**CAN THO UNIVERSITY**

**COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**🙞 🕮 🙜**

**Logo

Description automatically generated**

**GRADUATION THESIS IN**

**INFORMATION TECHNOLOGY**

**(HIGH-QUALITY PROGRAM)**

**EXERCISE CORRECTION WITH MACHINE LEARNING**

**Student: Ngô Hồng Quốc Bảo**

**Student ID: B1809677**

**Class: 2019 - 2023 (K44)**

**Advisor: Prof. Trần Công Án**

**Can Tho, 12/2022**

**CAN THO UNIVERSITY**

**COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**🙞 🕮 🙜**

**Logo

Description automatically generated**

**GRADUATION THESIS IN**

**INFORMATION TECHNOLOGY**

**(HIGH-QUALITY PROGRAM)**

**EXCERCISE CORRECTION WITH MACHINE LEARNING**

**Student: Ngô Hồng Quốc Bảo**

**Student ID: B1809677**

**Class: 2019 - 2023 (K44)**

**Advisor: Prof. Trần Công Án**

**Can Tho, 12/2022**

INSTRUCTOR’S COMMENTS

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Can Tho, ………………. 2022

(Instructor sign and write full name)

**ACKNOWLEDGMENT**

*I cannot express enough thanks to my instructors/teachers from the College of Information and Communication technology for their continued support and encouragement. My sincere thanks especially go to Prof. Trần Công Án for his guidance and advices throughout the development of this project*.

*My completion of this project could not have been accomplished without the support of my classmates. The supports and advices from them have always been the biggest motivation for me during the whole process. Their encouragement when the times got rough are much appreciated and duly noted.*

*Last but not least, I would like to thank my parents who helped me a lot in gathering different information, collecting data and guiding me, despite their busy schedules, they had always been there with me from time to time in making project.*

Can Tho, December 2022

Student

Ngô Hồng Quốc Bảo

**ABSTRACT**

Fitness is becoming an important part of human life as it brings many beneficial to personal health. However, exercises can also be ineffective and could be dangerous if they are performed incorrectly by the performer. Proper form when doing anything physical is important, but it is especially important when working out or training. Not only can proper form lower your risk of injury but it also allows you to move efficiently, increase your performance, and enables you to have a full range of motion. In my project, I use machine learning to provides detailed analysis and recommendation on the exercises’ performer to improve their form.

In addition, with deep learning and computer vision are being strongly researched and improve by the day. Especially the development of Mediapipe by Google, an open-source framework to “build word-class machine learning solutions” that provides cornerstone Machine Learning models for common tasks like hand tracking, posture detection, ... Make use of the power of Mediapipe’s pose detection, this project “Exercise Correction with Machine Learning” is built in order to analyze, detect and classifying the forms of fitness exercises. The experimental results show that the algorithm proposed in this paper can effectively identify correct and incorrect forms that are performed in an exercise.

**TABLE OF** **CONTENTS**

[INSTRUCTOR’S COMMENTS 1](#_Toc118582211)

[INTRODUCTION 9](#_Toc118582212)

[1. PROBLEM DESCRIPTION 9](#_Toc118582213)

[2. PURPOSE OF THE STUDY 9](#_Toc118582214)

[3. LIMITATION AND SCOPE 10](#_Toc118582215)

[4. GENERAL APPROACH 10](#_Toc118582216)

[5. CRITERIA FOR STUDY SUCCESS 10](#_Toc118582217)

[6. STUDY CONTENTS 10](#_Toc118582218)

[CONTENTS 11](#_Toc118582219)

[CHAPTER 1: AN OVERVIEW OF TECHNOLOGY AND LIBRARY USED 11](#_Toc118582220)

[1. COMPUTER VISION 11](#_Toc118582221)

[1.1. Definition 11](#_Toc118582222)

[1.2. Computer vision common tasks and real-life applications 12](#_Toc118582223)

[2. MEDIAPIPE FRAMEWORK 13](#_Toc118582224)

[2.1. Definition 13](#_Toc118582225)

[2.2. MediaPipe Architecture 14](#_Toc118582226)

[2.3. Advantages of MediaPipe 15](#_Toc118582227)

[3. MEDIAPIPE POSE 16](#_Toc118582228)

[3.1. Overview and Machine Learning pipeline 16](#_Toc118582229)

[3.2. BlazePose detector 17](#_Toc118582230)

[3.3. MediaPipe Pose Output 18](#_Toc118582231)

[4. OTHER TECHNOLOGY AND LIBRARIES USED 18](#_Toc118582232)

[4.1. Open CV 18](#_Toc118582233)

[4.2. Scikit-learn 18](#_Toc118582234)

[4.3. Keras 18](#_Toc118582235)

[4.4. Vue.js 18](#_Toc118582236)

[4.5. Django 18](#_Toc118582237)

[CHAPTER 2: GENRAL APPROACH ON BUILD MACHINE LEARNING MODELS 19](#_Toc118582238)

[1. EXERCISES SELECTION AND ERRORS DETERMINATION 19](#_Toc118582239)

[2. DATA COLLECTING 20](#_Toc118582240)

[2.1. Self-collected data 20](#_Toc118582241)

[2.2. Public Dataset for Kaggle 21](#_Toc118582242)

[3. DATA PROCESSING 21](#_Toc118582243)

[3.1. Detecting important landmarks 21](#_Toc118582244)

[3.2. Extract data from video using OpenCV and MediaPipe 22](#_Toc118582245)

[4. MODEL TRAINING 23](#_Toc118582246)

[4.1. Classification with Scikit-learn 23](#_Toc118582247)

[4.2. Neural Network for classification with Keras 24](#_Toc118582248)

[CHAPTER 3: DELIVERY SYSTEM DESIGN 26](#_Toc118582249)

[1. DATABASE DESIGN: 26](#_Toc118582250)

[2. FINDING AND NAVIGATING BETWEEN LOCATIONS 30](#_Toc118582251)

[2.1. Finding consignee location 30](#_Toc118582252)

[2.2. Navigating and distance calculating for order delivery 31](#_Toc118582253)

[3. USER INTERFACE DESIGN 32](#_Toc118582254)

[3.1. Login/Register page 32](#_Toc118582255)

[3.2. Orders Management page 33](#_Toc118582256)

[3.3. Employee Management page 33](#_Toc118582257)

[3.4. Profile page 35](#_Toc118582258)

[3.5. Pricing list management page 36](#_Toc118582259)

[3.6. Reports page 37](#_Toc118582260)

[3.7. Order Detail page 37](#_Toc118582261)

[3.8. Order placement page 38](#_Toc118582262)

[CHAPTER 4: TESTING AND EXPERIMENTAL RESULTS 41](#_Toc118582263)

