

# Al-Najah National University



Faculty of Engineering & Information Technology  
Computer of Engineering Department

## **Graduation Project 1**

A to B

### **Groub members:**

Ethar Suwan & Entimaa Bushkar

### **Supervised by:**

Dr. Raed Al-Qadi & Dr. Suliman AbuKharmeh

Presented in partial fulfilment of the requirements for  
Bachelor degree in Computer Engineering.

Date : 1/13/2023

## Table of Contents

Figures of table:.....	3
Abstract: .....	6
Chapter 1: Introduction .....	7
Statement of the problem: .....	7
1.2 Project Objective: .....	8
1.3 Project Scope:.....	8
1.4 Project importance: .....	8
Chapter 2: Constraints and Earlier coursework. ....	10
2.1 Constraints: .....	10
2.1.1 Programming Language:.....	10
2.1.2 Remote Notification: .....	10
Chapter 3: Literature Review .....	11
Chapter 4: Methodology.....	12
4.1 Tools, Methods and Programming Languages: .....	12
4.1.1 programming language: .....	12
4.1.2 Tools: .....	12
4.1.3 Database:.....	13
4.2 System Features Implementation:.....	17
4.2.1: Google Maps usage: .....	17
4.2.2: Notification:.....	18
4.2.3: Chat: .....	21
4.3 Mobile application: .....	25
4.3.1: Authentication and Forget Password:.....	25
4.3.2: First Page: .....	33
4.3.3: Customer Side:.....	34
4.3.4: Driver Side: .....	48
4.4: Website: .....	62
4.4.1: Tools, Methods and Programming Languages: .....	62
4.4.2 programming language: .....	62
4.2.3 Tools: .....	62
4.4.4: Database:.....	63
4.2 System Features Implementation:.....	63

Chat.....	63
5.1.3 Notification System:.....	68
4.4 The Website:.....	73
Admin:.....	87
Chapter 5: CONCLUSION & RECOMMENDATION.....	90
5.1 CONCLUSION: .....	90
5.2.....	90

## Figures of table:

Figure 1 .....	13
Figure 2 .....	14
Figure 3 .....	15
Figure 4 .....	16
Figure 5 .....	16
Figure 6 .....	17
Figure 7 .....	18
Figure 8 .....	18
Figure 9 .....	19
Figure 10 .....	20
Figure 11 .....	20
Figure 12 .....	20
Figure 13 .....	21
Figure 14 .....	22
Figure 15 .....	23
Figure 16 .....	23
Figure 17 .....	24
Figure 18 .....	24
Figure 19 (Sign Up) .....	25
Figure 20 (Sign Up) .....	26
Figure 21(related to Sign up) .....	26
Figure 22(the user enter their email) .....	27
Figure 23(the user received the reset code via email) .....	27

Figure 24(the user entered the new password and the reset code).....	28
Figure 25(the password successfully updated!).....	29
Figure 26 (Sign in Page) .....	30
Figure 27 (Setting tab) .....	31
Figure 28 (Setting -Account-) .....	32
Figure 29 (Setting -About-) .....	33
Figure 30(Customer or driver?).....	34
Figure 31(customer Home Page).....	35
Figure 32 (Tracking order-customer-) .....	36
Figure 33 (place order(sender Info)) .....	37
Figure 34 (place order (recipient info)) .....	38
Figure 35 (place order (Package info)).....	39
Figure 36 (Place order(Date picker)) .....	40
Figure 37(Place order (receipt)) .....	41
Figure 38 (Success Modal) .....	42
Figure 39 (current user orders) .....	43
Figure 40 (Order Details) .....	44
Figure 41 (order details but when its shipped) .....	45
Figure 42 (order details but when its shipped & delivered) .....	46
Figure 43 (Map- Tracking order) .....	47
Figure 44 (Chat between the driver and the customer) .....	48
Figure 45 (driver Home page) .....	49
Figure 46 (New Delivery req notification).....	50
Figure 47 (the modal with the delivery req details) .....	51
Figure 48 (Customer side when a driver accepts the order) .....	52
Figure 49 (Drivers Packages) .....	53
Figure 50 (order details shipped) .....	54
Figure 51 (order details deliverd).....	56
Figure 52 (order details shipped and delivered) .....	57
Figure 53 (Driver Map) .....	59
Figure 54(Driver Map) .....	60
Figure 55 (driver Chat).....	61
Figure 56 .....	63
Figure 57 .....	65
Figure 58 .....	66
Figure 59 .....	67
Figure 60 .....	67
Figure 61 notification .....	68
Figure 62 notification .....	69
Figure 63 chat.....	69
Figure 64 .....	70
Figure 65 .....	70

Figure 66 .....	71
Figure 67 .....	71
Figure 68 .....	72
Figure 69 .....	72
Figure 70 Home Page.....	73
Figure 71 About Page. ....	74
Figure 72 login.....	74
Figure 73 Admin Login Page.....	75
Figure 74 Sign-Up Page. ....	76
Figure 75 Password Recovery Page. ....	77
Figure 76 Password Recovery Page.....	78
Figure 77 Password Recovery Page.....	78
Figure 78Password Recovery Page.....	79
Figure 79 Password Recovery Page.....	79
Figure 80 .....	80
Figure 81Tracking Page.....	80
Figure 82 Tracking Page.....	81
Figure 83 .....	82
Figure 84 Profile Page.....	82
Figure 85 .....	82
Figure 86 .....	83
Figure 87 Request Page. ....	83
Figure 88Request Page. ....	84
Figure 89Request Page. ....	85
Figure 90Request Page. ....	85
Figure 91Request Page. ....	86
Figure 92 Request Page. ....	86
Figure 93 .....	87
Figure 94 .....	87
Figure 95Dashboard Page.....	88
Figure 96 .....	88
Figure 97 .....	89

### Abstract:

A to B is a delivery system that streamlines the delivery process for customers and drivers. The app allows customers to place orders online, track their order on a map and monitor its status, and contact the driver directly within the app. These features make it easy for customers to receive their packages in a timely manner and for drivers to manage their delivery, show them on the map, update orders status and contact customers. The system is user-friendly and easily accessible for both customers and drivers, making it convenient for businesses and individuals alike. Additionally, the system includes a website that allows customers to place orders and track them from a platform of their preference either from the application or the website. Overall, the A to B system aims to provide a user-friendly and efficient solution for delivery services, providing real-time updates for customers and drivers through push notifications, and ensuring a seamless and efficient delivery experience. The application is built with React Native and Node.js, the website for customers and administration is built with React and Node.js and we use MongoDB for both.

It is important to note that this system provides a solution for our daily lives as it caters to a wide range of delivery needs such as online shopping, restaurant, grocery store, and even personal deliveries for relatives. The increasing reliance on online shopping and food delivery services has made courier systems an essential aspect of our daily lives, and A to B aims to provide a convenient and efficient solution for these needs.

## Chapter 1: Introduction

### Statement of the problem:

The problem with traditional delivery systems is the lack of transparency and accountability. Customers often have difficulty tracking their packages and determining their exact location and placing new orders easily. A delivery app with tracking would solve this problem by providing real-time updates on the location and status of a package, giving customers peace of mind and the ability to plan accordingly. Additionally, businesses can benefit from the tracking feature as it allows them to optimize their delivery routes and improve customer satisfaction and give the customer the ability to place orders monitor them and contact the driver in the same place.

So this project focused these problems:

- The difficulty to reach delivery company and relies on social media or phone calls to place an order.
- The ability to track the order and the order status.
- The ability to contact driver on a delivery app not on phone or social media platform.
- The recipient can track the order from the website.

## 1.2 Project Objective:

This project is a delivery app and website for the customer side with tracking capabilities that improves transparency and accountability for customers and businesses. This will be achieved by providing real-time updates on the location and status of packages, as well as other features such as placing orders, contact the driver, and notifications. Additionally, the app is user-friendly and easy to navigate, and should have a secure login system to protect customer information. The goal of this project is to create a valuable tool that improves the delivery experience for both customers and drivers.

## 1.3 Project Scope:

The app will allow customers to place orders and track the status of their deliveries in real-time and contact drivers within the app. It will also have a feature that allows drivers to open and navigate to their assigned orders using a map. The website will be accessible to customers only and will provide similar functionality as the app. The app will also have a secure login system to protect customer information. This project will improve the transparency and accountability of the delivery process for both customers and drivers.

## 1.4 Project importance:

The development of a delivery app with tracking capabilities is important for several reasons:



- 1- Although delivery apps with tracking feature are not new and many exist in the market, it is not widely available in Palestine, or it may only have one or two features, which limits its accessibility and functionality for the customers and drivers in Palestine.
- 2- It improves transparency and accountability for customers by providing real-time updates on the location and status of their deliveries. This allows customers to plan accordingly and have peace of mind about the whereabouts of their package.
- 3- It enhances the customer experience by providing an easy-to-use, user-friendly app and website that allows customers to place orders, track deliveries, and access other relevant information.
- 4- It allows businesses to differentiate themselves by offering a convenient and transparent delivery service that sets them apart from competitors.
- 5- It can increase customer loyalty and retention by providing a valuable tool that improves the delivery experience.
- 6- It provides drivers with tool that allow them to receive delivery that is near them and have the ability to accept or reject the offer.
- 7- It provides the drivers with a map with suitable routes.

Overall, the development of a delivery app with tracking capabilities is an important step towards providing a better delivery experience

for customers and improving the efficiency and competitiveness of businesses in the delivery industry, especially in Palestine.

## Chapter 2: Constraints and Earlier coursework.

### 2.1 Constraints:

#### 2.1.1 Programming Language:

The team had chosen to use JavaScript, Node.js, React, and React Native to build the delivery app and website. However, the way the work was distributed among the team members resulted in isolated development efforts. Specifically, Ethar Suwan built the application using React Native and Node.js, while Entimaa Bushkar built the website using React and Node.js. This caused challenges in integrating and linking the website with the application.

#### 2.1.2 Remote Notification:

During the development process, efforts were made to implement push notification functionality using various platforms such as Firebase, Pusher, Pubnub, OneSignal and Sendbird. However, compatibility issues with the React Native version and gradle-related problems were encountered, causing delays and affecting the overall functionality of the app.

The problem was eventually resolved by using sockets and local push notifications as an alternative solution which allowed for

successful implementation of the push notification functionality in the app.

### Chapter 3: Literature Review

While Uber and DHL are well-known and successful companies in the logistics and transportation industry, they may not have the same level of recognition or availability in Palestine. DHL offers a wide range of logistics and transportation services including express delivery, freight transportation, and supply chain solutions. However, the company's global network may not be as easily accessible to individuals in Palestine. Similarly, Uber, known for its convenience, affordability, and efficient use of technology, may not have the same level of recognition or availability in the area. The company's transportation network and mobile app-based service may not be as well-suited to the unique needs and challenges of the Palestinian market. So to build a delivery system with the important features and make sure its user-friendly is important.

## Chapter 4: Methodology

### 4.1 Tools, Methods and Programming Languages:

#### 4.1.1 programming language:

Our application was built using:

- React Native Cli for the frontend.
- Node-JS for the backend
- MongoDB

#### 4.1.2 Tools:

- Visual Studio Code
- React Native 0.70.5
- Node-JS v16.17.1
- Real device to testing our project
- Postman

### 4.1.3 Database:

We use MongoDB it was reliable and suitable for our project

Customers Schema:

For the customer we store information about them like there username, phone, email, city, password and the reset Code is set null as default, we used it to temporarily store the reset code when the user forget there password.

```
const mongoose = require('mongoose');
const {Schema} = mongoose;
const CustomerSchema = new Schema(
{
  name: { type: String, required: true},
  email: {type: String, trim: true, required: true, unique: true},
  phone: { type: String, required: true, unique: true, },
  city: { type: String, trim: true, required: true, },
  password: {type: String, required: true, min: 6, max: 64},
  role: {type: String, default: "customer"},
  resetCode: {type: String},
},
{ timestamps: true }
);

module.exports = mongoose.model('Customers') || mongoose.model("Customers", CustomerSchema);
```

Figure 1

Driver Schema:

For the Driver we store information about them like there username, phone, email, city, password and the reset Code is set null as default, we used it to temporarily store the reset code when the user forget there password.

```

const mongoose = require('mongoose');
const { Schema } = mongoose;
const DriverSchema = new Schema(
{
  name: { type: String, required: true},
  driverlatitude:{type: Number,trim: true},
  driverlongitude:{type: Number,trim: true},
  email: {type: String, trim: true, required: true, unique: true,},
  phone: {type: String, required: true, unique: true, },
  city: {type: String, trim: true, required: true, },
  license:{type: String, trim: true, required: true,},
  password: {type: String, required: true, min: 6, max: 64,},
  role: {type: String,default: "driver",},
  resetCode: "",
  approved:{type:Boolean, default:false}},
{ timestamps: true }
);

module.exports = mongoose.model('Drivers')||mongoose.model("Drivers", DriverSchema);

```

*Figure 2*

## Package Schema:

For the Package we store the package id and the user who sent the order id and the name of the sender, sender Location, recipient Location and the driver location and their id add when the driver accept the delivery.

```

const mongoose=require('mongoose');

const PackageSchema= new mongoose.Schema({
  packageid:{type: String,unique: true,required: true},
  userid:{type:mongoose.Schema.Types.ObjectId , ref:'customers',required: true },
  driver:{type:mongoose.Schema.Types.ObjectId,ref:'Drivers'},
  recipient:{ type: String, trim: true},
  sender:{ type: String, trim: true},
  recipientEmail: {type : String,trim: true},
  recipientPhone:{type: String, trim: true },
  recipientlatitude:{type: String, trim: true },
  recipientlongitude:{type: String, trim: true },
  senderlatitude:{type: String,trim: true},
  senderlongitude:{type: String,trim: true},
  driverlatitude:{type: Number},
  driverlongitude:{type: Number},
  deliveryDate: {type: Date,trim: true},
  arrived:{type:Boolean , default:"false"},
  shipped:{type:Boolean , default:"false"},
  description:{type: String },
  isVisible :{type:Boolean, default:true},
  customer_address:{type:String, unique:false},
  to_address:{type:String, unique:false},
  createdAt:{type:Date},
},
  { timestamps: true })
const Packages=mongoose.model('packages',PackageSchema,'packages');
module.exports=Packages;

```

*Figure 3*

### Message Schema:

For the message we store the conversation id and the sender id and the content of the message.

```

const mongoose = require("mongoose");

const MessageSchema = new mongoose.Schema(
  {
    conversationId: {
      type: String,
    },
    senderId: {
      type: String,
    },
    text: {
      type: String,
    },
  },
  { timestamps: true }
);

const message=mongoose.model('message',MessageSchema,'message');
module.exports=message;

```

*Figure 4*

## Conversations Schema:

For the conversation we store the id of the conversation and the ids of both the customer and the driver who are members of the conversation.

```

const mongoose = require("mongoose");

const ConversationSchema = new mongoose.Schema(
  {
    members: {
      type: Array,
    },
  },
  { timestamps: true }
);

const Conversation=mongoose.model('conversation',ConversationSchema,'conversation');
module.exports=Conversation;

```

*Figure 5*

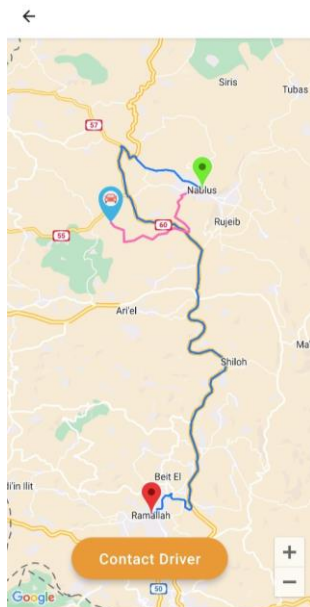


## 4.2 System Features Implementation:

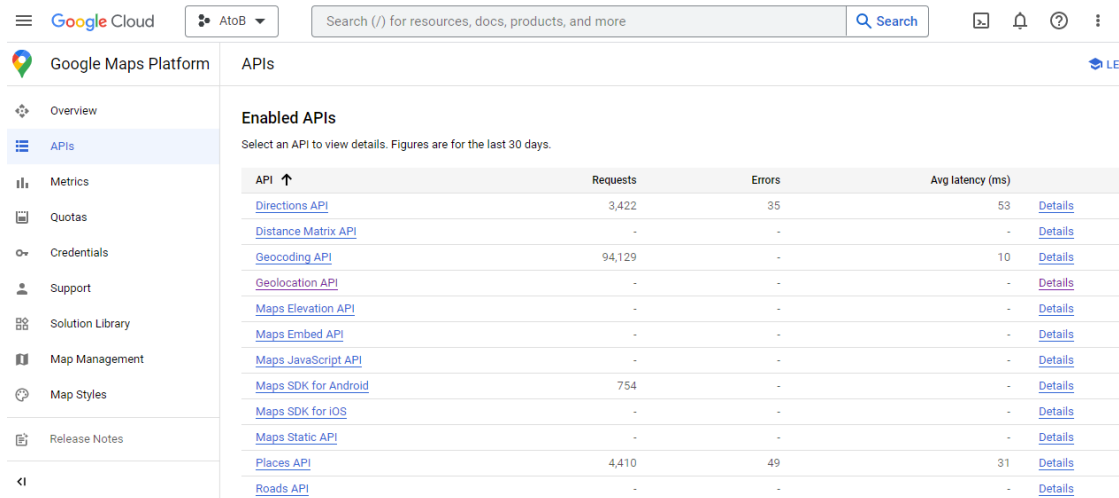
### 4.2.1: Google Maps usage:

In the application we used Google Maps and use a Google Map api key to determine the current location of both the customer and the driver, and we use it in the react-native-google-place-autocomplete to give the customer the ability to change their location and specify the recipient location.

