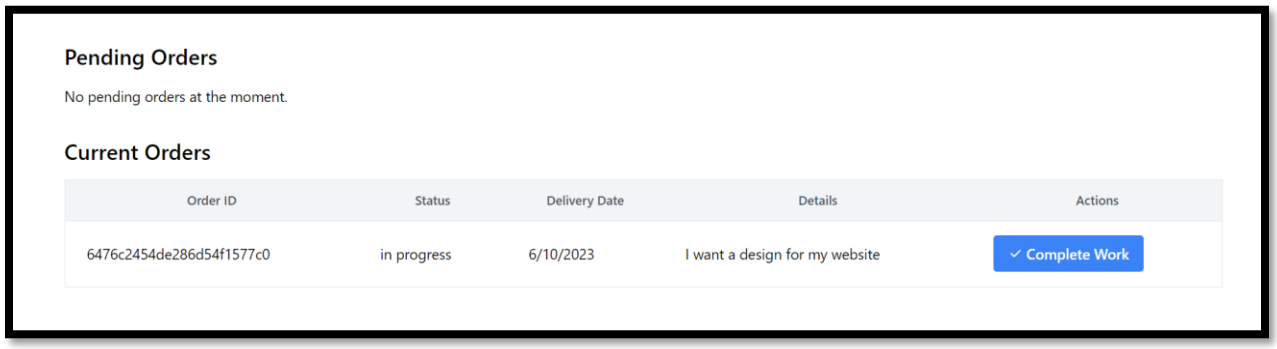
Back to the seller, he has a request of work so he can approve it or cancel it



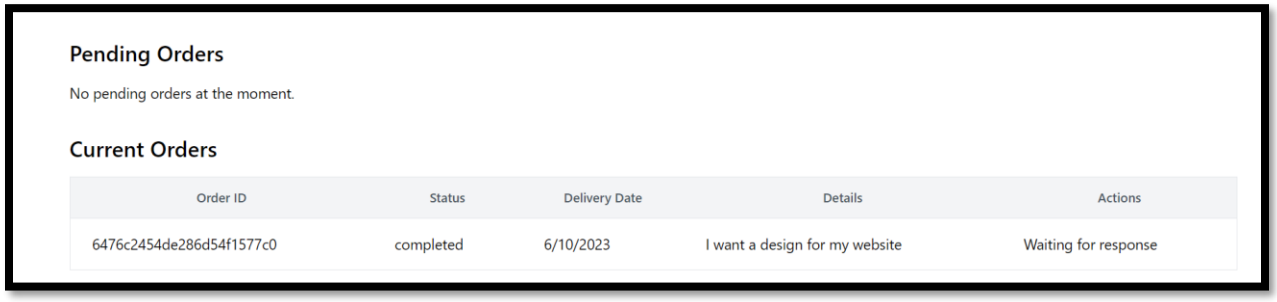
The seller read and interact with buyer



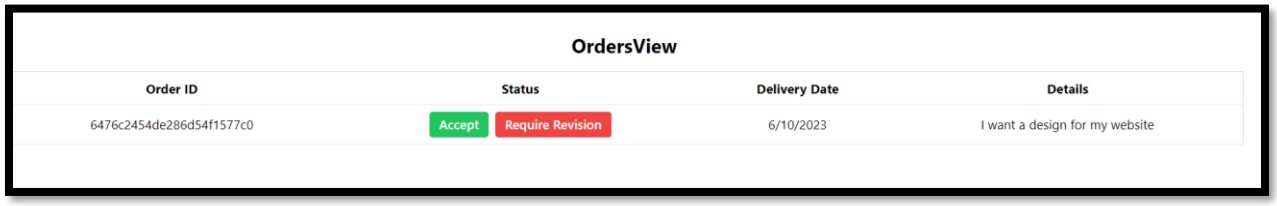
When he accepts, the work is in progress in view of buyer and waiting for seller that complete his work



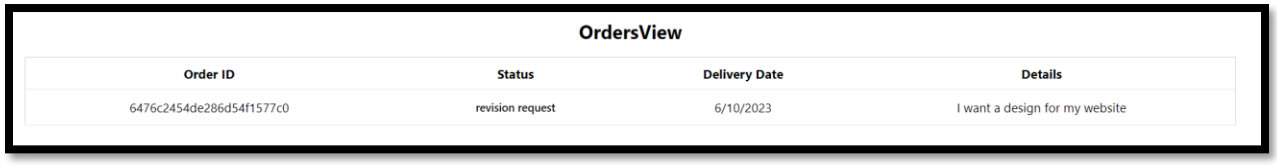
He completes his work and he wait response from buyer even accept or request revision