[1. The purposes of testing 41](#_Toc118582264)

[2. Testing explanation 41](#_Toc118582265)

[2.1. Details of the testing plan 41](#_Toc118582266)

[2.1.1. Testing features 41](#_Toc118582267)

[2.1.2. The goal of testing 42](#_Toc118582268)

[2.1.3. Result from the test cases 42](#_Toc118582269)

[2.2. Test environment 42](#_Toc118582270)

[2.3. Test cases 43](#_Toc118582271)

[2.3.1. Login 43](#_Toc118582272)

[2.3.2. Register 44](#_Toc118582273)

[2.3.3. Add/Edit pricing list 44](#_Toc118582274)

[2.3.4. Search for order by its id 45](#_Toc118582275)

[2.3.5. Find consignee information 46](#_Toc118582276)

[2.3.6. Navigate and calculate distance 47](#_Toc118582277)

[CONCLUSION 48](#_Toc118582278)

[1. Final results 48](#_Toc118582279)

[2. Future works 48](#_Toc118582280)

[REFERENCES 49](#_Toc118582281)

**LIST OF FIGURES**

[Figure 1: Computer Vision example 11](#_Toc118279823)

[Figure 2: Self-driving car using computer vision 13](#_Toc118279824)

[Figure 3: Object detection pipeline in MediaPipe 14](#_Toc118279825)

[Figure 4: Face landmark detection and segmentation using MediaPipe 15](#_Toc118279826)

[Figure 5: The 33 landmarks model in MediaPipe Pose predicts 16](#_Toc118279827)

[Figure 6: MediaPipe pose detection's pipeline 17](#_Toc118279828)

[Figure 7: Example of pose detection by MediaPipe 18](#_Toc118279829)

[Figure 8: A person doing lunge 19](#_Toc118279830)

[Figure 9: Example of a directory tree structure for each exercise collected videos 20](#_Toc118279831)

[Figure 12: Database diagram of the system 22](#_Toc118279832)

[Figure 13: System Screenshot - Register page 28](#_Toc118279833)

[Figure 14: System Screenshot - Login page 28](#_Toc118279834)

[Figure 15: System Screenshot - Order management page 29](#_Toc118279835)

[Figure 16: System Screenshot - Employee management 30](#_Toc118279836)

[Figure 17: System Screenshot - Consignor management page 30](#_Toc118279837)

[Figure 18: System Screenshot - Consignee management page 31](#_Toc118279838)

[Figure 19: System Screenshot - Profile page 31](#_Toc118279839)

[Figure 20: System Screenshot - Pricing list page on user view 32](#_Toc118279840)

[Figure 21: System Screenshot - Pricing list page on manager view 32](#_Toc118279841)

[Figure 22: System Screenshot - Report page 33](#_Toc118279842)

[Figure 23: System Screenshot - Order detail page 33](#_Toc118279843)

[Figure 24: System Screenshot - Tracking order in Order detail page 34](#_Toc118279844)

[Figure 25: System Screenshot - Order placement page 34](#_Toc118279845)

[Figure 26: System Screenshot - Order confirmation 35](#_Toc118279846)

**LIST OF TABLES**

[Table 1: Feature description - Login 21](#_Toc89681520)

[Table 2: Feature description - Register 22](#_Toc89681521)

[Table 3: Feature description - Order searching 23](#_Toc89681522)

[Table 4: Feature description - Order placement 24](#_Toc89681523)

[Table 5: Feature description - Order management 25](#_Toc89681524)

[Table 6: Feature description - Account management 26](#_Toc89681525)

[Table 7: Feature description - Pricing list 27](#_Toc89681526)

[Table 8: Feature description - Statistical report 28](#_Toc89681527)

[Table 9: Database design - User table 31](#_Toc89681528)

[Table 10: Database design - Profile table 31](#_Toc89681529)

[Table 11: Database design - Order table 32](#_Toc89681530)

[Table 12: Database design - Unit price table 33](#_Toc89681531)

[Table 13: Database design - Product table 33](#_Toc89681532)

[Table 14: Database design - Delivery table 33](#_Toc89681533)

[Table 15: Features require to be tested 45](#_Toc89681534)

[Table 16: Feature test cases - Login 46](#_Toc89681535)

[Table 17: Feature test cases - Register 47](#_Toc89681536)

[Table 18: Feature test cases - Add/Edit pricing list 48](#_Toc89681537)

[Table 19: Feature test cases - Order searching 49](#_Toc89681538)

[Table 20: Feature test cases - Find consignee location 50](#_Toc89681539)

[Table 21: Feature test cases - Navigate and calculate distance 50](#_Toc89681540)

INTRODUCTION

1. PROBLEM DESCRIPTION

The advancements in artificial intelligence and computational power, computer vision technology has taken a huge leap toward integration in our daily lives. Computer vision is the field of computer science that focuses on creating digital systems that can process, analyze, and make sense of visual data (images or videos) in the same way that humans do. The concept of computer vision is based on teaching computers to process an image at a pixel level and understand it. Technically, machines attempt to retrieve visual information, handle it, and interpret results through special software algorithms.

In addition, nowadays, exercise is proven to be extremely important for people’s daily lives. Being physically active can improve your brain health, help manage weight, reduce the risk of disease, strengthen bones and muscles, and improve your ability to do everyday activities. Everyone can experience the health benefits of physical activity regardless of their age, gender, ethnicity, shape ... When it comes to exercises quality is more important than quantity. How an exercise is performed may mean the difference between going harder and getting sidelined. Anyone the more experienced can benefit from some occasional form feedback. Perfecting form will boost performance, conserve energy and reduce injuries over time.

However, for many people, especially beginners or people who mainly do their exercise by themselves, a noticeable amount of them perform these exercises improper form due to the lack of awareness or instruction. With improper form, you risk straining/injuring your body rather than training it. One thing is for sure, you are not going to see results if you are sidelined with a frustrating injury.

This thesis is intended to build machine learning models for different exercises with the help of Mediapipe framework. For each exercise, the correspond model will detect whether if the exercise is performed properly or not. The trained model will be applied to build a web application where users can provide their workout videos to get feedbacks on their form.

1. PURPOSE OF THE STUDY

The purpose of this study is to develop 4 machine learning models for 4 of the most home exercises which each model can detect any form of incorrect movement while a person is performing a correspond exercise. In addition, a web application that utilize the trained models, will be built in other to analyze and provide feedbacks on workout videos.

1. LIMITATION AND SCOPE

This study researches deep learning knowledge such as computer vision, neural network and Mediapipe framework. The study also uses Python programming language, Open CV library for image processing; Sci-kit learn library and Keras library for building machine learning model; Vue.js framework and Django framework for the web application that utilizes the trained model for feedbacks on incorrect form to exercise videos.

