## Internship Report

**Intern:** Bruno SHEMA

**School:** European University of Lefke

**Student Number:** 22243402

**Faculty:** Software Engineering

**Company:** Fidalix.ltd

**Project:** Delivery website

**Supervisore:** Ir Jean de Dieu Nizeyimana / Irenee Munyaneza

**Position:** Software Engineer / Project Manager

**Started: 12/07/2025**

**Ended: 11/08/2025**

### Acknowledgment

I thank Fidalix.ltd team for giving me the opportunity to work as their intern for 30 working days, I learned a lot under their guidance. I am especially grateful to my supevisor Ir Jean de Dieu Nizeyimana / Irenee Munyaneza, who was supportive at every step and helped me put my academic knowledge into practice. I also thank my school for arranging this valuable summer training, which improved my practical skills and strengthened my career.

### Abstract

This report represent the development of a full-stack delivery website using , React, Node.js,Tailwindcss, Express, sequelize,Mysql , postman and git. Thewebsite has features like:

**Client side:**

- shops browsing,

- odering goods

**Shop side:**

- view orders made & serve them

- add products into their store

**Admin:**

- view all users & delete any

- view all partners & add or delete any

- view a complete history of orders made.

### Table of Content

Catalog

[Internship Report 1](#_Toc30980)

[Acknowledgment 1](#_Toc31202)

[Abstract 1](#_Toc30280)

[Table of Content 1](#_Toc24454)

[Introduction 2](#_Toc13867)

[Project Overview 2](#_Toc12810)

[System Design & architecture 3](#_Toc23093)

[Architecture 3](#_Toc31946)

[MVC architecture in a Delivery website 3](#_Toc14461)

[System Architecture: 4](#_Toc32005)

[Run the project locally: 4](#_Toc25275)

[Database Schema: 4](#_Toc2336)

[Implementation 5](#_Toc31499)

[Challenges & solution 19](#_Toc29418)

[Testing & Validation 21](#_Toc27546)

[Conclusion 22](#_Toc17680)

[Future Work 22](#_Toc205)

[References 23](#_Toc32035)

### Introduction

**Company:** Fidalix.ltd

**Location:** \_\_\_\_\_\_\_\_\_\_

**Field Area:** tech -----

**My Role:** Full-stack developer

**My goal:**  To build a responsive website for deliverying goods

### Project Overview

**Project title:** TUMA website (a delivery website)

Meaning: TUMA - send

**Purpose:** Facilitate the delivery services my locality; flexible with user’s schedule. Feasible even with local markets. Later expandable to being cross countries service.

**Tools used:**

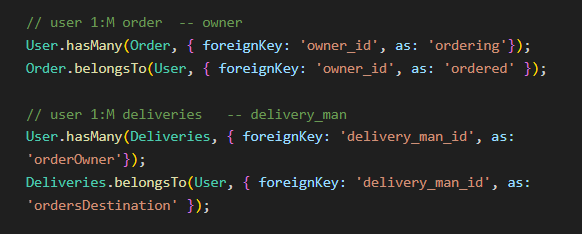
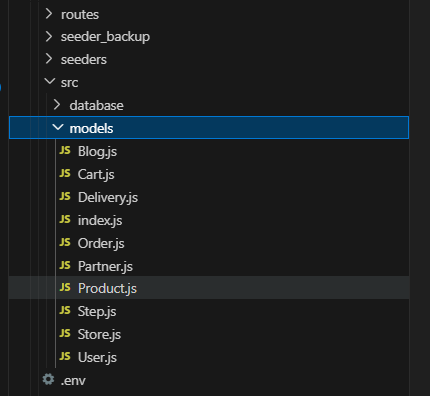
* Database: Mysql (*local database only used)*
* Backend:
  + Node.js // to run javascript
  + Express // for routing
  + Sequelize // for safe query onto Mysql
* Frontend:
  + React.js // to use javascript like files instead of html .jsx
  + Tailwindcss // for pre-made css class , to speed up the process
  + Lucide-react // icon components library full customizable
  + Frame-motion // for better css animation on to tags
* Others
  + Postman // to test the routes in the backend
  + Git // for pushing my work onto github

### System Design & architecture

#### Architecture

##### MVC architecture in a Delivery website

**Model --Backend** *feature* **:** refers to the structure of our data in the database, relation tables and relevant constraints around them. Using **ORM** Sequelize; I had to define the tables, and create assocciations between them, most importantly name the assocciation **--*in index.js*** file; as it changes how we access data.



**View --Frontend** *feature* **:** refers to fetching data and sending user input to&from the controller and rendering the relevant information.Using **React Components** I arranged how the in coming information will be displayed and how to retrieve user input and send it to the controller, Dynamically; By the help of States.

**Controller --Backend** *feature* **:** refers to manipulating the models or the stracture of our databse to provide what user needs. Using **Express** I created routes which are the end points the frontend will be hinting on for fetching the data, or sending some information;For communicating with the front end I sued **Restful API** an application program interface, that follows **Rest** architecture style.Rest*’’Representation state transfer’’* which uses Http/1.1 protocal. With **crud** operation implementation http requests.

**Example od Rest API in our project:**

* **GET /history** : this gets the recordings made on the order, who ordered, to which shop, who delivered.
* **GET /partner :** this retrieves all partners
* **POST /register/partner:** inserts partner
* **POST /register/product** :this inserts a new product in the database
* **POST /Login** : this handels the login lodic; why post: as I wanted to pass in body parameters it can not be get.
* **POST /user:** add in the user,
* **DELETE /user/:id :**  this deletes a specific user-- depending on the id
* **PUT /store/price:** this changes the price of a product, specific to which shop is in.