We use it as well on the maps that we used to track the order status and track the driver location on the customer side and we use it in the driver side so they can use it to show the orders sender and recipient location.



*Figure 6*



API	Requests	Errors	Avg latency (ms)
<a href="#">Directions API</a>	3,422	35	53
<a href="#">Distance Matrix API</a>	-	-	-
<a href="#">Geocoding API</a>	94,129	-	10
<a href="#">Geolocation API</a>	-	-	-
<a href="#">Maps Elevation API</a>	-	-	-
<a href="#">Maps Embed API</a>	-	-	-
<a href="#">Maps JavaScript API</a>	-	-	-
<a href="#">Maps SDK for Android</a>	754	-	-
<a href="#">Maps SDK for iOS</a>	-	-	-
<a href="#">Maps Static API</a>	-	-	-
<a href="#">Places API</a>	4,410	49	31
<a href="#">Roads API</a>	-	-	-

Figure 7

#### 4.2.2: Notification:

When a customer places an order a delivery request is sent to the nearby drivers, a socket.io client is used to build a reliable and real-time notification and then when the driver receive the notification a local push notification is fired an notify the driver that there is a new delivery request.

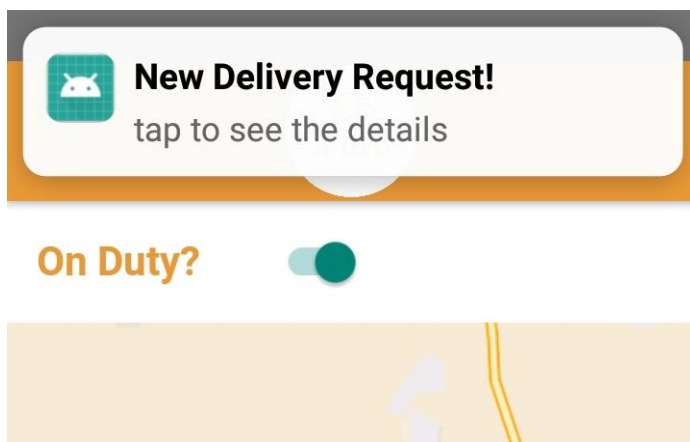


Figure 8

The code from the socket server:

This is used to connect both the driver and the customer to the socket and depending on their role they are stored on drivers or customers Array.

```
const io = require("socket.io")(8800, { cors: { origin: "*" } });

let Drivers = [];
let Customers = [];
let PackageDetails = {};

io.on("connection", (socket) => {
  // Add customer to the Customers array
  socket.on("addCustomer", (newUserId) => {
    if (!Customers.some((user) => user.userId === newUserId)) {
      Customers.push({ userId: newUserId, socketId: socket.id });
      socket.join("Customers");
      console.log("New Customer Connected", Customers);
      io.emit("get-Customers", Customers);
    }
  });

  // Add driver to the Drivers array
  // Add driver to the Drivers array
  socket.on("addDriver", (newUserId) => {
    if (newUserId &&!Drivers.some((user) => user.userId === newUserId)) {
      Drivers.push({ userId: newUserId, socketId: socket.id });
      socket.join("drivers");
      console.log("New Driver Connected", Drivers);
      io.emit("get-drivers", Drivers);
    }
  });
});
```

*Figure 9*

Here is how the delivery request is handled from the server side:

```
socket.on("send-deliveryReq", (data) => {
  PackageDetails[data.packageid] = {
    userId: data.userId,
    packageid: data.packageid,
    // other package details
  };
  console.log("New Package Request", data);
  io.to("drivers").emit("recieve-deliveryReq", data);
  socket.join(data.packageid);
});
```

Figure 10

Here is the socket that handle the acceptance of the request:

```
socket.on("acceptDelivery", (data) => {
  console.log("accepted", data);

  // find the driver who accepted the delivery request
  let driver = Drivers.find((user) => user.socketId === socket.id);
  driver.location = data.driver_location;
  console.log("driver", driver.userId)
  console.log('the driver after stor it', driver.location)
  // find the customer who made the delivery request
  let package = PackageDetails[data.packageid];
  if (package) {
    let customer = Customers.find((user) => user.userId === package.userId);
    console.log("ksjflskjdfklsdjflksd")
    if (customer) {
      console.log("customer infos", customer)
      console.log("package.packageid", package.packageid)

      // send the driver's location, name, and ID to the customer
      io.to(customer.socketId).emit("driverAccepted", data);
    }
  }
});
```

Figure 11

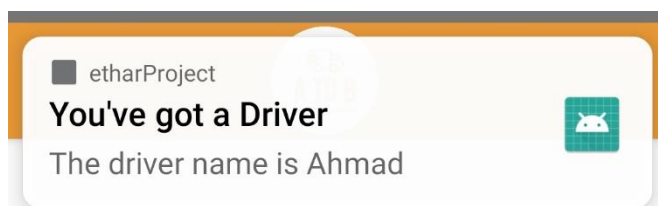


Figure 12

#### 4.2.3: Chat:

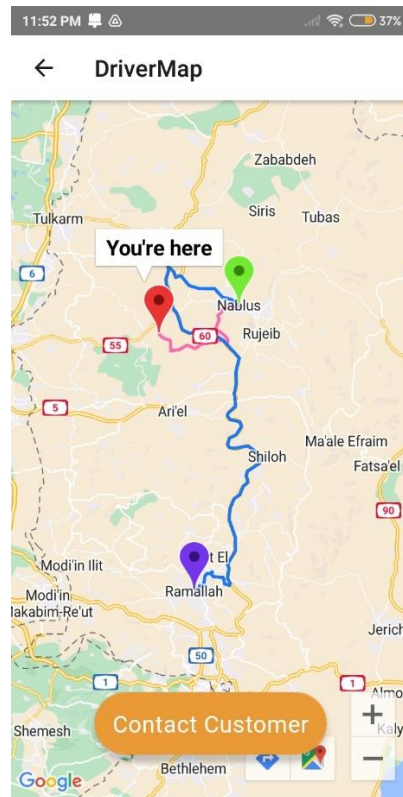
The chat on the application is implemented between the driver and the customer.

When the driver accepts the delivery a new conversation is created using the driver id and the sender id.

```
exports.createChat=async(req,res)=>{
  const newChat= new Conversation({
    members:[req.body.senderId,req.body.receiverId],
  })
  try {
    const result=await newChat.save();
    req.status(200).json(result);
  } catch (error) {
    res.status(500).json(error);
  }
}
```

*Figure 13*

And then when the driver or the customer press the contact button this code is used to find that conversation and take the id and get the messages and display them.



*Figure 14*



Figure 15

```
exports.findChat=async (req,res)=>{
  const firstId = req.query.firstId;
  const secondId = req.query.secondId;

  try {
    const chat=await Conversation.findOne({
      members:{$all: [firstId,secondId]}
    })

    res.status(200).json(chat);
  } catch (error) {
    res.status(500).json(error);
  }
}
```

Figure 16

```

exports.getMessages=async(req,res)=>{
  console.log("here!!!")
  const conversationId = req.query.conversationId;
  if(conversationId){
    try {
      const result=await Message.find({conversationId: conversationId});
      res.status(200).json(result);
    } catch (error) {
      res.status(500).json(error);
    }
  }else{
    res.status(404).json({error: "conversationId not found"});
  }
}

```

*Figure 17*

And here is the code we used to send a message:

```

exports.addMessage=async(req,res)=>{
  const conversationId=req.body.conversationId;
  const {senderId,text}=req.body;
  console.log("Conversation ID: ", conversationId);
  const message=new Message({
    conversationId,
    senderId,
    text
  })
  try{
    const result=await message.save();
    res.status(200).json(result);
  }
  catch(error){
    res.status(500).json(error);
  }
}

```

*Figure 18*



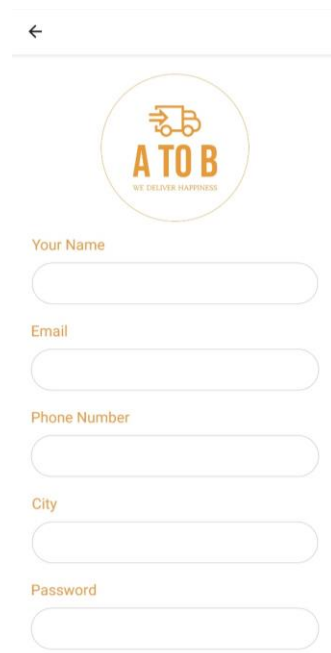
## 4.3 Mobile application:

### 4.3.1: Authentication and Forget Password:

The following pages (sign up and sign in and forget password, setting Tab) are the same for the customer side and the driver side (the difference on handling them on the backend)

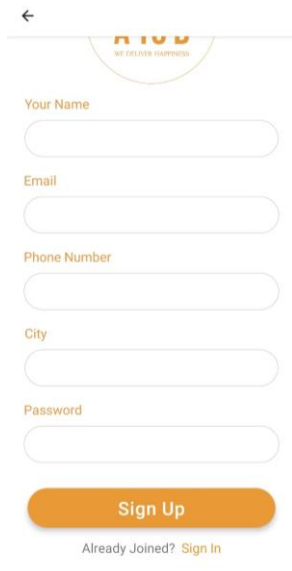
So I'll put them once on this section:

#### 4.3.1.1: Sign Up Page:



The image shows a mobile application sign-up page. At the top, there is a back arrow icon. Below it is the ATOB logo, which consists of a truck icon inside a circle, with the text 'ATO B' and 'WE DELIVER HAPPINESS' underneath. The form contains five input fields, each with a label above it: 'Your Name', 'Email', 'Phone Number', 'City', and 'Password'. The input fields are rounded rectangles with a light gray border.

*Figure 19 (Sign Up)*



←

**ATO B**  
WE DELIVER HAPPINESS

Your Name

Email

Phone Number

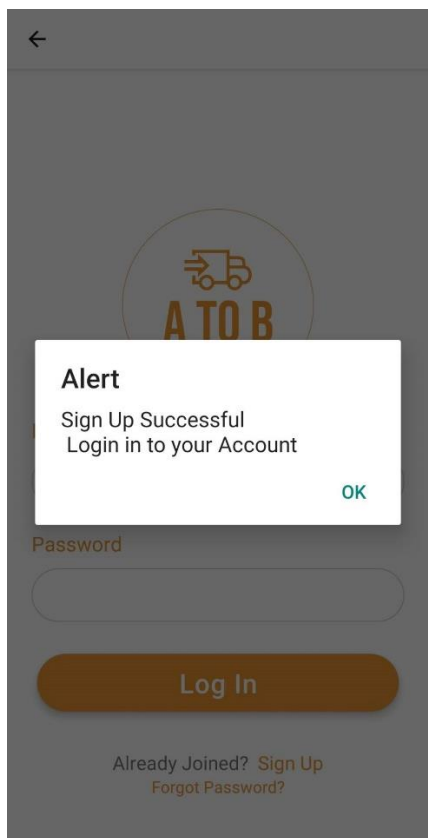
City

Password

**Sign Up**

Already Joined? [Sign In](#)

*Figure 20 (Sign Up)*



←

**Alert**  
Sign Up Successful  
Login in to your Account  
[OK](#)

**ATO B**

Password

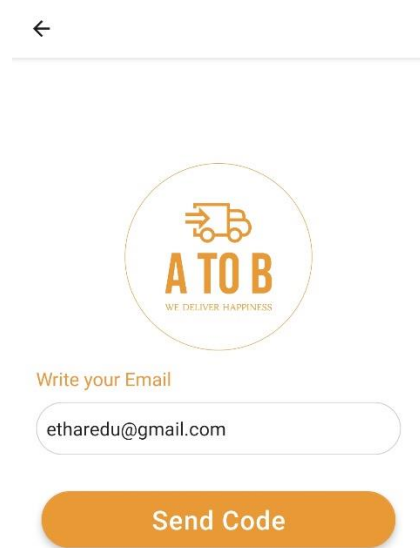
**Log In**

Already Joined? [Sign Up](#)  
[Forgot Password?](#)

*Figure 21(related to Sign up)*

#### 4.3.1.2 Forgot Password:

In this page if the user forgets their password they can simply enter their email and a reset code will be received on their email the can use the reset code on the navigated page and add the new password.



A screenshot of a web form for password reset. At the top left is a back arrow. Below it is a circular logo with a truck icon and the text 'A TO B' and 'WE DELIVER HAPPINESS'. The form has a label 'Write your Email' above a text input field containing 'etharedu@gmail.com'. Below the input field is an orange button labeled 'Send Code'.

Figure 22(the user enter their email)

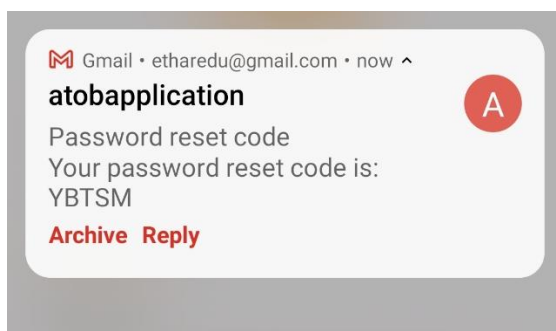


Figure 23(the user received the reset code via email)

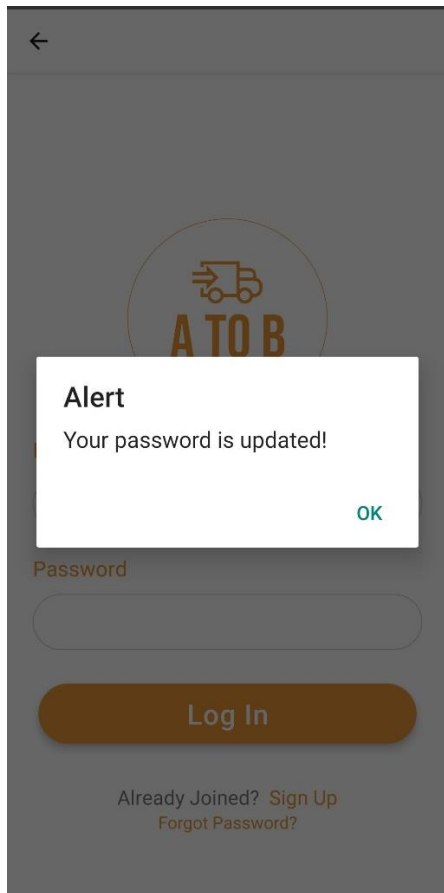


New Password

Reset Code

Update Password

*Figure 24(the user entered the new password and the reset code)*



*Figure 25(the password successfully updated!)*

#### *4.3.1.3 Sign in:*

In this page the user enters their email and password to access their account.