1. GENERAL APPROACH

* Research on which popular exercises which commonly improperly perform.
* Research on which technology to choose that is suitable to solve the problem.
* Collect and process data of the chosen exercises.
* Train and evaluate model for each exercise.
* Build a web application utilize all the trained models.
* Evaluate the final model and test the web application.

1. CRITERIA FOR STUDY SUCCESS

* Successfully build 4 models for 4 exercises which can detect proper and improper form of each exercise.
* Successfully build a web application which apply the trained models for providing feedbacks and recommendation for users bases on their exercise videos.
* Earn knowledge on deep learning and computer vision topics. Apply Mediapipe framework to real life application.

1. STUDY CONTENTS

- **Introduction**: An overview of the thesis: an introduction to the topic, research methods and layout of the thesis.

- **Content part**: The content of the thesis is divided into 3 chapters

+ Chapter 1: Required specifications

+ Chapter 2: An overview of technology used

+ Chapter 3: System descriptions

+ Chapter 4: Delivery system design

+ Chapter 5: Testing and experimental results

- **Conclusion**: Present the results achieved and the development direction of the system

CONTENTS

1. AN OVERVIEW OF TECHNOLOGY AND LIBRARY USED

1. COMPUTER VISION

1.1. Definition

Computer vision is the field of computer science that focuses on creating digital systems that can process, analyze, and make sense of visual data (images or videos) in the same way that humans do. The concept of computer vision is based on teaching computers to process an image at a pixel level and understand it. Technically, machines attempt to retrieve visual information, handle it, and interpret results through special software algorithms.

Computer vision trains machines to perform these functions, but it has to do it in much less time with cameras, data and algorithms rather than retinas, optic nerves and a visual cortex. Because a system trained to inspect products or watch a production asset can analyze thousands of products or processes a minute, noticing imperceptible defects or issues, it can quickly surpass human capabilities.

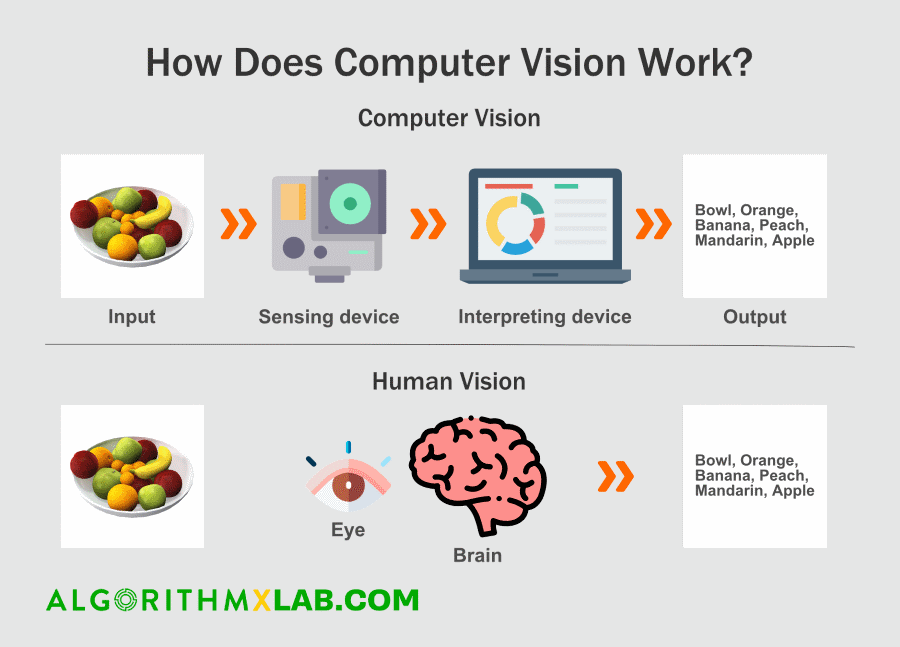


Figure 1: Computer Vision example

Computer vision needs lots of data. It runs analyses of data over and over until it discerns distinctions and ultimately recognize images. For example, to train a computer to recognize automobile tires, it needs to be fed vast quantities of tire images and tire-related items to learn the differences and recognize a tire, especially one with no defects.

1.2. Computer vision common tasks and real-life applications

In essence, computer vision tasks are about making computers understand digital images as well as visual data from the real world. This can involve extracting, processing, and analyzing information from such inputs to make decisions. The most popular computer vision tasks that we regularly find in AI jargon include:

* *Image Classification*: Given a group of images, the task is to classify them into a set of predefined classes using solely a set of sample images that have already been classified.
* *Object detection*: refers to detection and localization of objects using bounding boxes. Object detection looks for class-specific details in an image or a video and identifies them whenever they appear.
* *Image segmentation*: the division of an image into subparts or sub-objects to demonstrate that the machine can discern an object from the background and/or another object in the same image.
* *Face and person recognition*: where features are detected and localized, facial recognition performs not only detection, but also recognition of the detected face.
* *Video motion analysis*: refers to the study of moving objects or animals and the trajectory of their bodies.

There is a lot of research being done in the computer vision field, but it’s not just research. Real-world applications demonstrate how important computer vision is to endeavors in business, entertainment, transportation, healthcare and everyday life. A key driver for the growth of these applications is the flood of visual information flowing from smartphones, security systems, traffic cameras and other visually instrumented devices.

* Google Translate lets users point a smartphone camera at a sign in another language.
* IBM is applying computer vision technology with partners like Verizon to bring intelligent AI to the edge, and to help automotive manufacturers identify quality defects before a vehicle leaves the factory.
* The development of self-driving vehicles relies on computer vision to make sense of the visual input from a car’s cameras and other sensors. It’s essential to identify other cars, traffic signs, lane markers, pedestrians, bicycles and all of the other visual information encountered on the road.



Figure 2: Self-driving car using computer vision

2. MEDIAPIPE FRAMEWORK

2.1. Definition

MediaPipe is an open-source framework for building pipelines to perform computer vision inference over arbitrary sensory data such as video or audio. Using MediaPipe, such a perception pipeline can be built as a graph of modular components. MediaPipe was built for machine learning (ML) teams and software developers who implement production-ready ML applications, or students and researchers who publish code and prototypes as part of their research work.

In computer vision pipelines, those components include model inference, media processing algorithms, data transformations, etc. Sensory data such as video streams enter the graph, and perceived descriptions such as object-localization or face-keypoint streams exit the graph.