##### System Architecture:

* **Frontend (**client Side)
  + Built using React.js
  + Provides login form for( customer, business,delivery) a special access for admin ;
    - Customer: has an advertising blog ; ordering section, see partners, and contuct Us page
    - Business: see the orders made; add product in store
    - Admin: see all user(can delete them) ; see the full history so far ; add in partners
  + I was Sending HTTP request to the backend through a Restful API.
* **Backend (**Server side**)**
  + Built using Node.js and Express
  + Routes the end points and handles the querying logics:
    - Retrieve users, partners,products … to be rendered in frontend
    - Modifies the price of products
    - Deletes users based on their ids
    - Save new users into the database
* **Database**
  + Used Mysql , in storing users,products, partners ,and the path to where the profile images are saved.
    - Here I used **multer** to save images locally in public/images file and sabe the path in the database, instead of saving the image; this required to use a midleware in the backend routes; and after inserting a record with empty value on the path; I name the image, the user id and name and save it in public.image file, then update the path to the image.
  + Only accessed by the controller( backend).

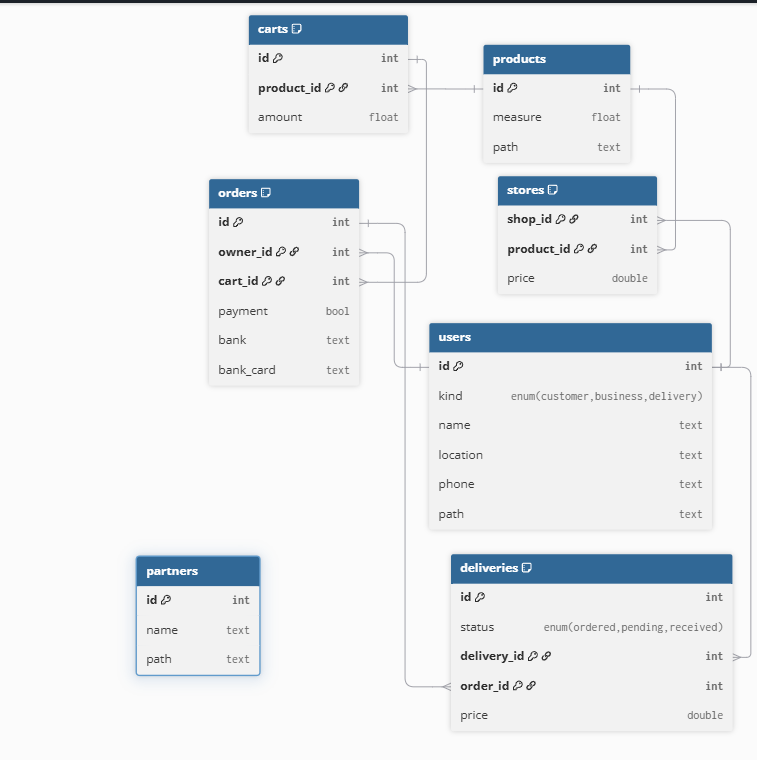
**Data Flow:** User sends request via Http request from frontend -> handled by Express through routes( end point) -> DB fetch/update is done -> [where nesseccary midleware is done] -> response sent back to frontend -> using frame-motion,tailwindcss, React the data is rendered.

##### **Run the project locally:**

1. npm I
2. Update the .env file in you backend: require DB\_HOST; DB\_USER; DB\_PASSWORD;DB\_NAME and SERVER\_PORT
3. Change backend/config/config.json : the password, username, database and dialect
4. Sequelize db:migrate // to create migrations
5. Most importntaly to run seeders: move all other files in sedder\_backup leave one file tu run; run then in assending order; run this command. sequelize db:seed:all
6. Then in backend folder: node.js --watch server.js
7. In the frontend folder: npm run dev
8. You are good to go.

##### **Database Schema:**

**ER-Diagram:**



**Schema:**

users (**id**, *kind* ,name,location,phone,path);

products (**id**,measure,path);

stores (**shop\_id,product\_id**,price);

carts (**id**,product\_id,amount);

orders (**id**, owner\_id, cart\_id, payment,bank,bank\_card);

Deliveries (**id,***status,*order\_id,delivery\_id,price);

partners:(**id,**name,path);

***Legend***

Bold: id primary key

Underlined: is refrence key

Shop\_id ,delivery\_id and owner\_id **ref**  users ; on business,delivery and customer *kind*  respectively

Product\_id **ref**  products

cart\_id **ref** carts

order\_id **ref** orders

Italic: is enum class

Kind(“customer”,”business”,”delivery”)

Status(“odered”,”pending”,”received”)

### Implementation

1. **Login**
   1. **Customer && business**

    app.post('/login', async (req, res) => {

      try {

        const { name, password } = req.body;

        const user = await User.findOne({ where: { name: name } });

        if (!user) {

          return res.status(404).json({ Error: `No user called ${user\_name}, recorded` });

        }

        const ans = password === user.password;

        if (ans) {

          res.status(200).json({ msg: "Logged in successfully", user });

        } else {

          res.status(200).json({ msg: "Wrong password" });

        }

      } catch (error) {

        res.status(500).json({ Error: `${error.message}` });

      }

    });

Using this API frontend send the name and password , API gives the msg and the user object important for next frontend url destination.

    const handleLogin = async () =>{

      if(!name||!password){

        alert(`all fleids are required`);

        return;

      }

      try {

        const response = await fetch('http://localhost:3010/login',{

          method: "POST",

          headers: {

            "Content-Type": "application/json"

          },

          body: JSON.stringify({ name, password })

        });

        const result = await response.json();

        if (result.msg === "Logged in successfully") {

          localStorage.setItem("user", JSON.stringify(result.user));

          alert(`logged in successfully`);

          handleclicklogin();

        }else{

          alert(`Error: ${result.msg}`);

        }

      } catch (err) {

        alert(`Login failed, Error: ${err.message}`);

      }

    }

And once the msg is ” logged in successfully “ we go to the next page ; but we first save the user object in the local storage to access it later. When we reach the next page.