Email

Password

Log In

Already Joined? [Sign Up](#)  
[Forgot Password?](#)

*Figure 26 (Sign in Page)*

#### 4.3.1.4: Setting tab:

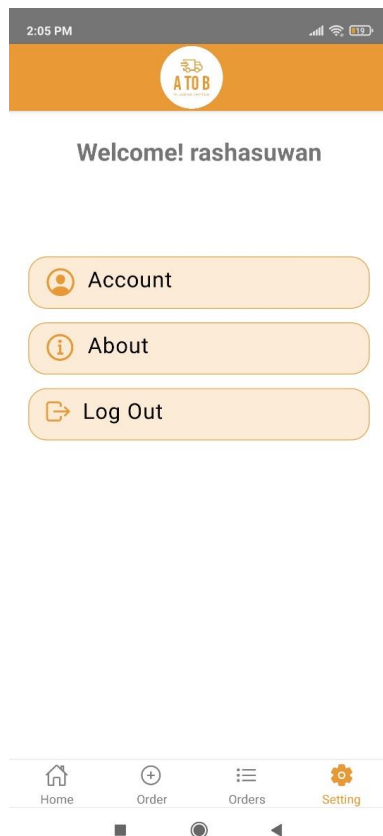
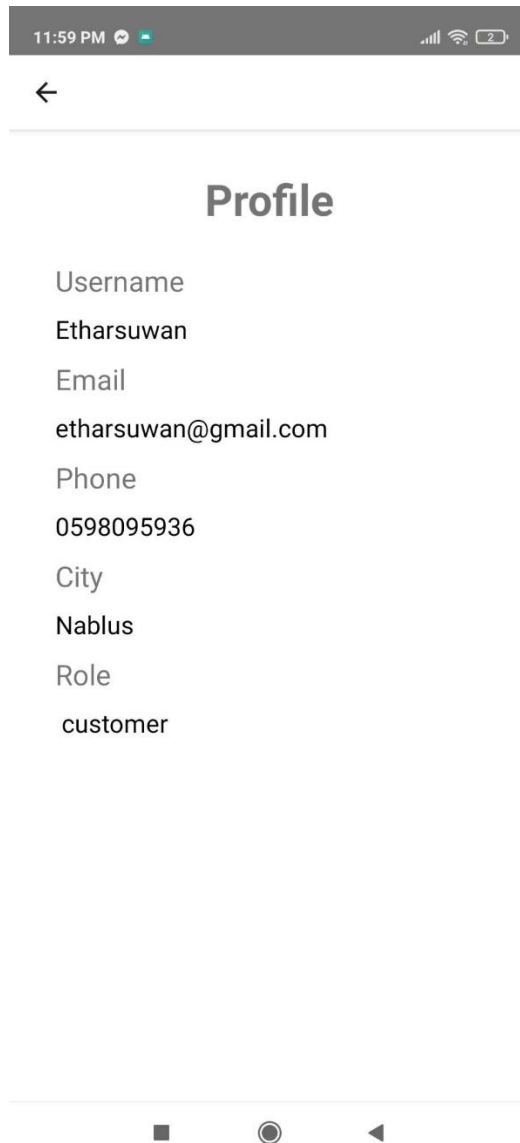


Figure 27 (Setting tab)

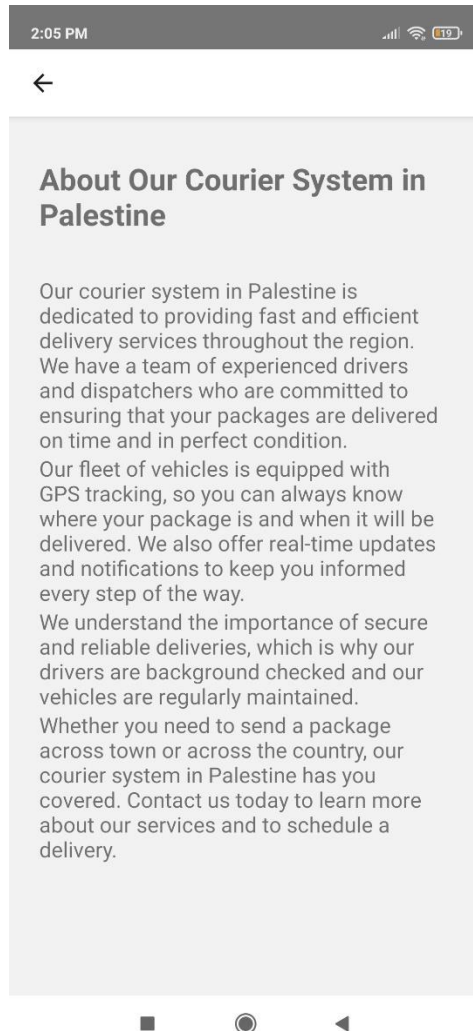
- as it is shown there is three buttons on the setting
- the first button Account is containing the information that the user filled when they signed up.



*Figure 28 (Setting -Account-)*

- the second button About is a brief about the application

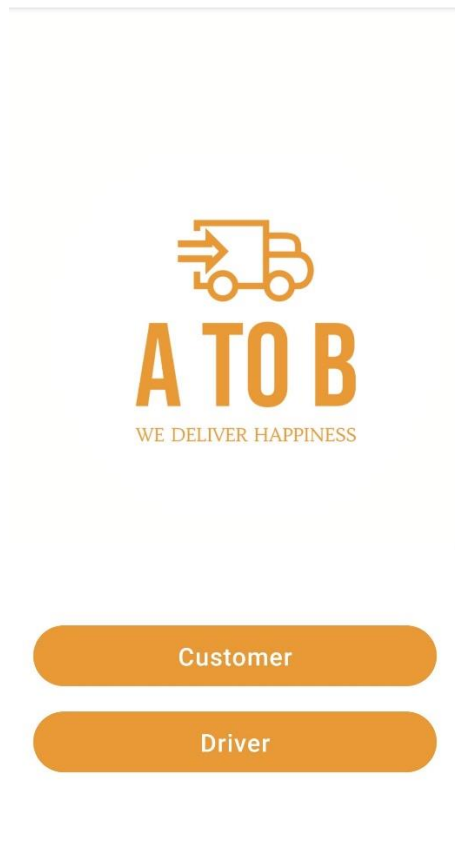




*Figure 29 (Setting -About-)*

#### 4.3.2: First Page:

This is the first page that appear when the application is opened:



*Figure 30(Customer or driver?)*

#### 4.3.3: Customer Side:

When the user press customer then the propose of their usage of the app is to send packages, track them and monitor their status.

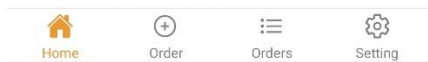
##### *4.3.3.1: Home Page:*

This page is used to search for a specific package and display it on a map to track the driver.



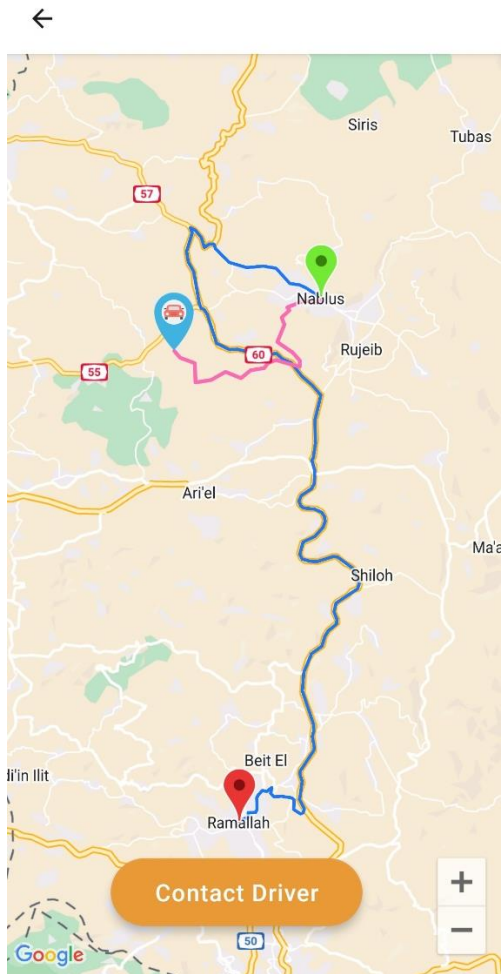
Enter Your Tracking Number

Show Delivery Details



*Figure 31(customer Home Page)*

This page navigates the user to the map with the sender, recipient and driver if there is a driver yet.




*Figure 32 (Tracking order-customer-)*

This map can be accessed on two ways from this search or from orders page I'll cover it on the following pages.

#### *4.3.3.2: Place a new Order:*

This page is used to send an order it contain multiple steps.

- **Sender Information:**  
This page is auto filled by the information that the user provided when they signed up:



### Sender Information

Ethar

etharedu@gmail.com

0598093858


8G4Q55R4+6H

Next

Home Order Orders Setting

*Figure 33 (place order(sender Info))*

- Recipient Information:  
Here the sender fill these about the recipient.



### Recipient Information

Recipient name

Recipient phone

Recipient Email


Recipient Address

Back Next

Home Order Orders Setting

*Figure 34 (place order (recipient info))*

- The next step is to fill the package information:



### Package Information


Sat Jan 21 2023


Package Description


Package cost


Back

Next

Home

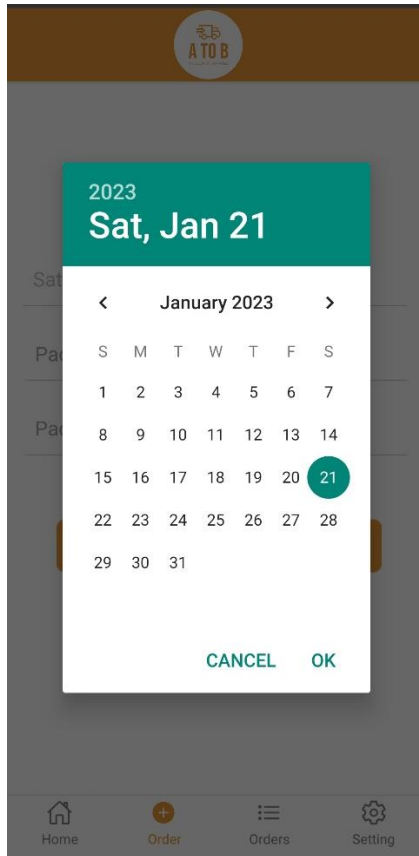
Order

Orders

Setting

*Figure 35 (place order (Package info))*

Here when the user presses the date a date picker will show up and they can change the delivery date.



*Figure 36 (Place order(Date picker))*

- Then there is the final step before confirm the order is the receipt.





**Order details**

**Recipient Email:** Rashasuwan@gmail.com

**Recipient phone:** 05980827818




**From Location:** Nablus

**To Location:** Ramallah

**Total:** 20

[Back](#) [Confirm](#)

---

 Home  **Order**  Orders  Setting

*Figure 37(Place order (receipt))*

- then when the customer confirms the order this modal while be shown.



Order sent successfully!

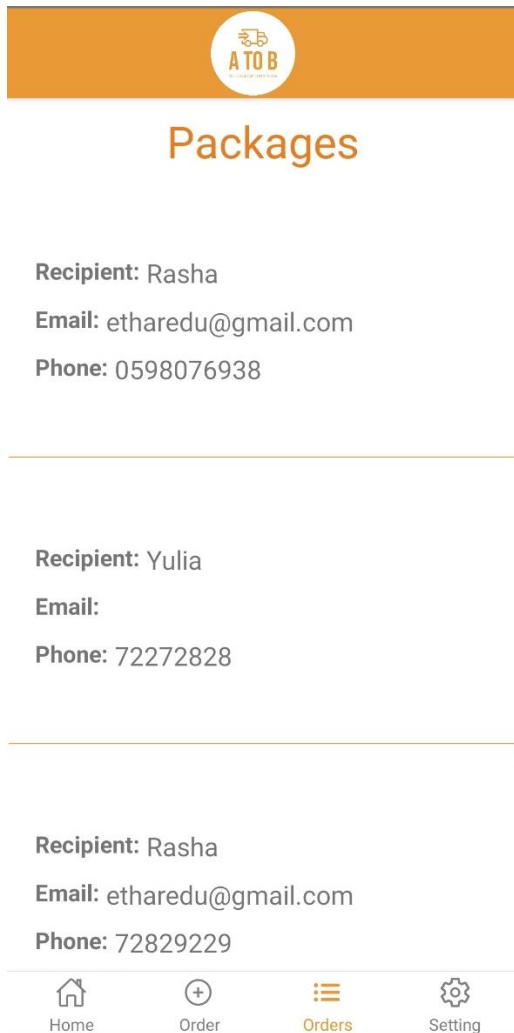
Your Tracking Number is :  
0.phurzni23b9



*Figure 38 (Success Modal)*

#### *4.3.3.3: Display orders:*

- When the user press on the orders tab the first screen will be a list with all the orders the user has placed.



*Figure 39 (current user orders)*

- When the customer press on one of the orders this screen will open with details about the order.



## Order Details

**Recipient:** Rasha

**Email:** etharsuwan@gmail.com

**Phone:** 05980865874

**Tracking Number:** 0.m1n6tezvg1s

**Shipped:** ☒

**Deliverd:** ☒

Track The Order!

*Figure 40 (Order Details)*

- Here the shipped and delivered are grayed that's mean the order is not picked up nor dropped off. In the next two screenshots the order details when its picked up and then when its delivered.