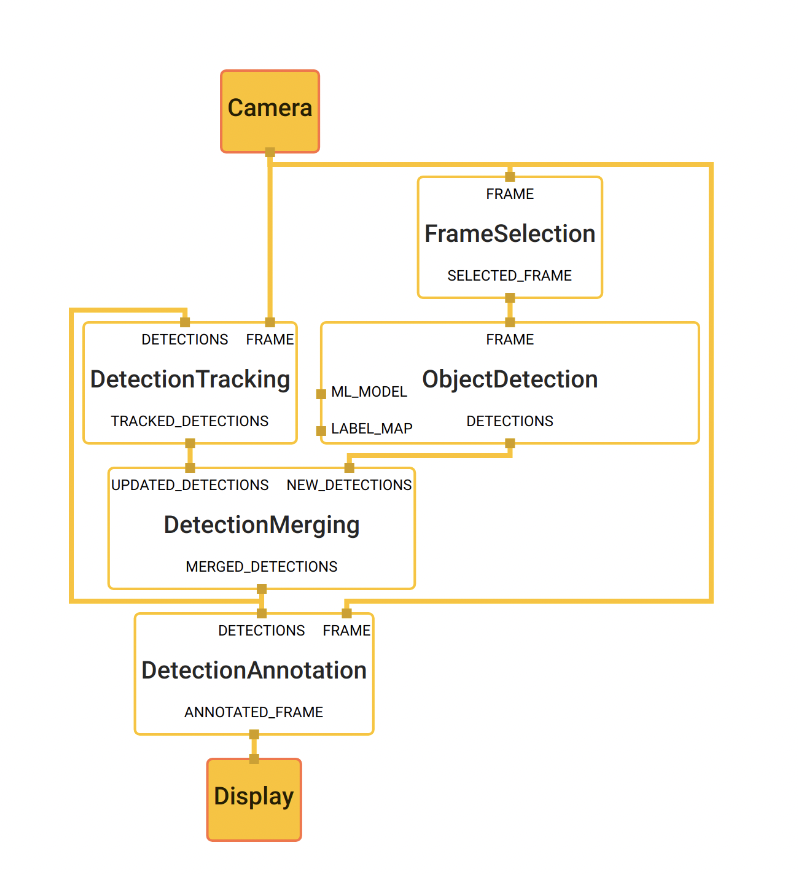


Figure 3: Object detection pipeline in MediaPipe

The main use case for MediaPipe is rapid prototyping of perception pipelines with inference models and other reusable components. MediaPipe also facilitates the deployment of perception technology into demos and applications on a wide variety of different hardware platforms. MediaPipe enables incremental improvements to perception pipelines through its rich configuration language and evaluation tools.

2.2. MediaPipe Architecture

MediaPipe allows a developer to prototype a pipeline incrementally. A pipeline is defined directed graph of components where each component is a *Calculator*. The graph is specified using a *GraphConfig* *protocol* buffer and then run using a Graph object.

In the graph, the calculators are by data, *Streams*. Each represents a time-series of data, *Packets*. Together, the calculators and streams define a data-flow graph. The packets which flow across the graph are collated by their timestamps within the time-series.

* *Packet*: basic data unit in MediaPipe. A packet consists of a numeric timestamp and a shared pointer to an immutable payload.
* *Stream*: each node in the graph is connected to another node through a stream. A stream carries a sequence of packet whose timestamps must be monotonically increasing.
* *Calculators*: implemented as each node of a graph. The bulk of graph execution happens inside its calculator.
* *Graph*: all processing takes places within the context of a Graph. A graph contains a collection of nodes joined by directed connections along which packets can flow.
* *GraphConfig*: is a specification that describes the topology and functionality of a MediaPipe graph. All the necessary configurations of the node, such its type, inputs and outputs must be described in the specification. Description of the node can also include several optional fields, such as node-specific options, input policy and executor.

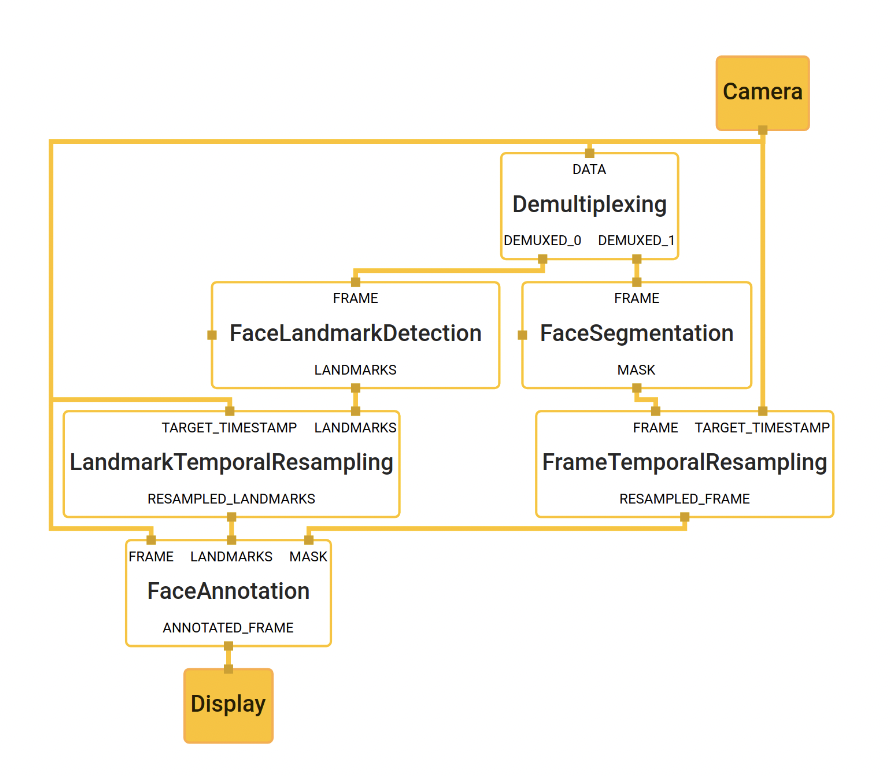


Figure 4: Face landmark detection and segmentation using MediaPipe

2.3. Advantages of MediaPipe

* *End-to-end acceleration*: Use common hardware to build-in fast ML inference and video processing, including GPU, CPU, or TPU.
* *Build once, deploy anywhere*: The unified framework is suitable for Android, iOS, desktop, cloud, web, and IoT platforms.
* *Ready-to-use solutions*: Prebuilt ML solutions demonstrate the full power of the MediaPipe framework.
* *Open source and free*: The framework is licensed under Apache 2.0, fully extensible, and customizable.