* 1. **Admin**

import { useNavigate } from 'react-router-dom'

const Login = () => {

    const navigate = useNavigate();

    const [kind,setKind] = useState('customer');

    const handleclick= (value) =>{

        setKind(value);

    }

  return (

    <div className='flex flex-col items-center gap-7'>

      <div className='flex w-full justify-end'>

        <button className='bg-primary h-[65px] w-[65px]  right-0 flex justify-center items-center text-white text-lg rounded-xl cursor-pointer'

        onClick={()=>navigate('/admin')}

        >Admin</button>

      </div>

For Admin there is just a button to click and access the admin url linked to admin page.

* 1. **Register**

Once for new users,they have an option to register,

  const handleclickRegister = () =>{

    navigate('/register');

  }

            <button

          className='bg-primary text-white'

          onClick={()=>handleclickRegister()}

          >Register</button>

      </div>

Onclick we move to the login page which is just a login form;

    const [name,setName] = useState('');

    const [location,setLocation] = useState('');

    const [phone,setPhone] = useState('');

    const [password1,setPassowrd1] = useState('');

    const [password2,setPassowrd2] = useState('');

    const [kind,setKind] = useState('');

    const [path,setpath] = useState(null);

Repair the states of every filed.So now we plan to use a midleware for image upload we use FormData to make our body.

    const handleSubmit = async () => {

    if (password1 !== password2) {

      alert('Passwords do not match!');

      return;

    }

    const formData = new FormData();

    formData.append('name', name);

    formData.append('location', location);

    formData.append('phone', phone);

    formData.append('password', password1);

    formData.append('kind', kind);

    formData.append('profile\_path', path);

Then we fetch on the relevant end point.

    try {

      const response = await fetch('http://localhost:3010/user', {

        method: 'POST',

        body: formData,

      });

If everything is ok, we clear the input fields and go back to the login pages ‘/’

      const result = await response.json();

      if (response.ok) {

        alert('Registered successfully');

            setName(''),

            setLocation(''),

            setPhone(''),

            setPassowrd1(''),

            setPassowrd2(''),

            setKind(''),

            setpath(null)

        navigate('/');

      } else {

        alert(`Registration failed: ${result.Error || 'Unknown error'}`);

      }

    } catch (error) {

      console.error('Registration error:', error.message);

      alert(`Error: ${error.message}`);

    }

  };

1. **Customer**

So in here on url ‘/Home’

We retrieve the user object we save in the localstorage and pass it as params in the components that needs it.

const Home = () => {

const [userValues, setUserValues] = useState({});

  useEffect(() => {

    const user = JSON.parse(localStorage.getItem("user"));

    if(user){

      setUserValues(user);

    }

  }, []);

  return (

    <div className='flex flex-col'>

        <Navbar user={userValues}/>

        <Blog/>

        <Order Id={userValues.id}/>

        <Partner/>

        <Contact/>

        <Footer/>

    </div>

  )

}

* 1. **Navbar**

Here nothing much: just a logo, links logout\_button and profile; the profile is why we sent user object here.

Only the cool feature is the links are not to navigate to the other page, it just scrolls into the view of some tags with relevant id;

  const links = [

    { text: "Best Deals", id: "blog" },

    { text: "Press Order", id: "order" },

    { text: "Our Community", id: "partner" },

    { text: "Contact Us", id: "contact" },

  ];

We prepair the links alongside their attached tag ids. That we are targeting;

These tags are the div in other components, one in Blog component, another in Order,Partner,Contact,Footr… like :

const Blog = () => {

  return (

    <div id="blog" className='grid w-screen h-[360px] bg-orange-300'>

And

const Order = () => {

  return (

    <div

      id='order'

      className='h-[700px] w-full text-black bg-secondary flex jus

* 1. **Blogs**

Here I use hardcoded information;

const titles = [

  "Reliability & Fast",

  "Safe and Professional Handling",

const descriptions = [

  "Every time, your schedule matters.",

  "Product in safe hands; trained riders handle every order with care.",

const imagepaths = [

  "/images/fastReliable.jpg",

  "/images/wellpackaging.jpg",

const blogs = titles.map((title, index) => ({

  title,

  description: descriptions[index],

  image: imagepaths[index]

}));

And made a blog object.

Then I accessed It in the Blog componet

import blogs from '../lib/BlogLib.js'

Of every index pass in this component I display relevant info:

const blog = ({pk}) => {

    const Id = parseInt(pk) % 6;

    const one = blogs[Id];

  return (

    <div className='flex justify-evenly items-center  h-[280px] gap-5'>

        <div className='flex flex-col gap-1 w-[600px] max-h-[500px]'>

            <div className='font-bold underline text-lg'>{one.title}</div>

            <div>{one.description}</div>

        </div>

        <div className='max-w-[300px] h-[400px]'>

        <img src={one.image} alt="blog image" className="w-full h-full object-contain rounded-md" />

        </div>

    </div>

  )

I used modulus 6 so it will only beable to have [01,2,3,4,5] values whether subtracted or added exessively.