## Order Details

**Recipient:** Rasha

**Email:** etharsuwan@gmail.com

**Phone:** 05980865874

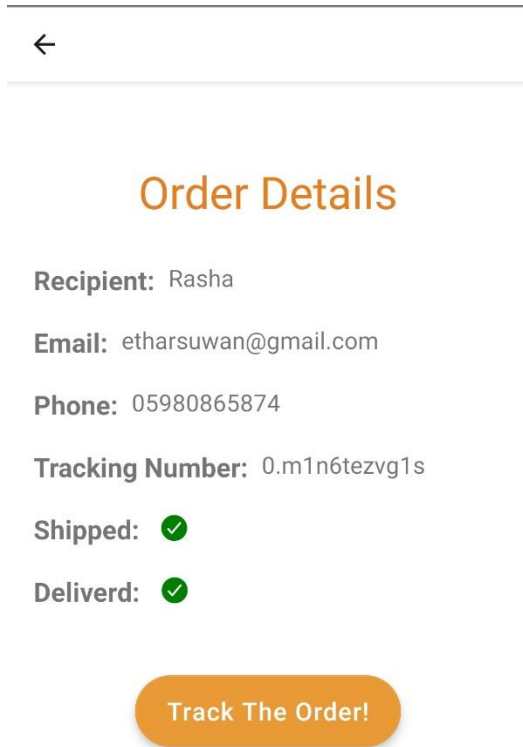
**Tracking Number:** 0.m1n6tezvg1s

**Shipped:** 

**Deliverd:** 

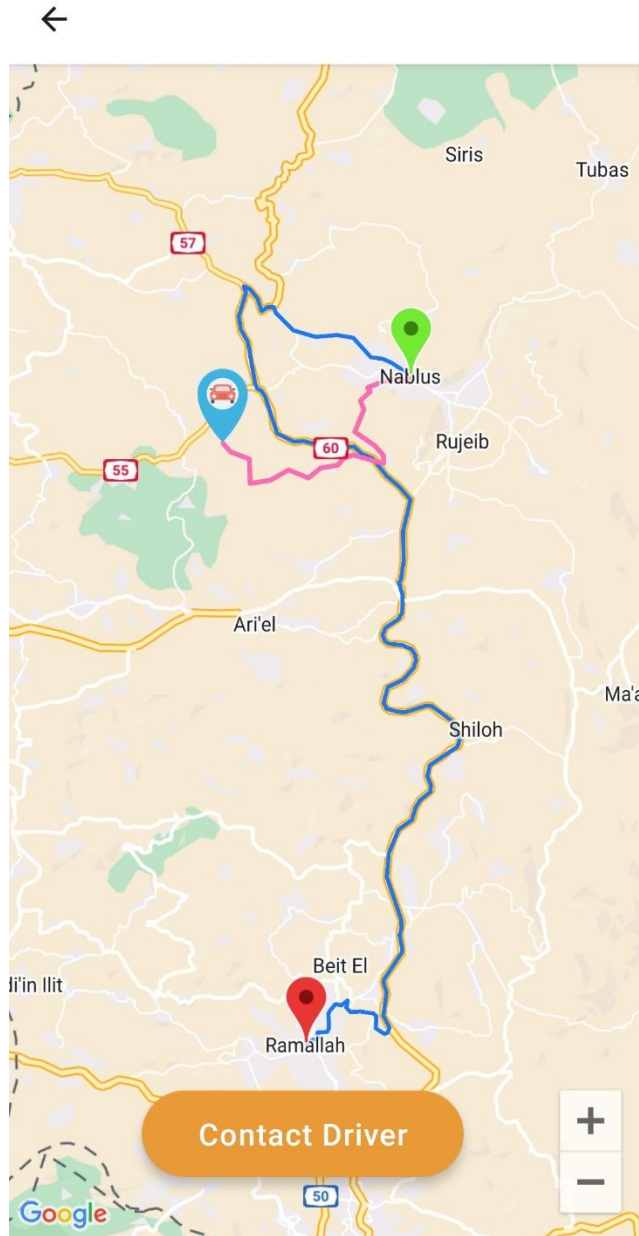
Track The Order!

*Figure 41 (order details but when its shipped)*



*Figure 42 (order details but when its shipped & delivered)*

- When the customer press track the order a map with the sender, recipient and driver locations will be displayed and the diver location will be updated so the customer can track the drive in a real time location.



*Figure 43 (Map- Tracking order)*

- When the customer press contact Driver a chat between the driver and customer will be open.



*Figure 44 (Chat between the driver and the customer)*

#### 4.3.4: Driver Side:

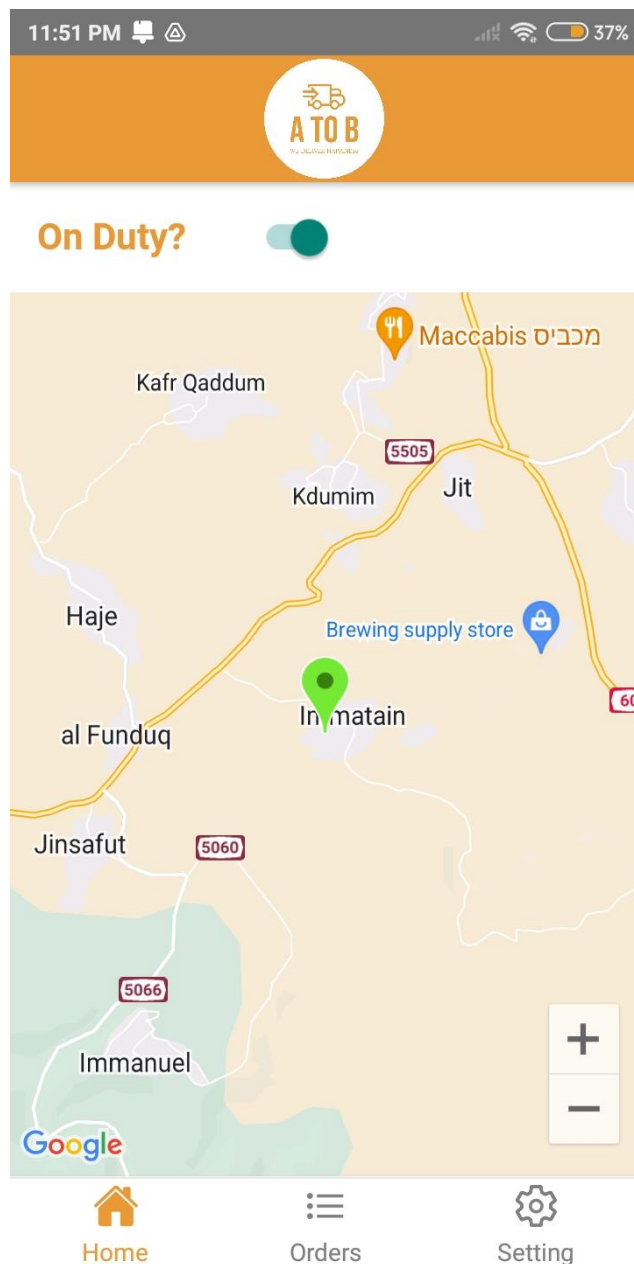
##### 4.3.4.1: Home Page:

- The first tab is Homepage it contains a map with the driver current location displayed on it and a switch button if the



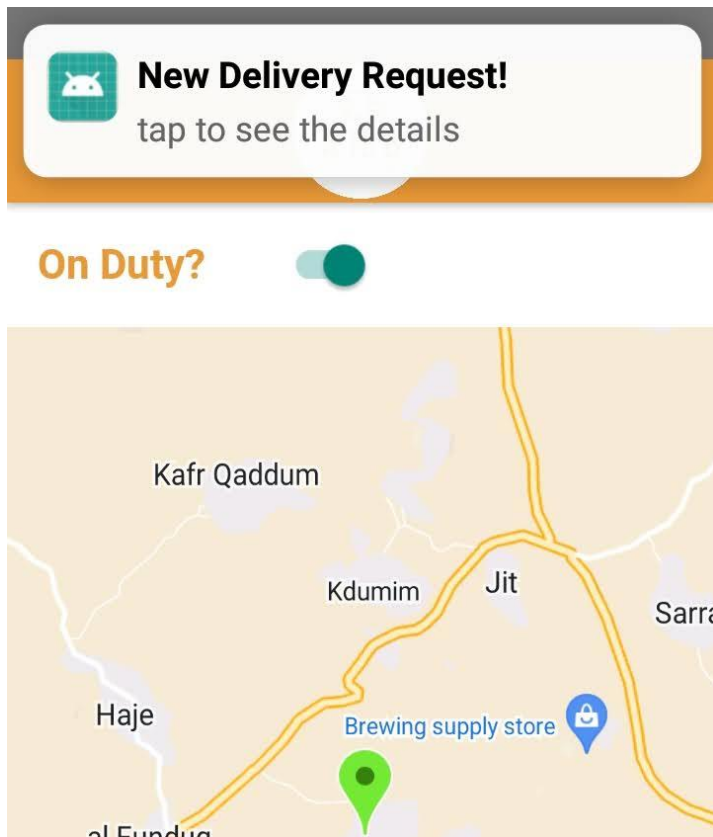
Driver switch it to on then their location will be shared with all the customers how the driver accepted their delivery.

And if it is of it give the driver the privacy to not share their location when they don't work.

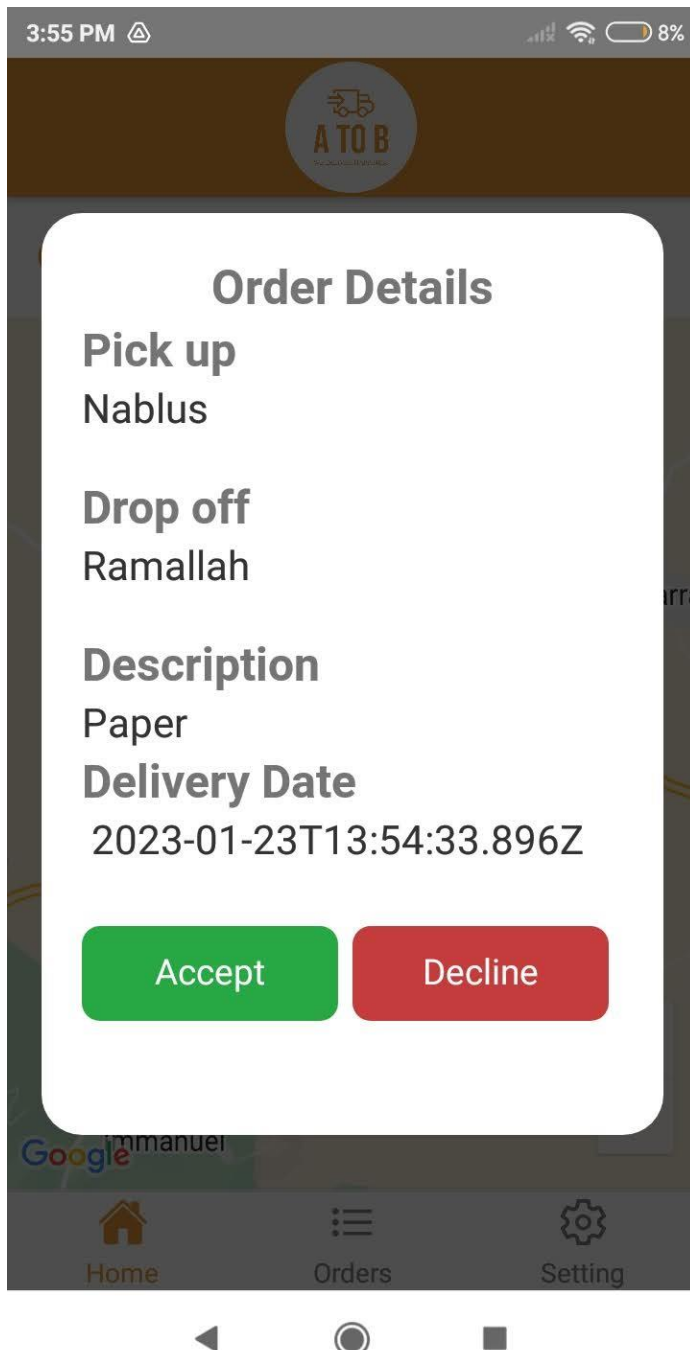


*Figure 45 (driver Home page)*

- When a new delivery request is made on the nearby area they got a notification if the user pressed the notification a modal with the order details will be displayed and there is two buttons Accept and reject.

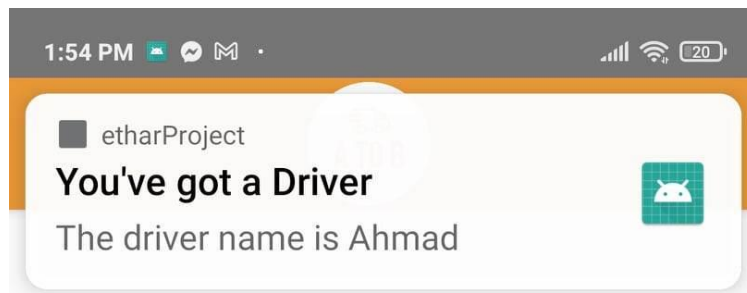


*Figure 46 (New Delivery req notification)*



*Figure 47 (the modal with the delivery req details)*

- When the driver press accepts a notification with the name of the driver who accepted it.



## Sender Information

Ethar

---

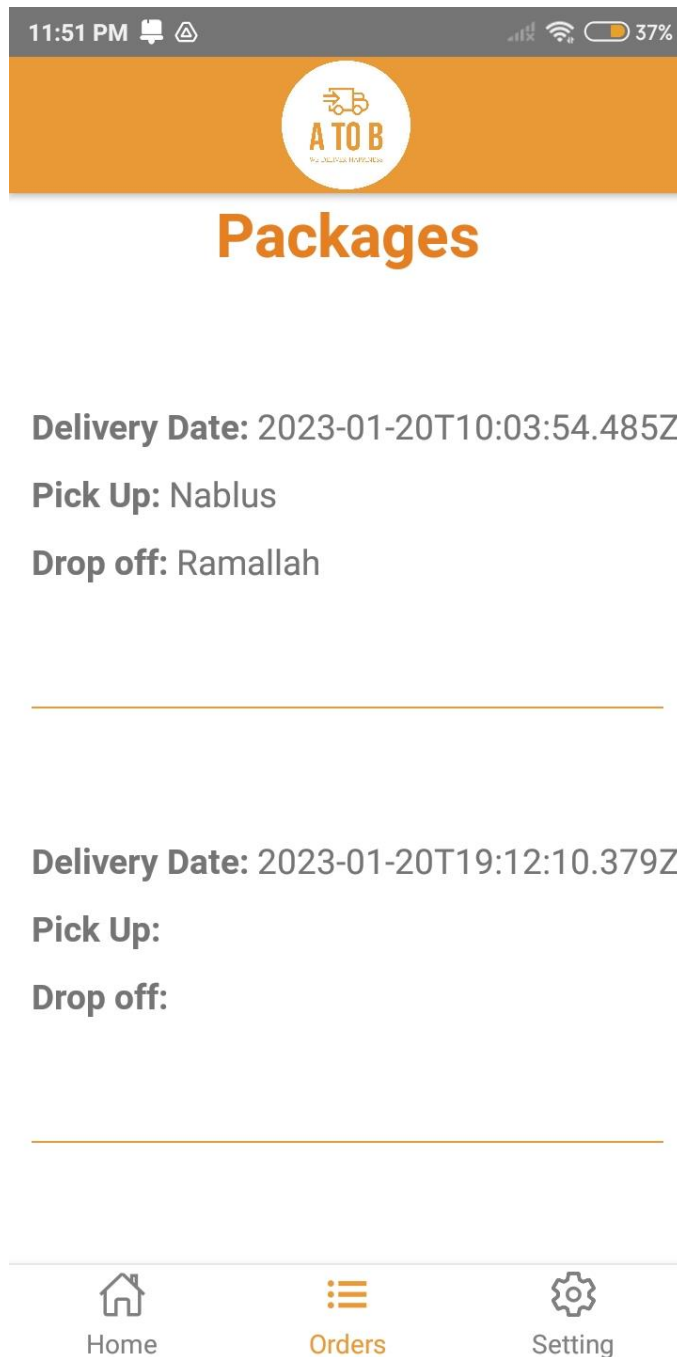
etharedu@gmail.com

---

*Figure 48 (Customer side when a driver accepts the order)*

### *4.3.4.2: Driver Orders:*

This tab contains all the orders the driver had accepted it and it give them the ability to update the status of the package and display the sender recipient and their location on the map and they can contact the customer.