3. MEDIAPIPE POSE

3.1. Overview and Machine Learning pipeline

MediaPipe Pose is a ML solution for high-fidelity body pose tracking, inferring 33 3D landmarks and background segmentation mask on the whole body from RGB video frames utilizing our BlazePose research that also powers the ML Kit Pose Detection API. Current state-of-the-art approaches rely primarily on powerful desktop environments for inference, whereas our method achieves real-time performance on most modern mobile phones, desktops/laptops, in python and even on the web.

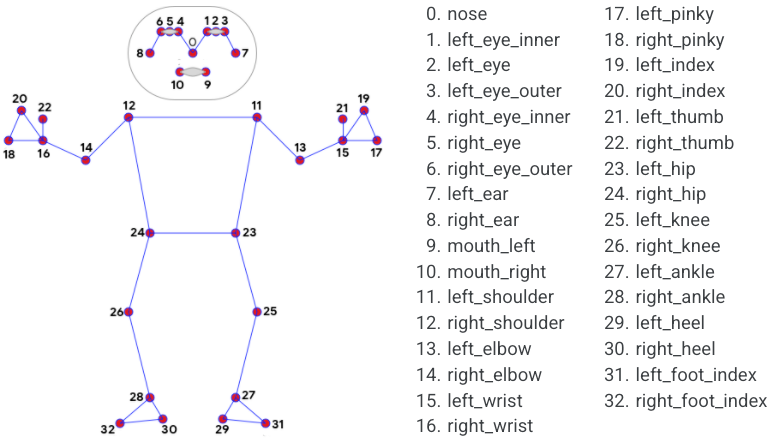


Figure 5: The 33 landmarks model in MediaPipe Pose predicts

The solution utilizes a two-step detector-tracker ML pipeline, proven to be effective in MediaPipe Hands and MediaPipe Face Mesh solutions. Using a detector, the pipeline first locates the person/pose region-of-interest (ROI) within the frame. The tracker subsequently predicts the pose landmarks and segmentation mask within the ROI using the ROI-cropped frame as input. Note that for video use cases the detector is invoked only as needed, i.e., for the very first frame and when the tracker could no longer identify body pose presence in the previous frame. For other frames the pipeline simply derives the ROI from the previous frame’s pose landmarks.

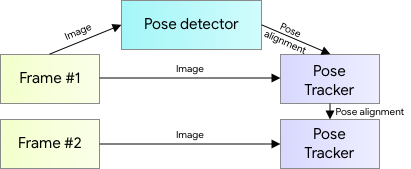


Figure 6: MediaPipe pose detection's pipeline

3.2. BlazePose detector

For real-time performance of the full ML pipeline consisting of pose detection and tracking models, each component must be very fast, using only a few milliseconds per frame. To accomplish this, it is known that the strongest signal to the neural network about the position of the torso is the person's face (due to its high-contrast features and comparably small variations in appearance). Therefore, BlazePose is a fast and lightweight pose detector by making the strong (yet for many mobile and web applications valid) assumption that the head should be visible for single-person use case.

Therefore, the detector is inspired by the lightweight BlazeFace model, used in MediaPipe Face Detection, as a proxy for a person detector. It explicitly predicts two additional virtual keypoints that firmly describe the human body center, rotation and scale as a circle. Inspired by Leonardo’s Vitruvian man, the model predicts the midpoint of a person’s hips, the radius of a circle circumscribing the whole person, and the incline angle of the line connecting the shoulder and hip midpoints.

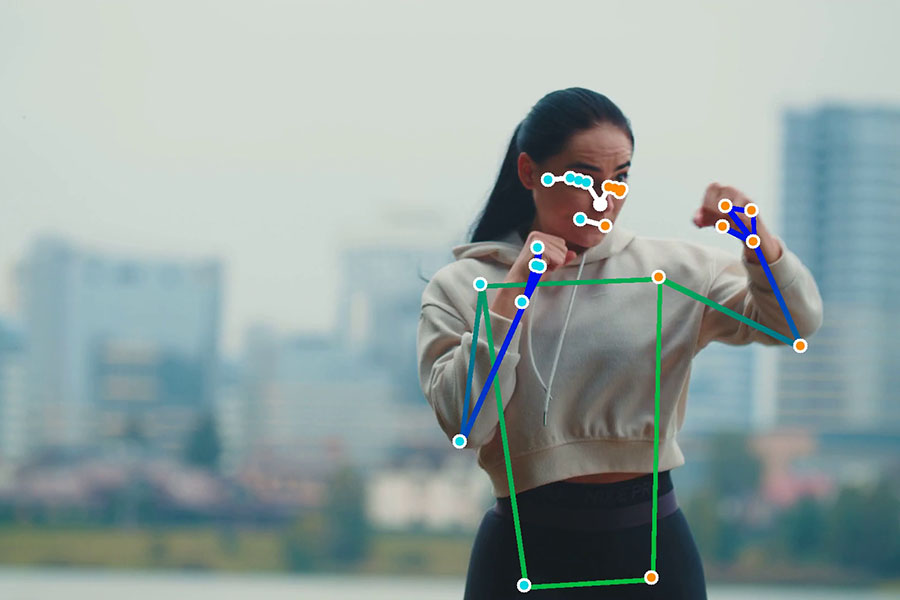


Figure 7: Example of pose detection by MediaPipe

3.3. MediaPipe Pose Output

MediaPipe Pose returns a list of 33 pose landmarks. Each landmark consists of the following properties:

* *x* and *y*: Landmark coordinates normalized to [0.0, 1.0] by the image width and height respectively.
* *z:* Represents the landmark depth with the depth at the midpoint of hips being the origin, and the smaller the value the closer the landmark is to the camera. The magnitude of *z* uses roughly the same scale as *x*.
* *visibility*: A value in *[0.0, 1.0]* indicating the likelihood of the landmark being visible (present and not occluded) in the image.

4. OTHER TECHNOLOGY AND LIBRARIES USED

4.1. Open CV

4.2. Scikit-learn

4.3. Keras

4.4. Vue.js

4.5. Django

1. GENRAL APPROACH ON BUILD MACHINE LEARNING MODELS

1. EXERCISES SELECTION AND ERRORS DETERMINATION

First and foremost, the initial step of this thesis is to decide on which exercises to train machine learning model for form correction. Due to the limitation of time for the thesis, only 4 exercises are picked. Each exercise needs to meet the following criteria:

* An exercise must be a common exercise.
* An exercise is popular among people who workouts at home or workouts mostly by themselves. Therefore, it would raise the possibility that the exercise would be performed improperly.
* An exercise must contain at least 2 popular errors which impact the inefficiency of the exercise.

Depends on the above criteria, I decided on 4 exercises: *bicep curl*, *basic plank*, *basic quat* and *lunge*.