Then in Blog section through an infinity loop of 8 sec interval I keep increasing the id passed in the blog component from 0 till 5; so id I had no other tags whose display depends on index being between [0 - 5] I would not have guarded the value as the blog component, ensures no error using % moduls. But the green dots, their css depends on 0-5 range.

  const [id, setId] = useState(0);

  const handleNext = () => {

    setId(prevId => (prevId === totalBlogs - 1 ? 0 : prevId + 1));

  };

I guard the value, and allow to change the id as we click on any of the dots

  useEffect(() => {

    const interval = setInterval(() => {

      handleNext();

    }, 8000);

    return () => clearInterval(interval);

  }, []);

Make a loop

* 1. **Order**

**So** now here I used a custome global context where in one file we can create a state that is available in the whole project. In src/context folder I create a user context; I use **creatContext** from react

import { createContext, useState } from 'react';

const UserContext = createContext();

Create a context call UserContext

  const [card,setCard] = useState([]);

Create a state you want.

  return (

    <UserContext.Provider value={{

      user, setUser,

      card,setCard

     }}>

      {children}

    </UserContext.Provider>

Here we specify the whatever becomes our child will access these values including the state ans setting the state function; here one can limit what he likes.Then we wrap the App around this component. In that case everything in the project atleast frontend , get access to these.

import { UserProvider } from './context/UserContext';

ReactDOM.createRoot(document.getElementById('root')).render(

  <React.StrictMode>

    <BrowserRouter>

      <UserProvider>

        <App />

      </UserProvider>

    </BrowserRouter>

  </React.StrictMode>

Then coming in the order section: we have

Searchbar// not functional yet though

A business Table component

  </div>

        <div className='bg-blue-300 h-[500px] w-[66vw] flex'>

          <BusinessTable />

        </div>

      </div>

In this componet we fetch users and render them

  const [rows, setRows] = useState([]);

  const headers = ["id", "name", "kind", "location", "phone"];

  const refined = rows.map((row) =>([

    row.id,

    row.name,

    row.kind,

    row.location,

    row.phone

  ]))

  useEffect(() => {

    const fetchBusiness = async () => {

      try {

        const res = await fetch('http://localhost:3010/user/business');

        const data = await res.json();

        setRows(data);

      } catch (error) {

        console.error('Fetch error:', error);

      }

    };

    fetchBusiness();

  }, []);

Then use a table componet to render them

      <Table rows={refined} headers={headers}/>

    </>

In here after receiving the titles and rows, we trim them so if the column is not provided, things won’t crash just it values won’t display

  const minLength = Math.min(headers.length||0, test.length || 0);

  const trimmedHeaders = headers.slice(0, minLength);

  const trimmedRows = rows.map((row) => row.slice(0, minLength));

Then the table will have paginations, and a maximum of 5 row per page

  const handleClick = (visibleRowIndex) => {

    const realIndex = page \* rowsPerPage + visibleRowIndex;

    const row = rows[realIndex];

    const rowId = row[0];

We click a given page, we update the rows to display

  const handleClick = (visibleRowIndex) => {

    const realIndex = page \* rowsPerPage + visibleRowIndex;

    const row = rows[realIndex];

    const rowId = row[0];

We calculate the maximum pages we are to need

 const rowsPerPage = 5;

  const totalPages = Math.ceil(trimmedRows.length / rowsPerPage);

  const visibleRows = trimmedRows.slice(

    page \* rowsPerPage,

    (page + 1) \* rowsPerPage

  );

Then display.

Down in this Table component; near the end there is a conditional rendered product componnet, ; it comes when you click on a business saying Shop; it will shop products that this shop has

 {visible && (

        <div className="absolute left-[10px] top-[20px] bg-blue-200/90 text-black rounded-lg border border-white border-4 h-[500px] w-[64vw] flex items-center flex-col">

          <button

            className=' absolute right-0 h-[8vh] w-[3vw] flex items-center justify-center text-white text-xl bg-red-500/70 hover:bg-red-700'

            onClick={handleClickreverse}>

              x

          </button>

          <h3 className='font-semibold underline text-xl'>Shop</h3>

          <div>

            <Products items={[`${infoVisible[0].id}`]}/>

          </div>

        </div>

      )}

Going into this component: cut things short, it follows nearly the same logic of paginations; then it fetch products to render.

  useEffect(() => {

    const fetchProducts = async () => {

      try {

        const res = await fetch(`http://localhost:3010/user/business/owner/${index}`);

        const data = await res.json();

        setProduct(data);

      } catch (err) {

        console.error(`Error: ${err.message}`);

      }

    };

no

Now once we click on order , of a product

           whileHover={{ scale: 1.03 }}

                    whileTap={{ scale: 0.95 }}

                    className="text-sm h-[40px] w-[60px] flex items-center justify-center bg-accent text-primary font-bold mt-2"

                    onClick={() => handleClick(item.id)}

                  >

                    order

                  </Motion.button>

The global context plays the role now; as now we are very deep in the nested component to move back the card information in the order component would be hectic.

import { UserContext } from '../context/UserContext';

Then we use that context realying on useContext from react

import React, { useEffect, useState, useContext } from 'react';

 const { setCard } = useContext(UserContext);

So now on the click of order, the card state will be changes and whoever accesses it will see the changes directly.

  const handleClick = (id) => {

    const selectedItem = product.find(item => item.id === id);

    if (selectedItem) {

      setCard(prev => [...prev, selectedItem]);

    }

  };

We take whatever the state had and append our new value, we distruct the previous contents and add our own and reset the card with that.