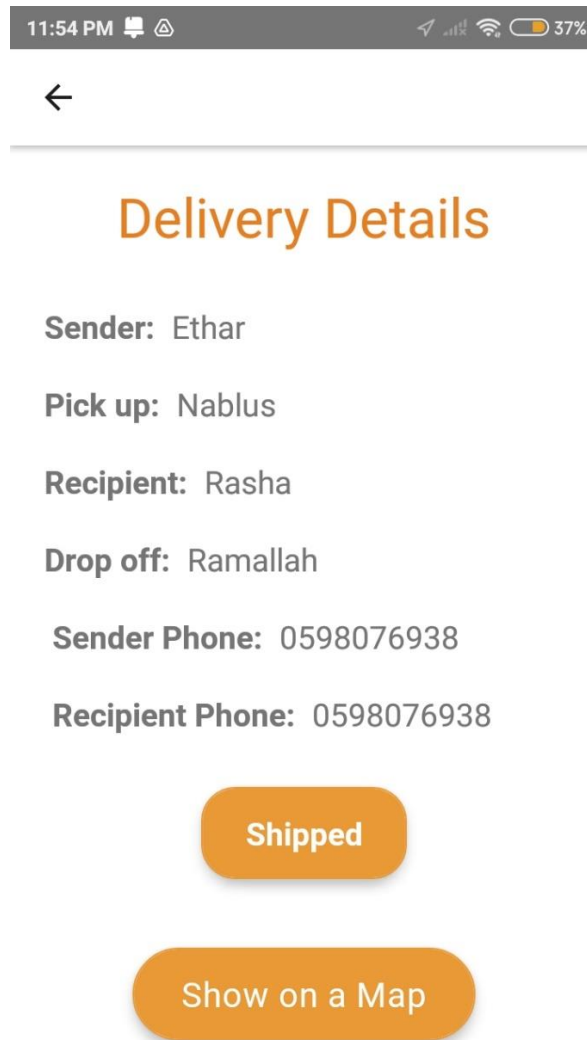


*Figure 49 (Drivers Packages)*

- When the driver press on the one of the orders a new screen is navigates with details about the order and there

is two button shipped button and the Show on a map button.

- When the driver presses the shipped button the status of the delivery is toggled to true and when the driver presses the arrived the order status toggled the delivered to true.



*Figure 50 (order details shipped)*



## Delivery Details

**Sender:** Ethar

**Pick up:** Nablus

**Recipient:** Rasha

**Drop off:** Ramallah

**Sender Phone:** 0598076938

**Recipient Phone:** 0598076938

Arrived

Show on a Map



## Delivery Details

**Sender:** Ethar

**Pick up:**

**Recipient:**

**Drop off:**

**Sender Phone:**

**Recipient Phone:**

Arrived

Show on a Map

*Figure 51 (order details delivered)*





## Delivery Details

**Sender:** Ethar

**Pick up:** Nablus

**Recipient:** Rasha

**Drop off:** Ramallah

**Sender Phone:** 0598076938

**Recipient Phone:** 0598076938

**This Order is Shipped and Deliverd**

Show on a Map

*Figure 52 (order details shipped and delivered)*

- When the driver press on Show on map button a map is shown.

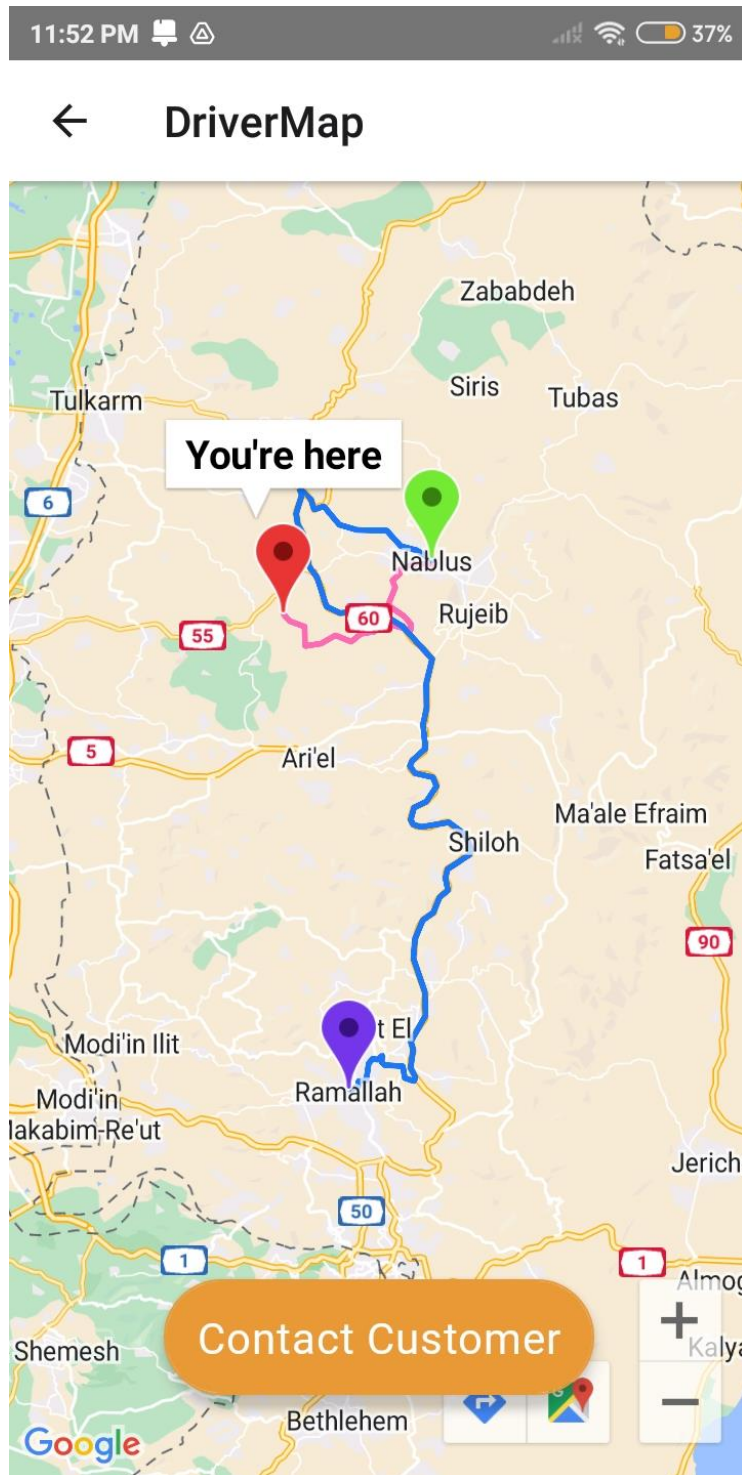


Figure 53 (Driver Map)

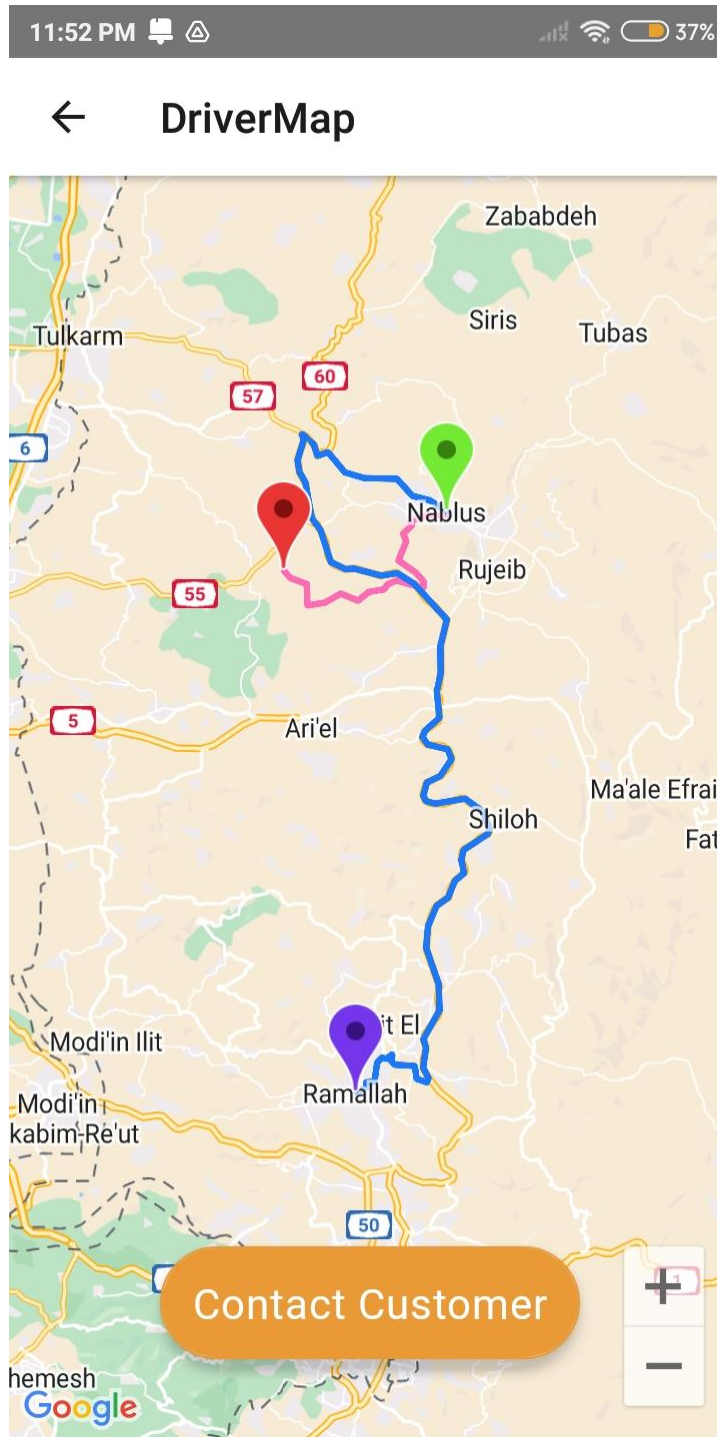


Figure 54(Driver Map)

- When the driver press on the Contact Customer button the chat between them will be opened and they can communicate.



Figure 55 (driver Chat)

And as I explained on the 4.3.1.4 there is the setting tab that have the same functionality on both Customer and Driver Sides.

#### 4.4: Website:

##### 4.4.1: Tools, Methods and Programming Languages:

##### 4.4.2 programming language:

Our website was built using two different languages:

- React-JS for the frontend.
- Node-JS for the backend.

##### 4.2.3 Tools:

- Visual Studio Code
- React 18.2.0.
- Node-JS .
- Real devices to testing our project.
- Studio 3T Free for MongoDB.
- Postman.

#### 4.4.4: Database:

We used MongoDB database and the needed collections as I mentioned on the application.

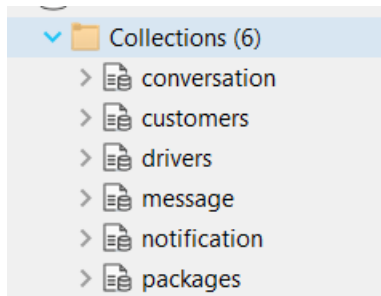


Figure 56

## 4.2 System Features Implementation:

### Chat

#### Introduction:

Almost every app and every website relies on chatting, and chatting has become an integral part of it. So it was necessary to include the chat in the website, as our site allows communication between the customers and the drivers.

There are many ways to develop a chat and the programming languages used can also decide how it has to be done. The chats in our project are made possible by Socket.IO implemented in Node-JS and React.Socket.IO is a library that enables low-latency, bidirectional and event-based communication between a client and a server.

#### implementation

To begin, we installed the required library on the Node-JS server. Following that, we wrote a code that waits for a connection to the IO server. Upon receiving the join request, the user socket will be added using the unique value that was sent from the client. When a message arrives the server searches for the receiver ID of the targeted user and sends the message to him.



```

const io = require("socket.io")(8800, {
  cors: {
    origin: "http://localhost:3000",
  },
});

let Customers = [];
let Customerdis=[];
let PackageDetails = {};

io.on("connection", (socket) => {
  socket.on("addCustomer", (newUserId) => {
    if (!Customers.some((user) => user.userId === newUserId)) {
      Customers.push({ userId: newUserId, socketId: socket.id });
      console.log("New Customer Connected", Customers);
    }
    io.emit("get-Customers", Customers);
  });

  socket.on("disconnect", () => {
    Customerdis = Customers.filter((user) => user.socketId === socket.id);
    Customers = Customers.filter((user) => user.socketId !== socket.id);
    console.log("Customer Disconnected", Customerdis);
    io.emit("get-Customers", Customers);
  });

  socket.on("send-message", (data) => {
    const { receiverId } = data;
    const user = Customers.find((user) => user.userId === receiverId);
    console.log("Sending from socket to :", receiverId)
    console.log("Data: ", data)
    if (user) {
      io.to(user.socketId).emit("recieve-message", data);
    }
  });
});

```

*Figure 57*

In the client side the connection with IO server was made.

When a change occurs to the variable `sendMessage` (when you send a new message), this message is sent to the socket.

Then the message from the socket reaches the intended recipient user.

Furthermore, the messages were all stored in the database so the user can see all the chat between them and the other users.

```
44 // Connect to Socket.io
45 useEffect(() => {
46   socket.current = io("ws://localhost:8800");
47   socket.current.emit("addCustomer", user.token._id );// currentUser
48   socket.current.on("get-Customers", (users) => {
49     setOnlineUsers(users);
50   });
51 }, [user.token]);
52
53 // Send Message to socket server
54 useEffect(() => {
55   if (sendMessage !== null) {
56     socket.current.emit("send-message", sendMessage);
57   }
58 }, [sendMessage]);
59
60
61 // Get the message from socket server
62 useEffect(() => {
63   socket.current.on("recieve-message", (data) => {
64     console.log("recieve-message", data)
65     setReceivedMessage(data);
66   }
67 );
68 }, []);
69
```

*Figure 58*

The message was stored in the database with information about the user who sent it by `senderId`, the `conversationId`, the message content as shown:

```

71     const message = {
72       senderId : currentUser,
73       text: newMessage,
74       conversationId: currentChat._id,
75     }

```

Figure 59

This code shows how messages are stored in the database:

```

// Send Message
const handleSend = async(e)=> {
  e.preventDefault()
  // console.log(newMessage)
  if(newMessage !== ""){
    const message = {
      senderId : currentUser,
      text: newMessage,
      conversationId: currentChat._id,
    }
    const receiverId = currentChat.members.find((id)=>id!==currentUser);
    setSendMessage({...message, receiverId})
    try{
      await axios.post("http://localhost:5000/messages/" , message)
      .then( response=> {
        setMessages([...messages, response.data]);
        setNewMessage("");
      })
      .catch(error=> {
        console.log(error);
      });
    }
    catch (err) {
      console.log(err);
    }
  }
}

```

Figure 60

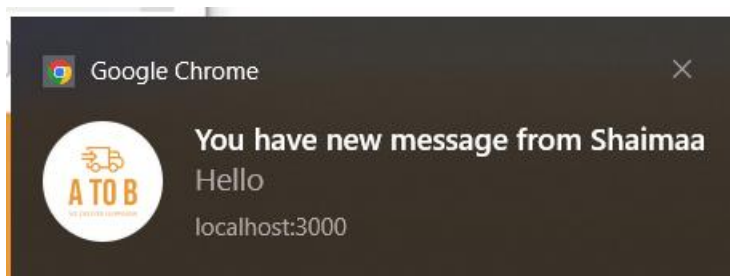
### 5.1.3 Notification System:

#### Introduction:

Notification systems are designed to let users know when something related to them occurs even if they don't use the website right then. The majority of users do not find it convenient to constantly check what is happening in the website. This makes it necessary to create a notification system to keep them connected to the website without having to exert so much effort.