Figure 8: A person doing lunge

After decided on 4 exercises, for each exercise, at least 2 popular errors for that exercise would be determined and then develop strategy to detect those errors (The detail of each error will be going in depth in this section). Here are the determined common errors for each exercise:

* *Bicep Curl*: Loose upper arm, weak peak contraction and lean-back standing posture.
* *Basic plank*: Lower back and High back.
* *Basic squat*: Foot placement too tight/wide and knee placement too wide/tight.
* *Lunge*: Knee over toe while going down.

2. DATA COLLECTING

To be able to build exercise correction machine learnings model for this thesis, for each exercise, there are steps to be followed: collecting data (online videos, self-made video, online dataset), processing data depends on the exercise, training and evaluating model.

2.1. Self-collected data

Due to the lack of videos or dataset online that recorded human doing exercises both in a proper or improper way, the majority of self-collected videos were either recorded by myself, my friends or my family. In total, there are 5 contributors for this dataset. For each exercise, each contributor is required to:

* Record 2 videos with different camera angles when the contributor performs the exercise correctly.
* Record 2 videos with different camera angles for each determined error of the exercises.

For each video that provided by each contributor, the video has to meet these following criteria:

* Minimum duration: depends on exercise. With exercises which does not require much action such as plank, the minimum required duration of a video would be 30 seconds. However, with exercises which involve more movements such as squat, the contributors are required to perform the exercise at least 15 reps for each video. (More detail in …)
* Video is recorded in environment with enough light so that the whole body of a person is in good visibility.
* For each exercise, video needs to show clearly the important joints or movement which important to the exercise.
* The contributor needs to be on frame throughout the video.

After collecting the videos, depends on the exercise, they are divided into multiple classes:

* “proper-form”: Videos of a contributor perform the exercise correctly.
* “error-name”: Each video of a contributor performs the exercise correctly will be in a separate class.

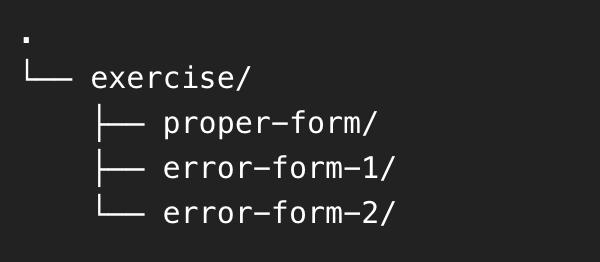


Figure 9: Example of a directory tree structure for each exercise collected videos

2.2. Public Dataset for Kaggle

With an exercise such as *Plank*, as there is not much movement during the exercise, I’m able to find a dataset from an open database from Kaggle. In short, Kaggle is an online community platform for data scientists and machine learning enthusiasts.

The found dataset is about many yoga poses but the very well-known ones are the downward dog pose, goddess pose, tree pose, plank pose and the warrior pose. The dataset contains 5 folders for 5 poses, each folder contains images of people correctly doing the correspond pose.

For the purpose of this thesis, only the folder contains the images of people properly doing plank is chosen. There are 266 image files in that folder, I handpicked all the images that represent a basic plank and discard the reset. In conclusion, there are 30 images which are arranged to the proper form class for *basic plank*.

3. DATA PROCESSING

In general, the operation of processing the self-collected video is to identify human pose with MediaPipe Pose and then store data in an *.csv* file. The operation is particular depends on which exercise, however, there are a few basic steps which are handled the same throughout all 4 exercises, this section will discuss about these steps.

3.1. Detecting important landmarks

For each exercise, there will be different poses/body’s position, therefore it is essential to identify which parts (shoulder, hip, …) of a body are contribute to the exercise. The important landmarks identified for each exercise are utilized to extract body part’s position while exercising using MediaPipe.

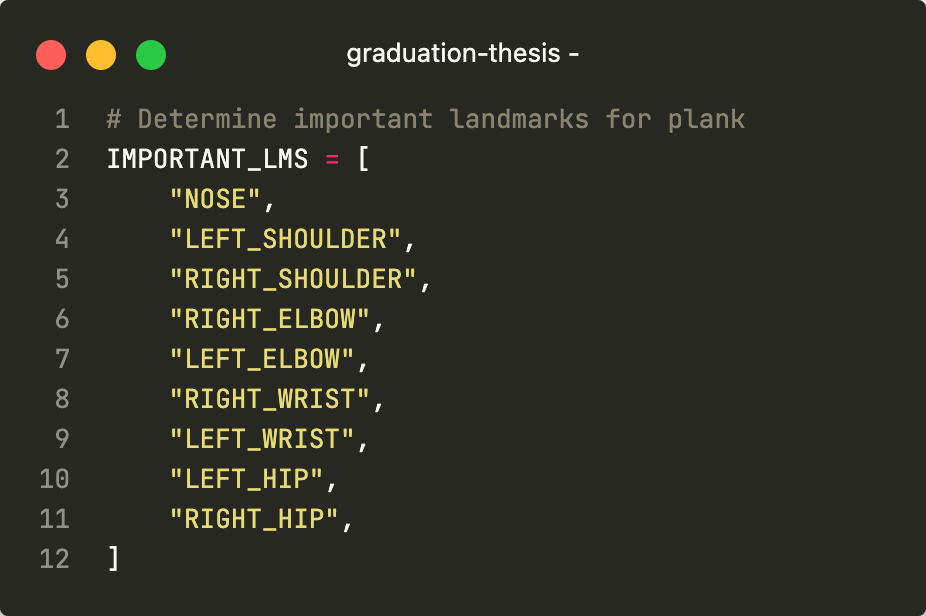


Figure 10: Example of important landmarks for plank exercise

In addition, every landmark’s property such as *x, y, z* and *visibility*, will be flatten to 4 headers for an *csv* file. For example, with the *NOSE* landmarks, the 4 headers would be: NOSE\_x, NOSE\_y, NOSE\_z and NOSE\_v(detail mentioned in page 18). Therefore, for each exercise, an empty *csv* file will be initialized. The first header is the “label” which contain a class for each datapoint, the rest of file’s headers are all the important headers with their properties flatten.

label,nose\_x,nose\_y,nose\_z,nose\_v

C,0.549,0.32,-0.175,0.992

L,0.538,0.301,-0.178,0.994

3.2. Extract data from video using OpenCV and MediaPipe

With every exercise, there are videos which are separated into different classes. With each video, every frame that matches a certain form of an exercise would be:

* Pose estimation: Process the video using OpenCV then utilize MediaPipe Pose to produce a list of coordinate predictions of all keypoint locations, and their corresponding prediction confidence.