**Then** back into the order component

Here we bring in both card and setCard from userContext

  const { card, setCard } = useContext(UserContext);

This is for accessing card, the setcard is for setting is toempty once user orders and pays..

Mean while if nothing has be kept in the card yet we tell the user that

  <div className='h-[700px] w-[30vw] min-w-[300px] bg-blue-300 p-4 flex flex-col items-center overflow-y-auto overflow-y-scroll'>

        {Array.isArray(card) && card.length === 0 ? (

          <p className='text-gray-700 text-center'>No product selected yet.</p>

        ) : (

Every cart has an option of changing the amount, put limit it to be 1 and above

    const handleAmountChange = (e) => {

                  const amount = Math.max(Number(e.target.value), 1);

                  const updated = [...card];

                  updated[index] = { ...updated[index], quantity: amount };

                  setCard(updated);

                };

As the card “typo - cart” changes we change the total price so far the user is due to pay

 useEffect(() => {

    if (Array.isArray(card)) {

      const newTotal = card.reduce((sum, item) => {

        const unitPrice = item?.storing?.[0]?.price || 0;

        const quantity = item.quantity || 1;

        return sum + unitPrice \* quantity;

      }, 0);

      setTotal(newTotal);

    } else {

      setTotal(0);

    }

  }, [card]);

Then once the order is made we reset everything to empty and message what happened; but nothing else is affect after this

/div>

              <button

                className='right-0 bg-accent text-primary font-bold'

                onClick={()=>alert(`You ordered A card worthy: ${total}`,setCard([]))}

              >Order My card</button>

* 1. **Partner**

Here nothing tricky is just rendering the partners that I fetched

const Partner = () => {

  const [partners, setPartners] = useState([]);

  useEffect(() => {

    const getPartners = async () => {

      try {

        const results = await fetch('http://localhost:3010/partner', {

          method: "GET"

        });

        const data = await results.json();

        setPartners(data);

      } catch (err) {

        alert(`Error: ${err.message}`);

      }

    };

    getPartners();

  }, []);

Too many files I was getting confused so I limited myself to only use more than one file if I have length componet; so this component

<Part key={index} path={item.path} name={item.name} />

Is also in the partner componet but as a function outside which I not being exported

const Part = ({ path = 'pathdefault', name = 'namedefault' }) => {

  return (

    <div className='flex flex-col h-[200px] w-[170px] items-center py-1'>

      <div className='h-[160px] w-[160px] bg-gray-300/80 flex items-center justify-center rounded-full'>

        <img src={path} alt="Logo" />

      </div>

      <div className='text-lg font-bold'>{name}</div>

    </div>

  );

};

1. **Business**

Here the trick was only to fetch the data and render them, also the table component s for these two each ; they were in the same file , Business.jsx

* 1. **Orders**

Using Table component ; in here I fetched all orders

  useEffect(() => {

    const getOrders = async () => {

      try {

        const result = await fetch('http://localhost:3010/order', {

          method: "GET",

        });

        const data = await result.json();

And flattened the info; making it into an array

   const refined = data.map(el => [

          el.ordered?.phone || '',

          el.ordered?.location || '',

          el.orderCost?.ordeCart?.name || '',

          el.orderCost?.amount || '',

          el.payment ? "Paid" : "Unpaid",

          new Date(el.createdAt).toLocaleString() || ''

        ]);

Then I set the rows and filterrows || rows the orignal data fetched, we need some other value that will change as we filter; if we directly change the rows, we will lose the information and require to re-fetch;

        setRows(refined);

        setFilteredRows(refined);

      } catch (err) {

At first we render everything. Through filteredRows

Then as the fecthed data changes or the search text changes we change the filteredRows

 useEffect(() => {

    const text = searchText.trim().toLowerCase();

    const matches = rows.filter((row) =>

      row.some((cell) =>

        String(cell).toLowerCase().includes(text)

      )

    );

    setFilteredRows(matches);

    setActiveTab(0);

  }, [searchText, rows]);

Now as we have limit of rows per page we prepair chunks

  const rowChunks = [];

  for (let i = 0; i < filteredRows.length; i += chunkSize) {

    rowChunks.push(filteredRows.slice(i, i + chunkSize));

  }

rowChunks is an arrays of rows. Then on tab 0 we render rowchunk[0] .. like that.

The pagination will show if the rowChunks has more that one item (chunk)

{rowChunks.length > 1 && (

        <div className="flex mb-4 gap-2 w-[70vw]">

          {rowChunks.map((\_, i) => (

            <button

              key={i}

* 1. **Products**

Here the table rendering is the same;

  useEffect(()=>{

    const getProducts = async()=>{

      const result = await fetch(` http://localhost:3010/products`,{

        method: "GET"

      });

      const data = await result.json();

      const flatten = data?.map(row=>(

        [

          row.id,

          row.path,

          row.name,

          row.measure

        ]

      ));

      setRows(flatten);

    }

    getProducts();

  },[]);

only difference is that there is a conditional rendering whenever user want to add a product

  const Addproduct = () =>{

    setVisible(true);

  }

 {/\* add a partner form \*/}

      {visible && (

        <div className='h-[470px] w-[400px] bg-blue-200/90 flex  flex-col items-center border-4 border-white rounded-xl absolute top-2 left-5 mt-3'>

And a relative closing action

  const Addproductclose = () =>{

    setVisible(false);

  }

 <button

          className='absolute right-0 top-0 font-bold text-sm text-white bg-red-500 '

          onClick={()=>Addproductclose()}

          >

          x

        </button>

So once it is visible there is a form , and I prepair the states to save the info.

  const [productName,setProductName] = useState('');

  const [productFile,setProductFile] = useState(null);

  const [productMeaure,setProductMeasure] = useState('');

Product file is null as we are expecting a file[0]. a single file.