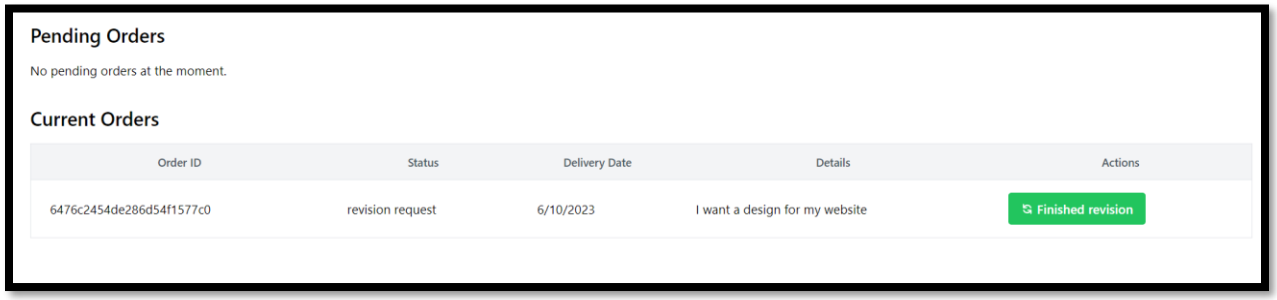
In view of buyer:



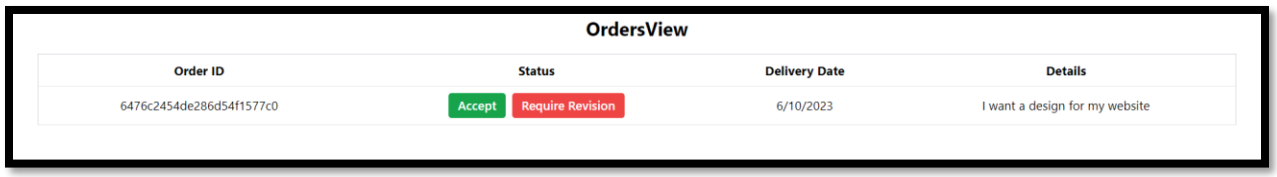
If he requests revision of work:



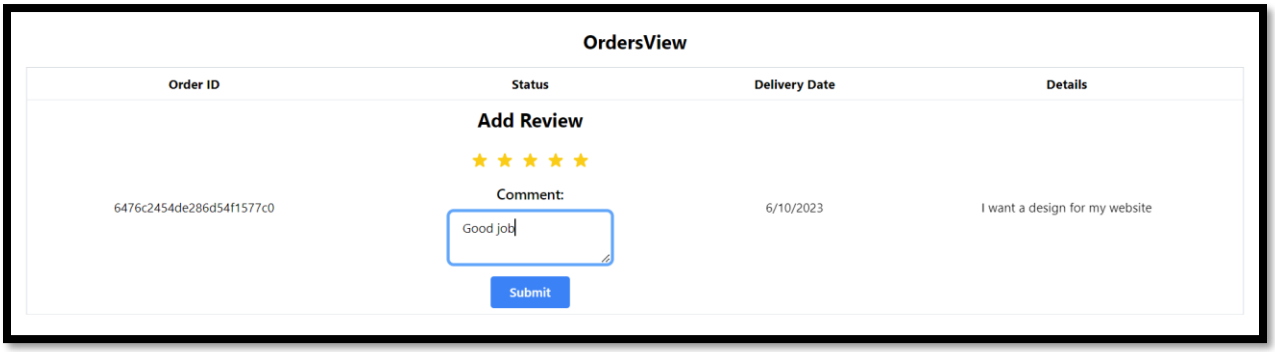
The seller makes changes and then finished revision:



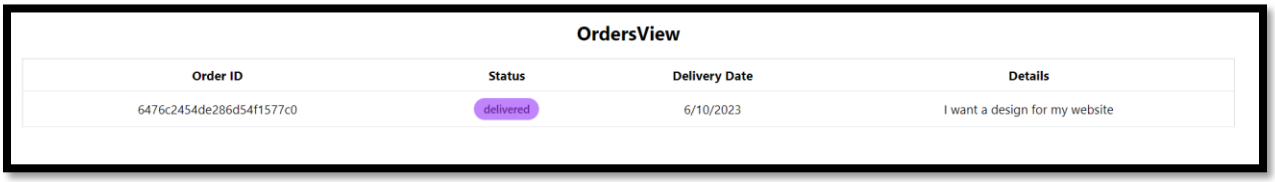
If all things are good, buyer accept the work and pass to make a review