Configuration options for Medipipe Pose:

* + STATIC\_IMAGE\_MODE: If set to false, the solution treats the input images as a video stream. It will try to detect the most prominent person in the very first images, and upon a successful detection further localizes the pose landmarks. In subsequent images, it then simply tracks those landmarks without invoking another detection until it loses track, on reducing computation and latency. If set to true, person detection runs every input image, ideal for processing a batch of static, possibly unrelated, images. Default to false.
  + MODEL\_COMPLEXITY: Complexity of the pose landmark model: 0, 1 or 2. Landmark accuracy as well as inference latency generally go up with the model complexity. Default to 1.
  + SMOOTH\_LANDMARKS: If set to true, the solution filters pose landmarks across different input images to reduce jitter, but ignored if static\_image\_mode is also set to true. Default to true.
  + ENABLE\_SEGMENTATION: If set to true, in addition to the pose landmarks the solution also generates the segmentation mask. Default to false.
  + SMOOTH\_SEGMENTATION: If set to true, the solution filters segmentation masks across different input images to reduce jitter. Ignored if enable\_segmentation is false or static\_image\_mode is true. Default to true.
  + MIN\_DETECTION\_CONFIDENCE: Minimum confidence value ([0.0, 1.0]) from the person-detection model for the detection to be considered successful. Default to 0.5.
  + MIN\_TRACKING\_CONFIDENCE: Minimum confidence value ([0.0, 1.0]) from the landmark-tracking model for the pose landmarks to be considered tracked successfully, or otherwise person detection will be invoked automatically on the next input image. Setting it to a higher value can increase robustness of the solution, at the expense of a higher latency. Ignored if static\_image\_mode is true, where person detection simply runs on every image. Default to 0.5.

cap = cv2.VideoCapture("/path/to/video")

with mp\_pose.Pose(min\_detection\_confidence=0.5, min\_tracking\_confidence=0.5) as pose:

while cap.isOpened():

ret, image = cap.read()

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

results = pose.process(image)

* For every frame with the list of predicted landmarks, the important landmarks for a correspond exercise would be extracted in append as a row to the csv file with the label column matches the class of the video.
* The collected data which is saved in a csv file would be split to train and evaluate model. 80% of the data will be used for model training and the remaining 20% will be used for model evaluation.

4. MODEL TRAINING

There are 2 methods used in this thesis for model training. For each exercise, the models trained for each method will be compared and the best model will be chosen.

* Classification with Scikit-learn.
* Building a Neural Network for classification with Keras.

4.1. Classification with Scikit-learn

The machine learning algorithms used for model training are: Decision Tree/Random Forest, K-Nearest Neighbors (KNN), C-Support Vector (SVC), Linear classifiers (SVM, logistic regression, etc.) with SGD training and Classifier using Ridge regression.

After the training session, each result from each algorithm will be evaluate with metrics such as: precision score, accuracy score and F1 score. The algorithm which provides the best results will be selected. Below is the sample code for this section.

algorithms = [("LR", LogisticRegression()),

("SVC", SVC()),

('KNN',KNeighborsClassifier()),

("DTC", DecisionTreeClassifier()),

("SGDC", SGDClassifier()),

("Ridge", RidgeClassifier()),

('RF', RandomForestClassifier()),]

models = {}

for name, model in algorithms:

trained\_model = model.fit(X\_train, y\_train)

models[name] = trained\_model

model\_results = model.predict(X\_test)

p\_score = precision\_score(y\_test, model\_results,

average="macro")

a\_score = accuracy\_score(y\_test, model\_results)

f1\_score\_result = f1\_score(y\_test, model\_results,

average=None, labels=["C", "L", "H"])

4.2. Neural Network for classification with Keras

With Keras, composing a neural network by creating layers and linking them together. In this project, one type of layer called a fully connected or Dense layer. In Keras, this is defined by the keras.layers.Dense class.

A dense layer has a number of neurons, which is a parameter which can be chosen when the layer is created. When connecting the layer to its input and output layers every neuron in the dense layer gets an edge (i.e. connection) to all of the input neurons and all of the output neurons.

Occasionally, I used another type of layer which is Dropout layer. The term “dropout” refers to dropping out the nodes (input and hidden layer) in a neural network. The nodes are dropped by a dropout probability of p. Dropout layer is used to combat the problem of overfitting. In overfitting, a unit may change in a way that fixes up the mistakes of the other units, using dropout, it prevents these units to fix up the mistake of other units, thus preventing co-adaptation, as in every iteration the presence of a unit is highly unreliable. Therefore, by randomly dropping a few units (nodes), it forces the layers to take more or less responsibility for the input by taking a probabilistic approach.

One of a parameter for a hidden layer is activation function. In short, activation function in the hidden layer will control how well the network model learns the training dataset. The choice of activation function in the output layer will define the type of predictions the model can make. In this thesis, there are only 2 activation functions which is used:

* relu function: Applies the rectified linear unit activation function. With default values, this returns the standard ReLU activation: max(x, 0), the element-wise maximum of 0 and the input tensor..
* Softmax function: converts a vector of values to a probability distribution. The elements of the output vector are in range (0, 1) and sum to 1.

Below is a sample code for an neural network built with Keras.

model = Sequential()

model.add(Dense(68, input\_dim = 68, activation = "relu")) model.add(Dropout(0.5))

model.add(Dense(68, activation = "relu")) model.add(Dropout(0.5))

model.add(Dense(14, activation = "relu"))

model.add(Dense(3, activation = "softmax")) model.compile(Adam(lr = 0.01), "categorical\_crossentropy", metrics = ["accuracy"])

model.summary()

1. DELIVERY SYSTEM DESIGN

1. DATABASE DESIGN:

The below image represents how the database of delivery system is designed. There are 6 tables, the details of every table will be described from the tables below.