As we are to post and it envolve files we will use multer which is acting in the midleware , so we sent our body as FormData

const formData = new FormData();

    formData.append("name",productName);

    formData.append("profile\_path",productFile);

    formData.append("measure",productMeaure);

    try {

      const res = await fetch(" http://localhost:3010/register/product",{/// name,measure,profile\_path

        method:"POST",

        body:formData,

      });

Notably: ensuring the name we are calling the file and other fields is the same as what the backend is expecting. Cause it may raise a bug, a serious one, as you think that the logic is correct but it is actually a typo error , in the backend ‘s sense.

Once what you are setting is done we set back the fields to empty

 if(res.ok){

        alert(data.msg || "partner registerd");

        setProductName('');

        setProductMeasure('');

        setProductFile(null);

      }else{

        alert(data.Error || "upload failed")

      }

Then one can close the view if he/she likes.

At last I businness component itself there is a logout button

const Business = () => {

  const navigate = useNavigate();

  return (

    <div className='flex flex-col justify-center items-center gap-10 p-10 relative'>

      <button className='absolute top-0 right-0 m-5 h-[40px] w-[70px] flex items-center justify-center text-white bg-primary'

      onClick={()=>navigate('/')}

      > logout</button>

1. **Admin**

As in business the components itself has nothing much the tables to render thes bellow section are the one with a lot of logic behind.

Also there is a log out button

const Admin = () => {

  const navigate =useNavigate();

  return (

    <div className='relative flex flex-col bg-blue-300/40 justify-center items-center gap-10 p-10'>

      <button className='absolute top-0 right-0 mx-12 h-[40px] w-[90px] flex items-center justify-center'

      onClick={()=>navigate('/')}

      >LogOut</button>

* 1. **All users**

In Table component we fetch users and render them, in chuncks as usual

  useEffect(() => {

  const getUsers = async () => {

    try {

      const res = await fetch('http://localhost:3010/user');

      const data = await res.json();

      setUsers(data);

    } catch (err) {

      console.error(err.message);

    }

  };

The other logic is the same as in business

We have also the deletin function, so in every row we have a delete button once we clik on delete

{rowChunks[activeTab]?.map((row, rowIndex) => (

              <tr key={rowIndex} className="hover:bg-gray-50">

                {Array.isArray(row)

                  ? row.map((cell, i) => (

                      <td key={i} className="px-4 py-2 border-b border-blue-200">

                        {cell}

                      </td>

                    ))

                  : headers.map((key, i) => (

                      <td key={i} className="px-4 py-2 border-b border-gray-200">

                        {row[key]}

                      </td>

                    ))}

                <td className="px-4 py-2 border-b border-gray-200 text-center">

                  <button

                    onClick={() =>HandleDelete(row.id)}

                    className="text-red-600 hover:underline"

                  >

                    Delete

We fetch on the end point using the user’s id and delete him/her

  const HandleDelete = async (id) => {

    try {

      const res = await fetch(`http://localhost:3010/user/delete/${id}`, {

        method: "DELETE",

      });

* 1. **Partners**

Tablepartner component also we do two things; we fetch all and display them

  // Fetch partners

  const getPartners = async () => {

    try {

      const res = await fetch("http://localhost:3010/partner");

      const data = await res.json();

      setPartners(data);

    } catch (err) {

      console.error(err.message);

    }

  };

And in chucnks we render them

  const rowChunks = [];

  for (let i = 0; i < filteredRows.length; i += chunkSize) {

    rowChunks.push(filteredRows.slice(i, i + chunkSize));

  }

        {rowChunks[activeTab]?.map((partner, rowIndex) => (

              <tr key={rowIndex} className="hover:bg-gray-50">

                <td className="px-4 py-2 border-b border-blue-200">

                  <img

                    src={partner.path}

                    alt={partner.name}

We also have a form to add a partner which is conditional rendered; once the user clicks on add a partner button

<button

        className="absolute right-0 p-4 m-5 font-bold text-lg text-white bg-primary"

        onClick={() => setVisible(true)}

Same logic as we are uploading an image, we use FormData to make a body, and on success we rest the fields to empty.

  const formData = new FormData();

    formData.append("name", partnerName);

    formData.append("profile\_path", partnerFile);

    try {

      const res = await fetch("http://localhost:3010/register/partner", {

        method: "POST",

        body: formData,

      });

      const data = await res.json();

      if (res.ok) {

        alert(data.msg || "Partner registered");

        setPartnerName("");

        setPartnerFile(null);

        setVisible(false);

        getPartners(); // Refresh table

      } else {

* 1. **History**

Tablehistory component, the one with a very complex dat fetching, as we joined nearly every table we had.