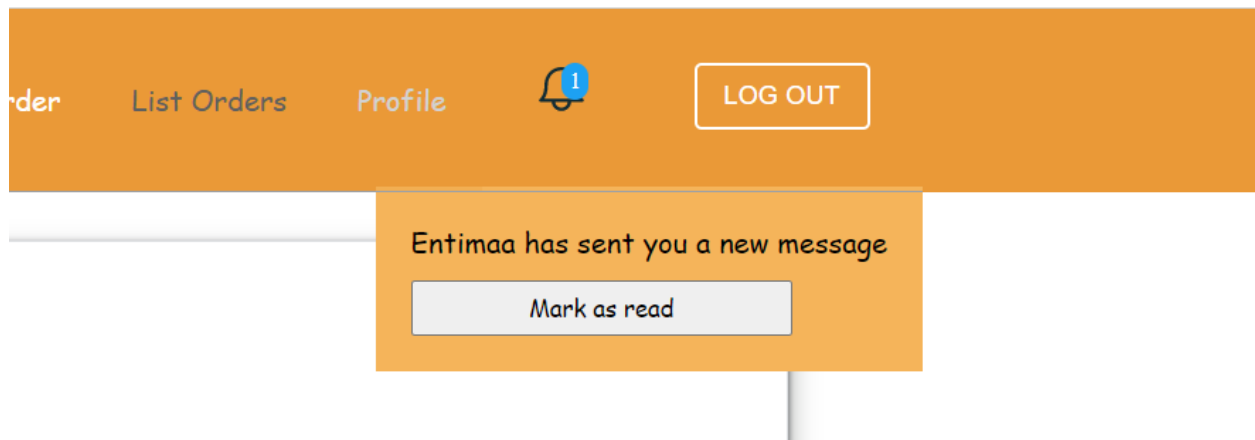
In our website, we using Socket-IO was used to make notifications happen.

At first I used React-push-notification but later I replaced the method and used socket-IO.

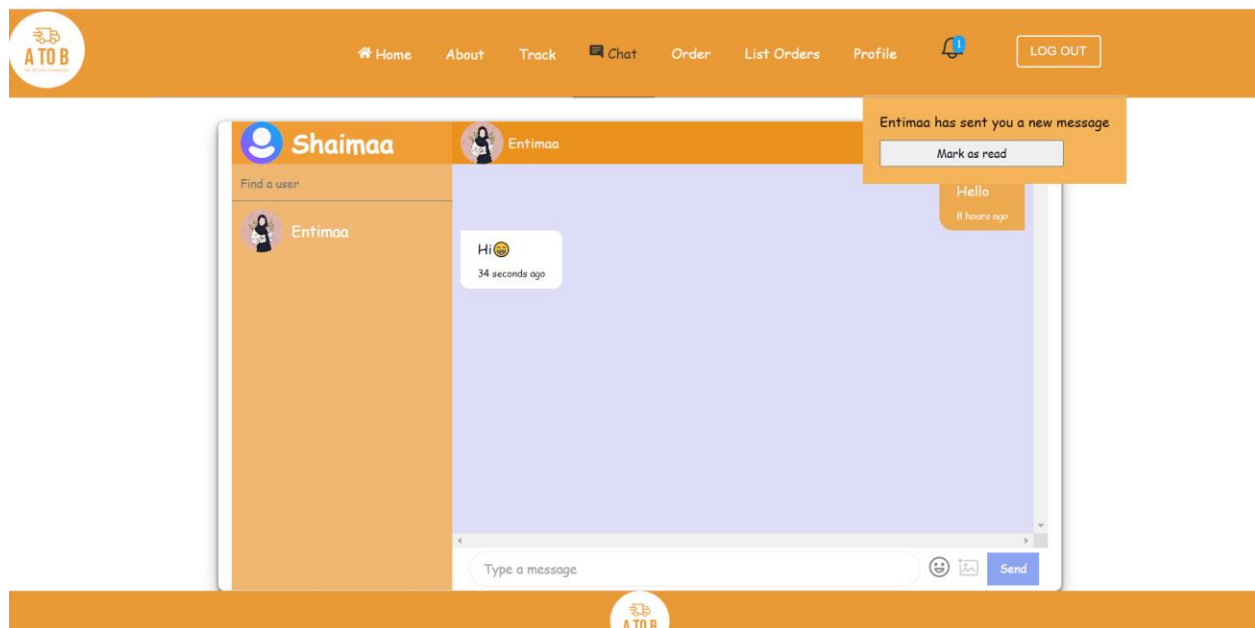


*Figure 61 notification*

React-push-notification



*Figure 62 notification*



*Figure 63 chat*

Socket-IO

implementation:

As I said earlier, I used the socket.IO, when the site is connected to the socket server , it takes the notification data from the socket server directly, and when it is not connected, we take

the notification data from the database. In both cases, the data coming from the socket server is stored in the database, and it is deleted when the user presses a button “mark as read”.

```
50 socket.on("sendNotification", (data) => {
51   const { receiverId } = data;
52   const user = Customers.find((user) => user.userId === receiverId);
53   console.log("Sending from socket to :", receiverId)
54   console.log("Data: ", data)
55   if (user) {
56     io.to(user.socketId).emit("getNotification", data);
57   }
58 });
```

*Figure 64*

This is code from socket server side.

In client side :

```
96 useEffect(() => {
97   if (sendMessage !== null) {
98     socket.current.emit("send-message", sendMessage);
99     socket.current.emit("sendNotification", user.sendNotification);
100   }
101 }, [user.sendNotification]);
102
```

*Figure 65*

This code for send notification data to Socket Server.

```

27     useEffect(() => {
28         if (islogin) {
29             socket.current.on("getNotification", (data) => {
30                 console.log("recieve Notification", data)
31                 context.setreceivedNotification(data)
32             });
33         }
34     }, [context.token]);
35
36     useEffect(() => {
37         if (islogin) {
38             console.log("Notification Arrived: ", context.receivedNotification)
39             if (context.receivedNotification !== null) {
40                 context.setNotifications([...context.notifications, context.receivedNotification]);
41             }
42         }
43     }, [context.userData.message]);
44

```

*Figure 66*

This code for get notification data from Socket Server.

```

91     const notification = {
92         senderName: useraccount?.name,
93         receiverName: receiverId,
94         type: 1,
95     }
96     setSendNotification({ ...notification, receiverId })
97     try {
98         await axios.post("http://localhost:5000/notification/", notification)
99         .then(response => {
100             context.setNotifications([...context.notifications, response.data])
101         })
102         .catch(error => {
103             console.log(error);
104         });
105     }
106     catch (err) {
107         console.log(err);
108     }
109 }
110
111 }
112

```

*Figure 67*

This is code for store notification data in database.

```
45   useEffect(() => {  
46     const fetchNotifications = async () => {  
47       try {  
48         await axios.get("http://localhost:5000/notification/" + context?.token?._id)  
49           .then(res => {  
50             context.setNotifications(res.data)  
51           }).catch(err => {  
52             console.log(err);  
53           })  
54       } catch (err) {  
55         console.log(err);  
56       }  
57     };  
58  
59     if (islogin) fetchNotifications();  
60   }, [context?.token?.id]);
```

*Figure 68*

This is code for get notification data from database.

```
77   const handleRead = () => {  
78     context.setNotifications([]);  
79     axios.delete(`http://localhost:5000/notification/` + context?.token?._id)  
80     setOpen(false);  
81   };
```

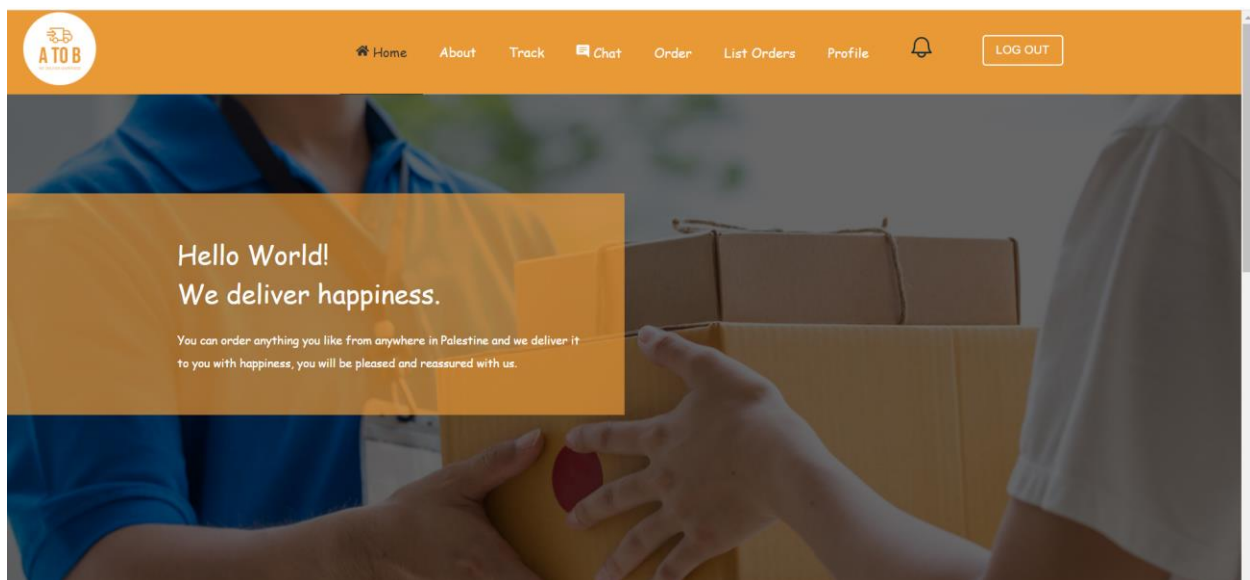
*Figure 69*

This is code for delete notification data from database.



#### 4.4 The Website:

The website was created using React.js.



*Figure 70 Home Page.*

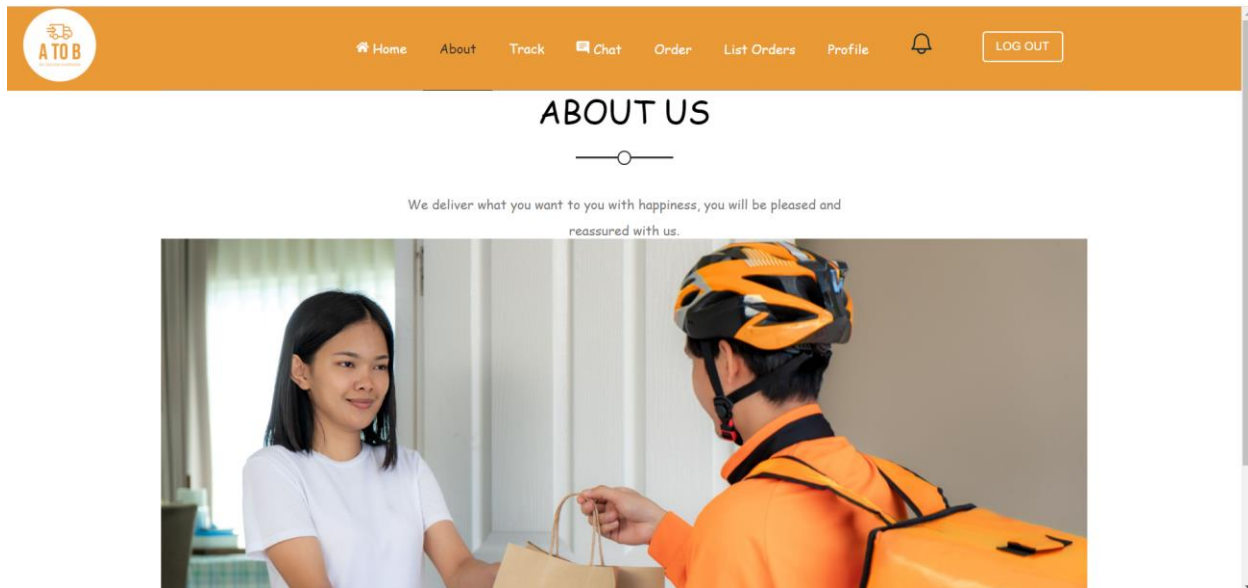


Figure 71 About Page.

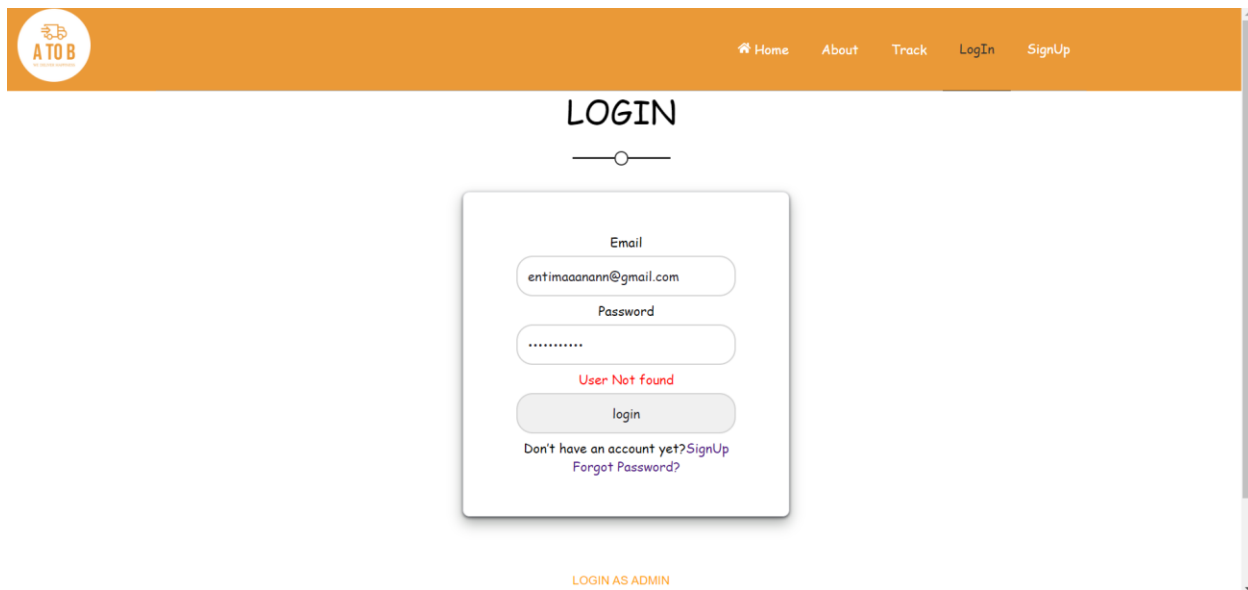


Figure 72 login

This page is to allow customers log In. The page shows warning messages if the entered values are wrong.

ATO B

Home About Track LogIn SignUp

## LOGIN AS ADMIN

Email

adminn@gmail.com

Password

.....


Admin Not found

login

[LOGIN AS CUSTOMER](#)

*Figure 73 Admin Login Page.*

This page is to allow admin log In ,and if the entered values are wrong, the page displays warning messages as show in picture.



[Home](#) [About](#) [Track](#) [LogIn](#) [SignUp](#)

## SIGNUP

First name

Enter First name

Last name

Enter Last name

Username

Enter Username

Email

Enter Email

Phone Number

Enter Phone Number

City

Enter City

Password

Enter Password

Confirm Password

Enter Password Again

SignUp

Already have an account? [Sign in](#)

*Figure 74 Sign-Up Page.*

This page allows creating a new account for the customer, making sure that the username, email and phone number are unique.

ATO B

Home About Track LogIn SignUp

## PASSWORD RECOVERY

Email

Enter Email

Proceed

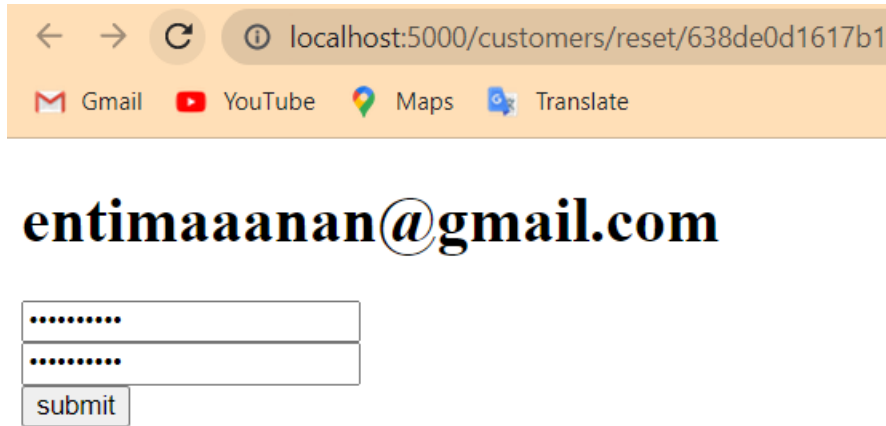
We will send you a recovery link  
Check your junk or spam folders  
for a message coming from  
atobwebsite2023@gmail.com

*Figure 75 Password Recovery Page.*





The account can be recovered in the event that the account password is forgotten, provided that the customer remembers the email that he used when creating the account, he enters it, and after that he receives a message on his email with a link to reset his account password, and once the password is reset, the link turns directly to a login page.