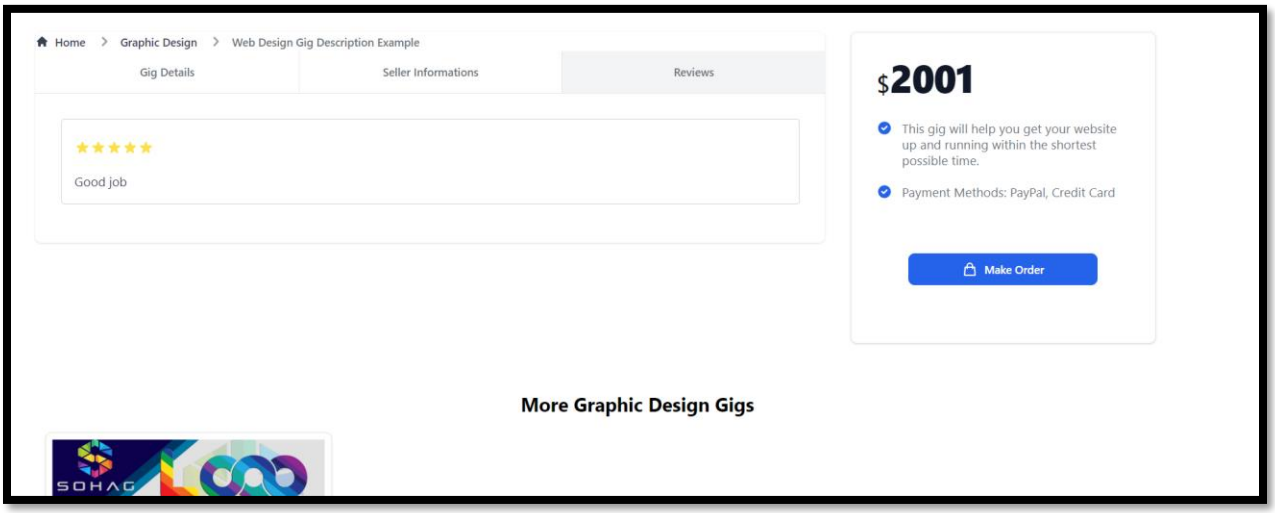
Rating and comment:



So, the order is delivered now



In details of gig, we can see the reviews



X. Workflow:

GitHub played an important role in our team's collaboration by serving as a centralized platform for storing and tracking our progress. We utilized its features to manage our code repository and make the development process smoother. If you're interested, you can access our repository at

https://github.com/S2-Projects/clone-fiverr.

XI. Perspectives:

We have completed most of the features for our application based on the timeline. However, there are still some important features that need to be added in the future to improve user experience. These features include:

- Payment feature: that will allow our application to act as a secure and reliable intermediary between sellers and buyers. Implementing a payment system will enable seamless transactions within the application

- Notification: Implementing notifications is essential for a progressive web application, as it keeps users informed about important updates, messages, or actions related to their activities on the platform.

- Attachment files in messages: Implementing notifications is essential for a progressive web application, as it keeps users informed about important updates, messages, or actions related to their activities on the platform.

- Messages filtering: Implementing notifications is essential for a progressive web application, as it keeps users informed about important updates, messages, or actions related to their activities on the platform.

These are just a few examples of the additional features we plan to implement to create a more comprehensive and perfect application.

XII. Conclusion:

In conclusion, our project is a progressive web application that combines practical improvements and modern technologies to deliver a seamless user experience. We have utilized ReactJS, Redux, TailwindCSS, and Heroicons for the frontend, along with Node.js, Express, GraphQL/Apollo and MongoDB for the backend.

Throughout development, we have relied on various npm packages such as Framer Motion, Apollo Client, React Router DOM, and React Redux to enhance functionality and streamline the development process.

GitHub has served as our collaboration platform, enabling us to store code and track progress.

While we have completed most features, there are important additions planned for the future. These include implementing payment functionality, notifications, attachment files in messages, and message filtering for privacy.

By continuously improving and expanding our feature set, we aim to create an exceptional application that provides a seamless and engaging experience for users.

We would like to express our sincere gratitude to Professor **Mr. ELAACHAK Lotfi** for assigning this project. We appreciate the opportunity to work on this project and have learned valuable skills and knowledge in the process.

XIII. Bibliography:

https://www.npmjs.com/

https://react.dev/

https://www.howtographql.com/

https://www.apollographql.com/docs/apollo-server/

https://create-react-app.dev/docs/making-a-progressive-web-app/

https://openclassrooms.com/fr/courses/6390246-passez-au-full-stack-avec-node-js-express-et-mongodb

https://flowbite.com/docs/getting-started/introduction/

https://heroicons.com/