So we have the search functionality and chunks rendering like all the other , the trick was to access these extrem nested json data, I learnt to us ? for accessing a value if only not undefined as thing kept being undefined for a lonf time.

    const getHistory = async () =>{

      try {

        const res = await fetch("http://localhost:3010/history",{

          method:"GET",

        });

        const data = await res.json();

        const flat = data?.map(row=>(

          [

          row.payment?"true":"false",

          row.bank,

          row.bank\_card,

          row.ordered?.ownerName,

          row.ordered?.ownerLocation,

          row.ordered?.ownerPhone,

          row.ordered?.ownerProfile,

          row.orderCost?.amount,

          row.orderCost?.ordeCart?.productImage,

          row.orderCost?.ordeCart?.Product,

          row.orderCost?.ordeCart?.storing?.[0]?.unitPrice,

          row.orderCost?.ordeCart?.storing?.[0]?.shopOwner?.shopName,

          row.orderCost?.ordeCart?.storing?.[0]?.shopOwner?.shopLocation,

          row.orderCost?.ordeCart?.storing?.[0]?.shopOwner?.shopPhone,

          row.orderCost?.ordeCart?.storing?.[0]?.shopOwner?.shopProfile,

          row.orderDelivery?.deliveryStatus || "NOT YET",

          row.orderDelivery?.totalCost || "NOT YET",

          row.orderDelivery?.ordersDestination?.deliveryName || "NOT YET",

          row.orderDelivery?.ordersDestination?.deliveryLocation || "NOT YET",

          row.orderDelivery?.ordersDestination?.deliveryPhone || "NOT YET",

          row.orderDelivery?.ordersDestination?.deliveryProfile || "NOT YET"

        ]

        ));

And also render them, as usual.

### Challenges & solution

1. **Late Change of Backend tools:**

At first I was not using vanila javascript and implemented Http protocals from scratch. It was tiring but at last it was working, unfortunately my set-up was remarkably vunerabel and my supervisor saw it as a worst start and a bad practice, so I whiched to using Express, Sequelize for mu back end. Which slowed done the project.

**Solution:**  I refactored the whole idea simplified it and relied mostly on documentation other than youtube videos as they were taking me very long time to even realise that I am repeating some parts more often.

1. **Very new to the full stack idea**

I had to learn how to use the frame works, not total from srcatch but to piece things together so that they work. Was new to me; which learning the tools itself was not going to meet the 30 days deadline.

**Solution:** limiting myself to 4 days I followed a youtube videos, not to enterly get the process just to understand the structure, as I found better structure less the codes are luckily to break each other, and I learnt a concept as I reached.

1. **Changing from normal Sql queries to Sequelize:**

Sequelize even though it comes with a more secure ways to access the database me , I used to be flexible with normal queries but for the knowledge sake ; I had to learn its synthax; where by I thought the database should not take my time, ended being also a thing to tackle.

**Solution:** As I didn’t know any video to learn it and nail it in an instance, I used to write an sql of what I want and ask AI to transform it, then I learn from it; later I write it on my own then ask AI the correct me , and went on like that. Till where it started t make sense.

1. **Sequelize ‘s mvc approach**

This tool not only terified me with the query sythax, its version control part, is another thing; said it is sequeize-cli which uses command prompt to create a model , as well as the querying thing , I learnt it with help of AI; and took not of it, until I found the patern , them it started making sense , abit. Now thinking I am done with structuring my database tables(model) even creating rondom entries required to seed them, ANOTHER sequelize-cli thing. Which had its issues also.

**Solution:** I relied of AI’s guidance until I learnt how to do it and not break everything else; I was not thing of reading documentation as I though either way I was not gone learn exactly what I need to keep my project going, youtube videos was not even near of a choice. Luckily I pulled it up.

1. **Sequelize ‘ s Seeder**

As I said prior , even making some random enteries into the database , in sequlize one uses seeder, which is a command line based tasks. Now the issues here even AI couldn’t provide a proper walk around of this; if I created a first seeder, it was working, but creating the second, and try to seed it, the first already seeded file, was throwing an error. I first thought of deleting them, but from my last experience with the models, I made a mistake and runed :undo which reversed to my state where things were working; so deleting also seeder I didn’t think it would be wise.

**Solution:** I created another file near by with another name other than seeder as command line for seeding , generated files in a folder called seeder. Then I seeded them one by one.

1. **Later changes into the database not being recognized**

Whatever I was changing in the model files it was not being reflected so, sequelize recognized it; or to manually change things in the mysql workbench.

**Solution:** the issue was not syncing the alteration and allow it to be done whenever the server run.

const startServer = async () => {

  try {

    await sequelize.authenticate();

    console.log(` Database connected successfully`);

    await sequelize.sync({ alter: true });

    console.log(` All models synchronized with DB`);

Sync({alter: true}) ; made it possible.

1. **Frontend is not communicating with backend // cors**

After nearly 3weeks struggling with the backend ; now I was heading to the frontend everything, trying to make the rest ,in a week or so; first structered my frontend rendering only colored divs. Now when I tried to fetch data , hitting the backend’s end points nothing was showing.

**Solution:**  I remembered that , during developing the http protocals from scratch they used to say things of cors, and in the console , every where it said violation of cors things. Then I watched a youtube videos, lucky enough what they told me to write in my serve.js solved it.

const app = express();

app.use(cors());

I just had to npm install the cors then use them that’s all

1. **Database can not save images, it was size limiting // multer**

Now that I was to use images, I never thought about where the images do come from, really. i was thinking of putting them in public/images folder of front end and manually name them and keep their name in the backend. But now I wondered, I don’t really know howmany products will my app have, nor the user, any new entry in the database, equals new images. So I needed an automation of this thing.

I thought of applauding files; which was a good approach, whn I tried mysql, its data type for file which is BLOB , keep changing files into binary, which slows down the process,and the database being slow, it can be notably slow, talking into minutes.