The following pictures show the steps that I mentioned





← → ↻ ⓘ localhost:5000/customers/reset/638de0d1617b1

 Gmail  YouTube  Maps  Translate

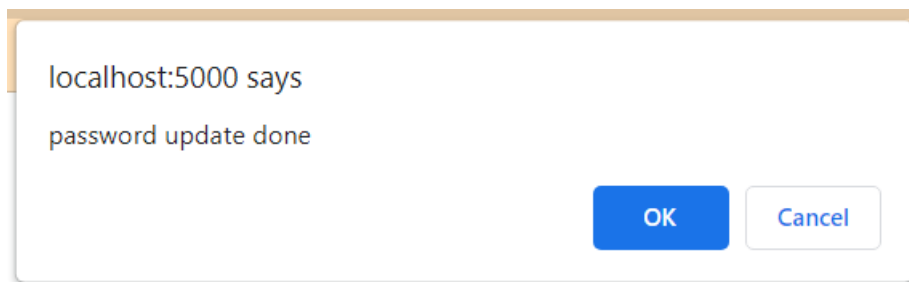
**entimaaanan@gmail.com**

.....

.....

submit

*Figure 78 Password Recovery Page.*



*Figure 79 Password Recovery Page.*

To send the email with the password change link, I used nodemailer on the server side.

```

200     const link = `http://localhost:5000/customers/reset/${oldUser._id}/${token}`;
201     var transporter = nodemailer.createTransport({
202       service: "gmail",
203       auth: {
204         user: "atobwebsite2023@gmail.com",
205         pass: "gfztrwbocgkvkhuu",
206       },
207     });
208
209     var mailOptions = {
210       from: "youremail@gmail.com",
211       to: email,
212       subject: "Password Reset",
213       text: "We have received your request to recover your AtoBWebSite account information\nYour username: "+oldUser.name+" \nTo change you
214     };
215
216     transporter.sendMail(mailOptions, function (error, info) {
217       if (error) {
218         console.log(error);
219       } else {
220         console.log("Email sent: " + info.response);
221       }
222     });

```

Figure 80

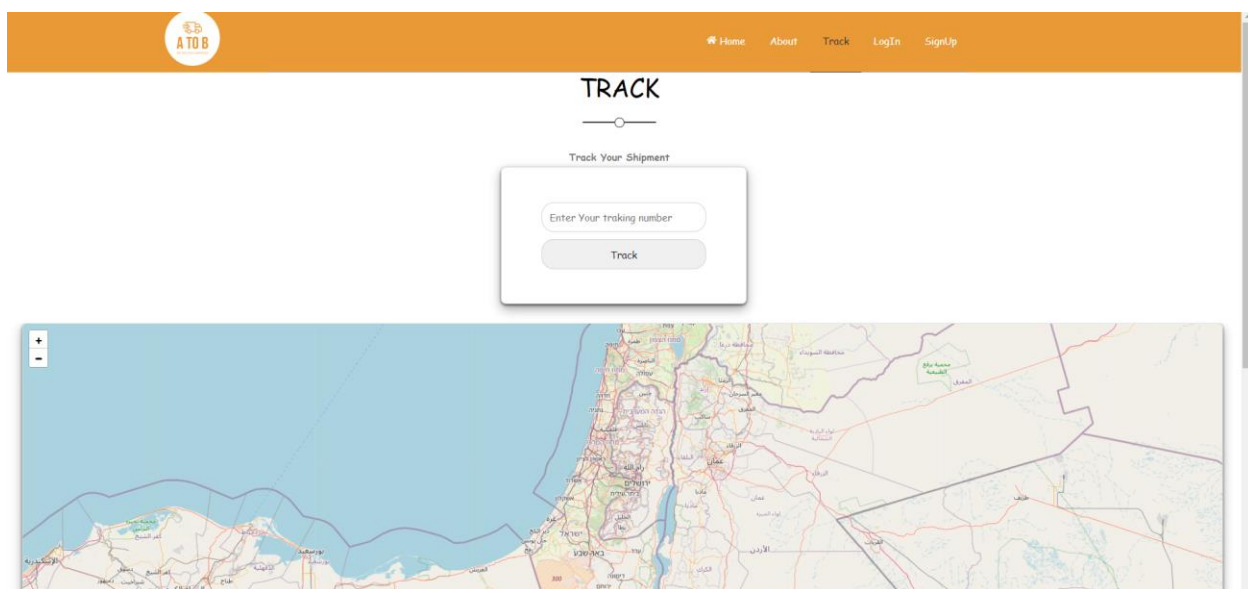
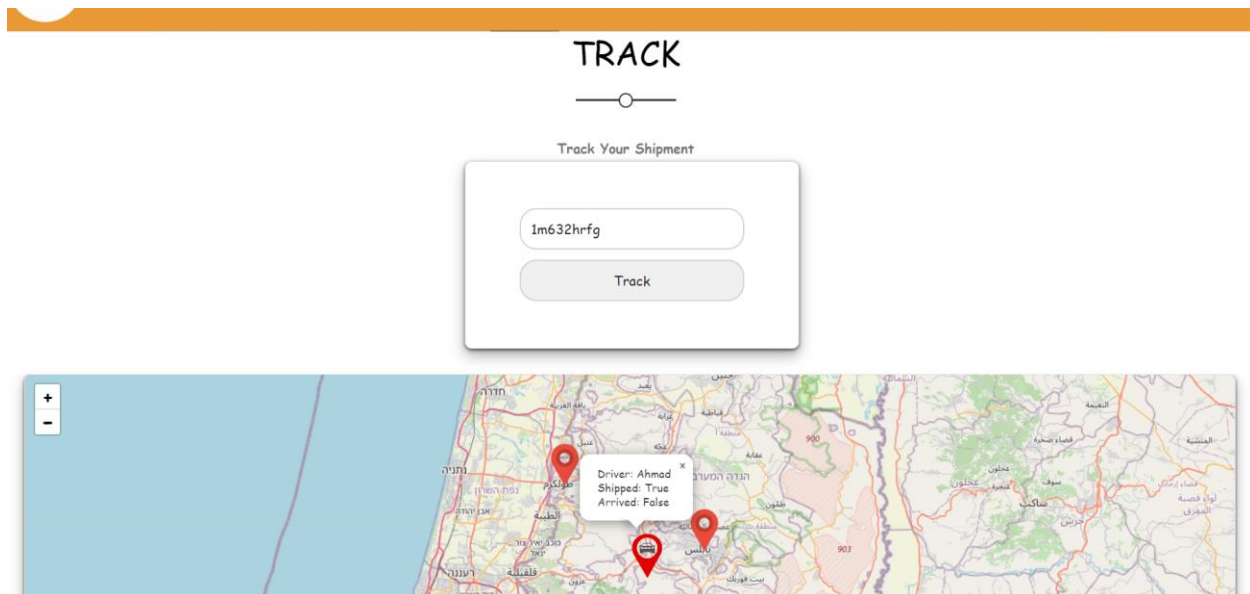


Figure 81 Tracking Page.

Through this page, anyone who has the tracking number can carry out the tracking process and find out information about the order. In the event that the order is not approved by any driver, only the location of the sender and recipient will appear on the map, and in the event that there is a driver who approved the order, the driver's location can



appear on the map. With the order information if it was shipped or not, and if it reached the receiver or not.



*Figure 82 Tracking Page.*

To create a map I used "react-leaflet", after several attempts to get a key to use Google maps using several purchase cards.

Leaflet is the leading open-source JavaScript library .

```

62   return (
63     <div className="leaf">
64       <div className="main">
65
66         <MapContainer center={center} zoom={ZOOM_LEVEL} scrollWheelZoom={true} >
67           <TileLayer
68             attribution='&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
69             url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
70           />
71           <MyMarkers data={points} />
72           {(typeof driverlat !== 'undefined') ? <Marker position={[driverlat, driverlong]} icon={markerIconcar}>
73             <Popup>
74               Driver: {driver.name} <br />
75               Shipped: {shipped ? 'True' : 'False'}<br />
76               Arrived: {arrived ? 'True' : 'False'} </Popup>
77           </Marker> : <></>
78         </MapContainer>
79
80       </div>
81     </div>
82   )
83 }
84
85
86 export default LeafletMap

```

Figure 83

With this code I have created the map.

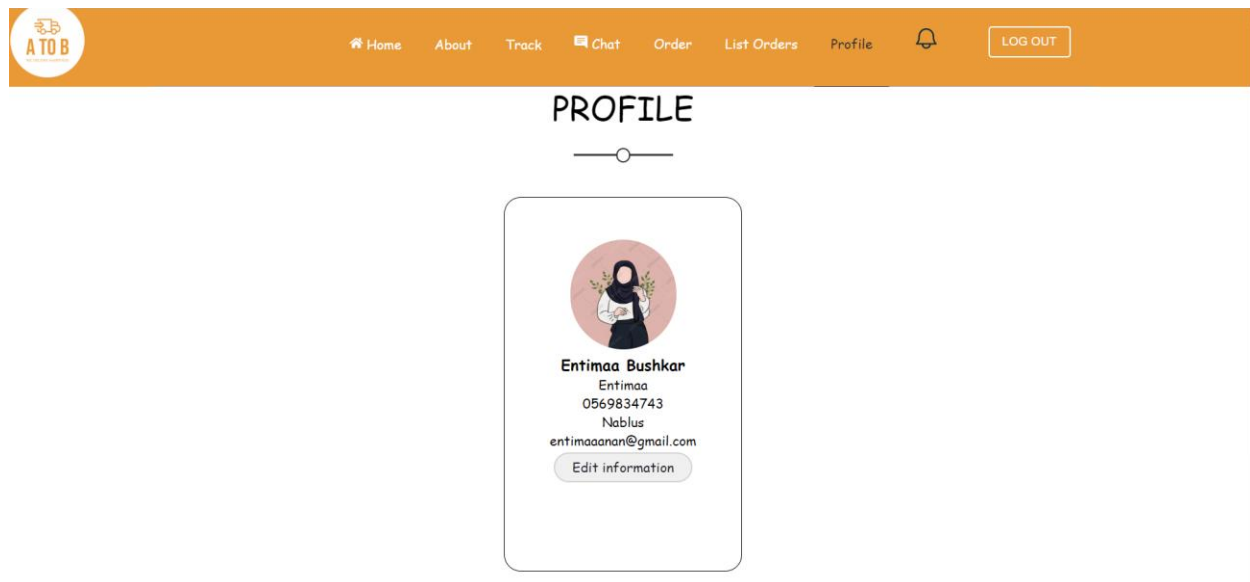
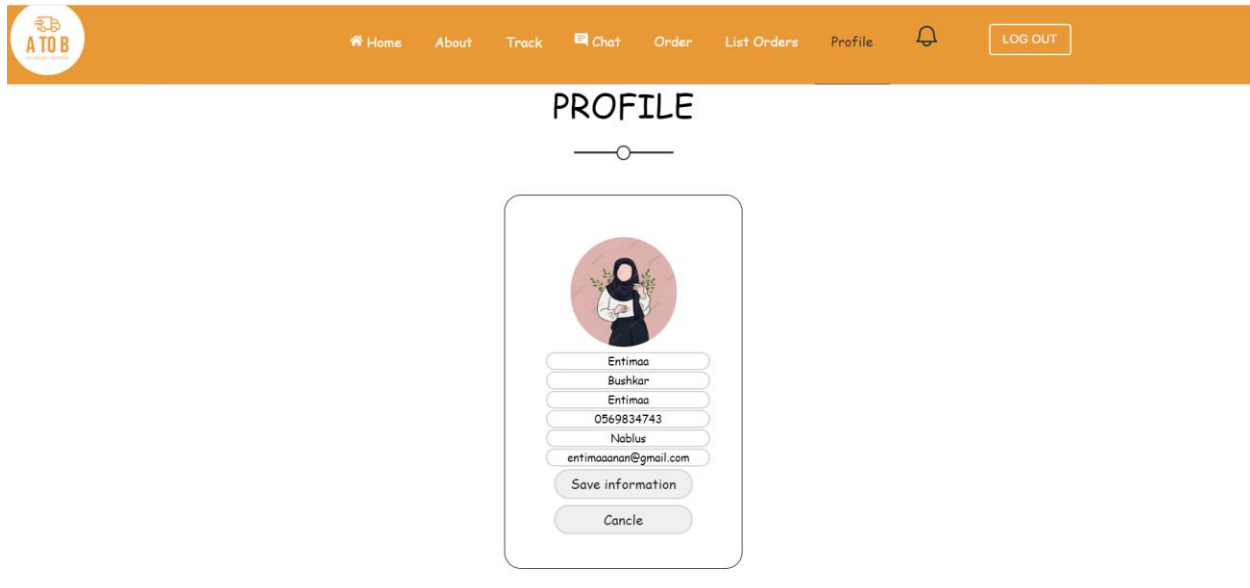


Figure 84 Profile Page.

Figure 85



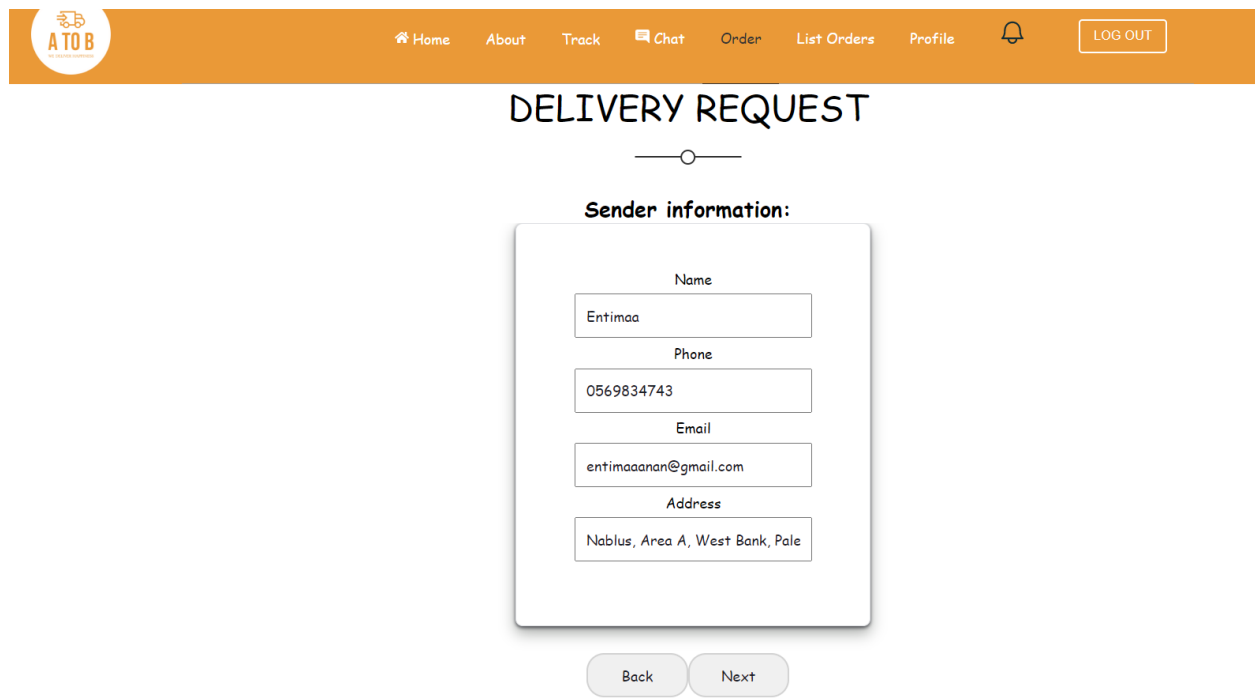
*Figure 86*

Through this page, the customer can view and modify his account information as well.