**Solution:** Maybe my AI prompt was not well formulated, but when I asked it told m we can not save file uploaded by user into a file on local divice and access its path later, to store it in the data base. But when I looked on youtube I found that a tool called multer can actually do that, but working as a midleware , in between the routes, working on http protocals. i cofigured it to first insert data without the path, and then retrieve the id of the entry and use it to formulate the name, and save it in the public folder of my frontend. It works.

import multer from 'multer';

import fs from 'fs'

import path from 'path';

You prepair a memor storage, in which we upload into

const storage = multer.memoryStorage();

const upload = multer({storage});

Then in between the end point and call back function we put there the upload function with a name of what it is expecting

pp.post('/register/product', upload.single('profile\_path'), async (req, res) => {

We save the data but living the path to the image with a garbage value.

 const result = await Product.create({ name,measure, profile\_path: '' });

      const id = result.id;

The we retrieve the id on out entry to modfy the name of the file

 const result = await Product.create({ name,measure, profile\_path: '' });

      const id = result.id;

Thanks to path and fs impotrs we use direct path and guide our saving , to save in the front end’s public folder.

const imageDir = path.join(\_\_dirname, '..', 'frontend', 'public', 'images');

          if (!fs.existsSync(imageDir)) {

            fs.mkdirSync(imageDir, { recursive: true });

      }

If the file does not exist we create it, and the files in it. After we write out file in that place.

const fullPath = path.join(imageDir, Filename);

      fs.writeFileSync(fullPath, req.file.buffer);

Then we properly form the path of image to be saved in the database; here we mind that it will be access from the frontend, so the path is in a way that fevors the frontend.

const imagePath = `/images/${Filename}`;

      const [updated] = await Product.update({ profile\_path: imagePath }, { where: { id } });

Then we check if the path is updated in the databse.

### Testing & Validation

For the backend, most of it it was postman for the, routes functionality, the for interaction with the backend, I was manually checking in mysql workbench and see if the changes are made in the database tables. So Mostly it was Unity testing, no fancy tools.

Frontend, the browser itself tests you, so the errors were to display in the console, as it was in the backend as well, or the console on=f the browser; mostly when the site blanked out , suddenly, the issues we listed in the browser console, not in the vscode console.And most importantly for async functions, when dealing with callbacks I used try and catch extensively, alerting the errors which speeded up my work.

In short; I used Unity testing, thanks to try and catch; the only specific tool I used was postman in the backend and chrome web browser in the frontend.

If I new how to fix it I did so, but if I found that I am breaking more I handed it to the AI to tell me what is wrong, I located the exact place, as indicated in the error; the copy the error, And I learnt from it, that way.

### Conclusion

At first I truly admit , I didn’t know what I was signing for , whatever they told me I was naively saying yes I can but truly me too , I was testing myself; how far can I go. The first thing I learnt is to read the documentation and retrieve what is important. Cause I only relied on youtube, and AI mostly and reading the documentation; were never my first go to approach.

Secondary, I learnt how important and very in-handy separation of concern is, in everything. At first I didn’t understand what component is, not like what is it , entirely , but what can we call a component, and what can we say is made of components. This I learnt it on the way, whenever I found that ideas are too much on a feature, I immediately new, that I am dealing with multiple component in one component. And separating things wisely did help me in the long run.

I am immensely grateful for the opportunity of working on full stack as even in the future I might not get this chance again, which I don’t wish it to be; this tough me every bit of everything; which now I am more than confident to assist in any form of project as I know what to expect. And atleast now I have a language that I am confident to build my final project in **JavaScript.**

### Future Work

As of now it is barely 70 percent of what my idea of the website was initially like; and to what I think of adding late, if it came to be.

* Proper alerts using toasters
* to make loaders, as the backend traffic increase, on this kind of websites, data lagging might come about, and better loader can ease then user’s impatience
* For now I did not dig deep into how the delivery man come about, I assume the shop will assign a delivery man from the trusted source, but this is invisible to the customer as of , now, I think of adding this transparency later on as it is very crucial.
* I did not get time to properly use the framer-motion library, hence the website has very few animations, which is boring to the user,
* most of the used features are 2 d, if applicable, I my consider in-bending, spline objects (3D object)
* If it happen to be a product to be brought to market, I may consider, twilio services.
* The admin access is too exposed I will immediately consider, using a special authentication, for it, either, a single one known password or two tier authentication

### References

React:

<https://create-react-app.dev/docs/getting-started>

Tailwind:

<https://v3.tailwindcss.com/docs/installation>

<https://nerdcave.com/tailwind-cheat-sheet>

Express:

<https://expressjs.com/>

Sequelize:

<https://sequelize.org/docs/v6/getting-started/>

Cors:

<https://expressjs.com/en/resources/middleware/cors.html>

Multer:

<https://dev.to/ramcpucoder/how-to-use-multer-with-reactjs-and-nodejs-for-file-uploads-23ie>

Framer-motion:

<https://www.geeksforgeeks.org/reactjs/framer-motion-introduction-and-installation/>

Youtube Videos:

File upload: <https://www.youtube.com/watch?v=i8yxx6V9UdM>

Framer-motion: <https://www.youtube.com/watch?v=znbCa4Rr054>

React -up and running: <https://www.youtube.com/watch?v=wOT4nKB_0mw>

Express crash course(50 mins): <https://www.youtube.com/watch?v=CnH3kAXSrmU&t=2831s>

Tailwindcss(3.4.17): <https://www.youtube.com/watch?v=qkbijl5EoHc>

Ui colors: <https://www.youtube.com/watch?v=UWwNIMHFdW4>

Figma: <https://www.youtube.com/watch?v=ezldKx-jPag>