*Figure 87 Request Page.*



Through this page, the customer can submit a delivery request that contains the information of the sender and recipient, as well as the information of the order itself.

Where the customer's information is fixed, taken from the data base, except for his location, which can change.



The screenshot shows a web application interface for a delivery request. At the top is an orange navigation bar with the ATOB logo on the left and links for Home, About, Track, Chat, Order, List Orders, Profile, and a LOG OUT button on the right. Below the navigation bar, the title "DELIVERY REQUEST" is centered, followed by a horizontal line with a circle in the middle. The main content area is titled "Sender information:" and contains a form with four input fields: "Name" (filled with "Entimaa"), "Phone" (filled with "0569834743"), "Email" (filled with "entimaanan@gmail.com"), and "Address" (filled with "Nablus, Area A, West Bank, Pale"). Below the form are two buttons: "Back" and "Next".

*Figure 88Request Page.*


[Home](#)
[About](#)
[Track](#)
[Chat](#)
[Order](#)
[List Orders](#)
[Profile](#)

[LOG OUT](#)

## DELIVERY REQUEST

---

**Recipient information:**

Name

Fidaa

Phone


059866157

Email

fidaanan1993@gmail.com

Address

tulkarm


Tulkarm, Area A, West Bank,  
302, Palestinian Territory





Tulkarm, King Husein, تulkarm,  
Tulkarm, Area A, West Bank,  
302, Palestinian Territory

Figure 89Request Page.


[Home](#)
[About](#)
[Track](#)
[Chat](#)
[Order](#)
[List Orders](#)
[Profile](#)


LOG OUT

## DELIVERY REQUEST

---

**Package information:**

Delivery date

1 / 24 / 2023

X

Description

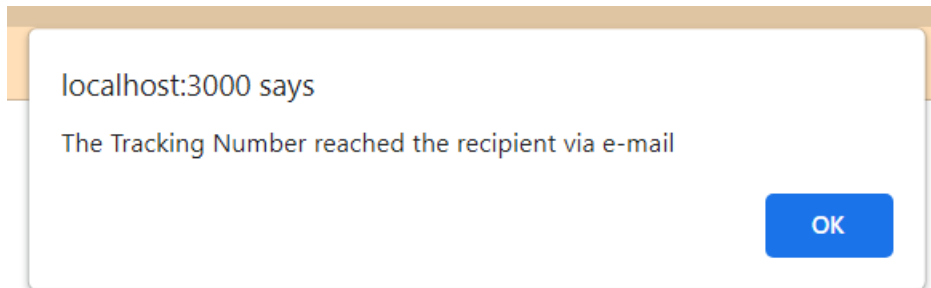
Clothes  
cost:100ILS

Back

Submit

Figure 90Request Page.

After sending the order, a message will be sent via e-mail to the recipient, bearing the tracking number, package description, price list, and tracking page link, so that he can track his package.



### Package information:

Figure 91 Request Page.

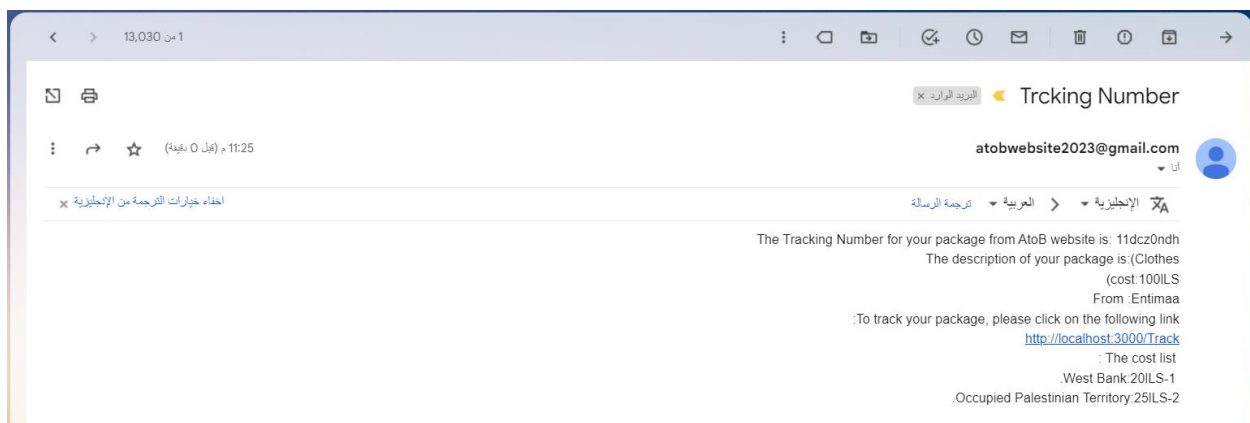


Figure 92 Request Page.

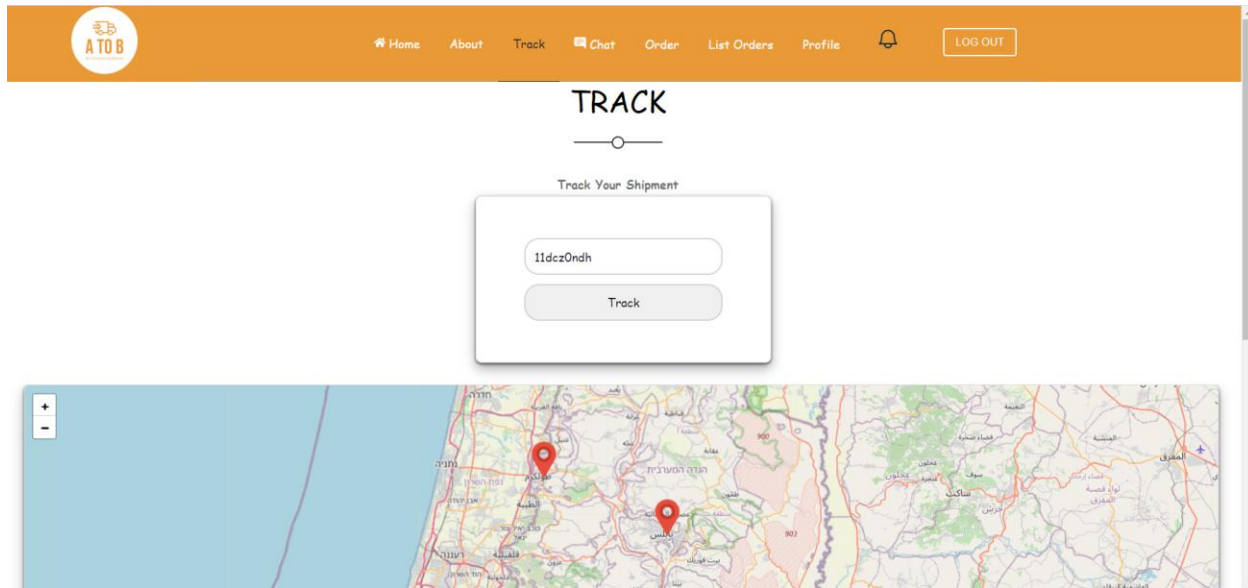


Figure 93

After that, it will go directly to the list orders page.

Through this page, the customer tracks the information of the orders that he sends.

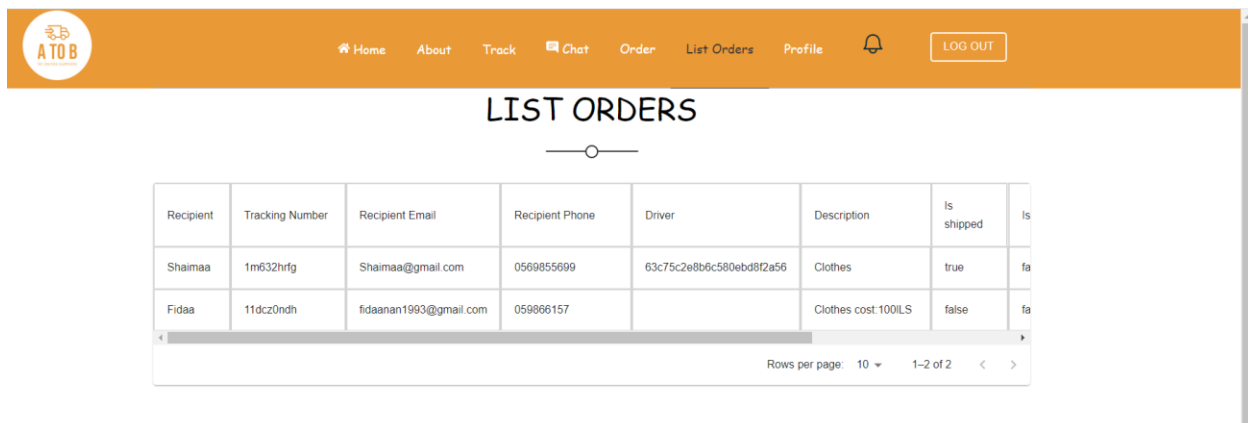


Figure 94

Admin:

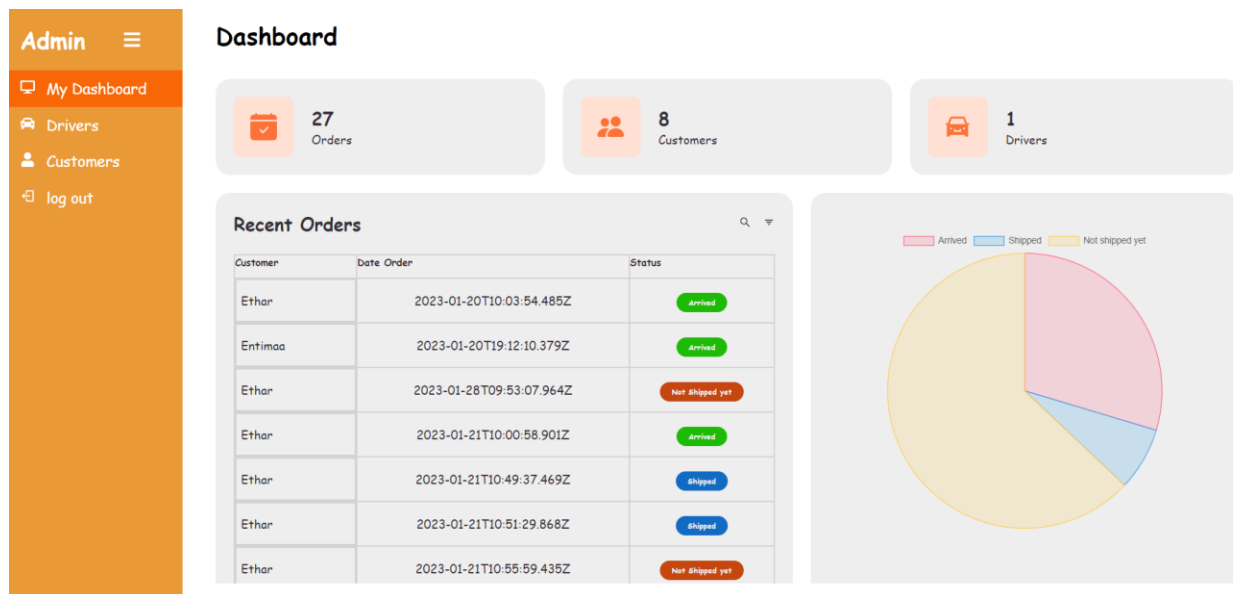


Figure 95 Dashboard Page.

This is a dashboard page with a graph showing the status of orders, showing the number of orders in general and the number of both customers and drivers using the site.

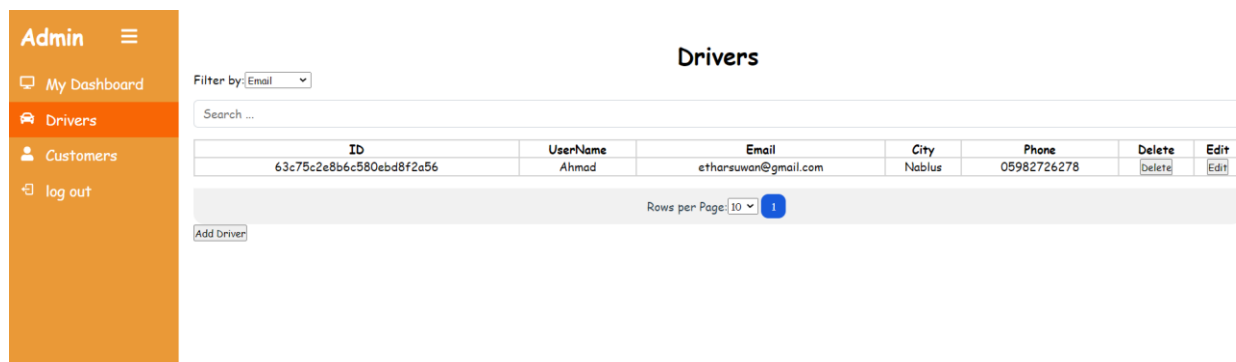


Figure 96

Drivers data page, where the admin can use filtering, searching, modifying and deleting the data as required. It also has the ability to add a driver.



Admin

My Dashboard

Drivers

Customers

log out

Customers

Filter by: First Name

Search ...

ID	First Name	Last Name	UserName	Email	City	Phone	Delete	Edit	
63c695138d9c3fb543ab88d1	Ethar	Suwan	Ethar	etharedu@gmail.com	Nablus	0598093858	Delete	Edit	Show
638de0d1617b17f8ade137c0	Entimaa	Anan	Entimaa	entimaanan@gmail.com	Nablus	0569834743	Delete	Edit	Show
63c71bb96a6cf27bc2b1d34e			Rasha	S11821362@stu.najah.edu	nablus	05983983684	Delete	Edit	Show
63c89eb75553000a2382683d			Bushra	etharsuwan@gmail.com	Nablus	0598027299	Delete	Edit	Show
63cbca748136a8ea5418a390	rasha	suwan	rashasuwan	atobapplication@gmail.com	nablus	8383838333	Delete	Edit	Show
63cbff74846dfb0e5f088576			Bushra	ethaan@gmail.com	Qalqiya	0598095936	Delete	Edit	Show
63cc5e27a2e64d009ba9bbe7	Fidaa	Anan	Fidaa	fidaanan1993@gmail.com	Nablus	0569922657	Delete	Edit	Show
63cc61c6a2e64d009ba9bcaa	Asem	Bushkar	aseem	aseem@gmail.com	Nablus	05454842569	Delete	Edit	Show

Rows per Page: 10

Add Customer

Figure 97

Customers data page, where the admin can use filtering, searching, modifying and deleting the data as required. It also has the ability to add a customer.

## Chapter 5: CONCLUSION & RECOMMENDATION

### 5.1 CONCLUSION:

In conclusion, the A to B delivery system is a user-friendly and efficient solution for delivery services. The system streamlines the delivery process for customers and drivers by allowing customers to place orders online, track their order on a map and monitor its status, and contact the driver directly within the app. Additionally, the system includes a website that allows customers to place orders and track them from a platform of their preference.

Overall, the A to B delivery system provides an efficient solution for delivery services, while ensuring customer satisfaction.

### 5.2 RECOMMENDATIONS:

In order to improve the A to B delivery system, the following recommendations can be considered:

- Implement a rating system for customers and drivers to provide feedback and improve the overall delivery experience.
- Add a payment system.
- Include voice records and voice calls on the chat between the driver and customer.