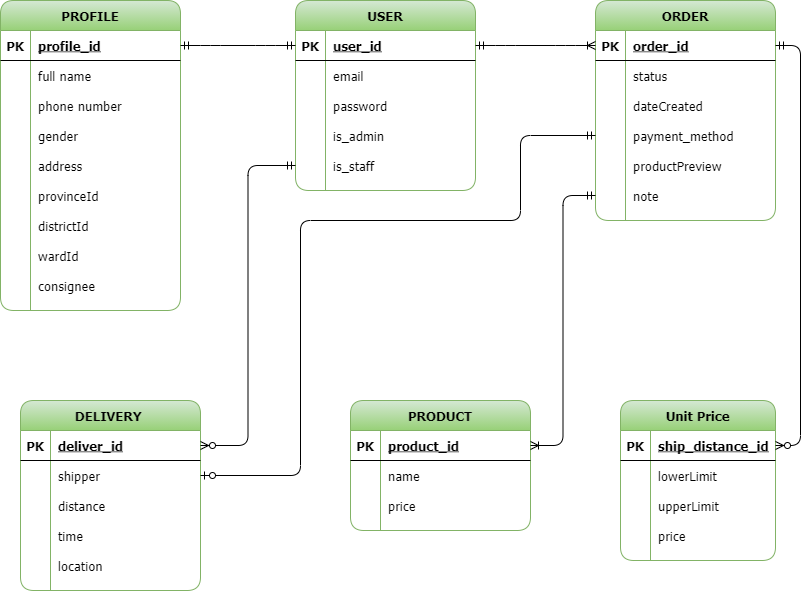


Figure 11: Database diagram of the system

The User table contains private and identity information about an account.

| **No** | **Attribute** | **Data type** | **Primary**  **Key** | **Unique** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | user\_id | int | x | x | User identification |
| 2 | email | varchar |  | x | Account email for login and reset password |
| 3 | password | varchar |  |  | Account password for authentication |
| 4 | is\_admin | boolean |  |  | To identify if account is an admin (manager) |
| 5 | is\_staff | boolean |  |  | To identify if account is an employee of the system (driver or manager) |

Table 9: Database design - User table

The Profile table contains personal information about an account or a consignee that have used the system.

| **No** | **Attribute** | **Data type** | **Primary**  **Key** | **Unique** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | profile\_id | int | x | x | Profile identification |
| 2 | full\_name | varchar |  |  | Account’s owner full name |
| 3 | phone | varchar |  |  | Account’s owner phone number |
| 4 | gender | boolean |  |  | Account’s owner gender |
| 5 | address | boolean |  |  | Account’s owner detail address, include house number plates and street names |
| 6 | province\_id | int |  |  | Vietnam’s province code |
| 7 | district\_id | int |  |  | Vietnam’s district code |
| 8 | ward\_id | int |  |  | Vietnam’s ward/subdistrict code |
| 9 | consignee | boolean |  |  | Check if this profile belong to a consignee of an order |

Table 10: Database design - Profile table

The Order table holds information of an order.

| **No** | **Attribute** | **Data type** | **Primary**  **Key** | **Unique** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | order\_id | int | x | x | Order identification |
| 2 | status | varchar |  |  | Status of an order: processing (a consignor just places an order), delivering (the order are being delivered), delivered (the order’s successfully delivered) and failed (fail to deliver) |
| 3 | paymentMethod | int |  |  | How customers choose to pay for an order:  Pay by consignor  Pay by consignee |
| 4 | productPreview | int |  |  | Restriction when a consignee receives an order.  Consignee does not allow to observe the product.  Consignee allows to observe but not to try the product.  Consignee allows to try the product. |
| 5 | note | text |  |  | Any note from consignor for the driver or consignee. |
| 6 | dateCreated | datetime |  |  | Date and time when the order is created |

Table 11: Database design - Order table

The Unit Price table contains information about the price of an order depends on the distance to deliver.

| **No** | **Attribute** | **Data type** | **Primary**  **Key** | **Unique** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | unit\_id | int | x | x | Unit price identification |
| 2 | lower\_limit | int |  | x | If the distance is greater than this attribute, the price of the order will be this unit price.  Otherwise, the price will belong to other unit price |
| 3 | upper\_limit | int |  | x | If the distance is lower than this attribute, the price of the order will be this unit price.  Otherwise, the price will belong to other unit price |
| 4 | price | int |  |  | The price corresponds to the 2 limits (Vietnam Dong) |

Table 12: Database design - Unit price table

The Product table holds information about each product from the package of an order.

| **No** | **Attribute** | **Data type** | **Primary**  **Key** | **Unique** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | product\_id | int | x | x | Unit price identification |
| 2 | name | varchar |  |  | Name of a product |
| 3 | price | int |  |  | Price of a product |

Table 13: Database design - Product table

The Delivery table holds information about the delivery process of a delivering order.

| **No** | **Attribute** | **Data type** | **Primary**  **Key** | **Unique** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | deliver\_id | int | x | x | Delivery process identification |
| 2 | distance | float |  |  | The calculated distance has to travel to deliver an order when using Leaflet-Routing-Machine API (meter) |
| 3 | time | float |  |  | The calculated time has to travel to deliver an order when using Leaflet-Routing-Machine API (second) |
| 4 | location | varchar |  |  | The current location of an order, update according to the driver location. This attribute is saved as a JSON string format. |

Table 14: Database design - Delivery table

2. FINDING AND NAVIGATING BETWEEN LOCATIONS

One of the most important features of the system is calculate the distance between the consignee and the company location to announce the delivery price for the customer. Therefore, the system has to be able to identify the geolocation (latitude and longitude) of the consignee address. Besides pricing, determine consignee geolocation also helps with routing and navigating for easier delivery of an order and tracking its location.

2.1. Finding consignee location

The process of finding consignee location will be happened once per order, and it is start when the consignor submits their order placement in the Order Placement page.

The tools I used for finding location is Open Route Services API. It is a free API service that provide API endpoints for direction, finding geolocation, … In this system, I use it to find the geolocation of the consignee.

By feeding the Open Route Services API the address as a string format, the service will return a geolocation of that address if it is found by the service. Throughout out the process of using and testing this service, I realize that the service works at its best only when the address does not contain specific information such as house number plates or street names. Therefore, an address will contain only the ward, the district and the province/city.

The data returns from the API will be in a JSON format of an array. That array contains all the results that the API found about that address. For each element in the array, there are latitude and longitude of a location. In this case, I will choose the first element from the array as the position for the address as I found that it is the most reliable element.

Example of an address for API call: “Hưng Lợi, Cái Răng, Cần Thơ”

Example of a JavaScript code making POST request using axios library to the Open Routes Services API:

const url = "https://nominatim.openstreetmap.org/search?format=json&limit=3&q=" + encodeURIComponent(‘Hưng Lợi, Cái Răng, Cần Thơ’);

const { data } = await axios.get(url);

if (data.length > 0) {

const { lat, lon } = data[0];

}

Example of a geolocation of an address:

const data = {

lat: 10.01792665,

lon: 105.78839074112506

}

2.2. Navigating and distance calculating for order delivery

Leaflet Routing Machine API is the tool that I use for navigate and calculate the distance between the consignee and the company location.

The inputs for the API are the geolocation (latitude and longitude) of a consignee’s location and the company’s location.

The data returns from the API is packed with a lot of information such as the routes found from the two locations, the estimated travel distance (in meter), estimated time to travel (in second), the instructions to navigate to consignee locations ....

Example of inputs data for the API calls:

const data = {

lat: 10.01792665,

lon: 105.78839074112506

}

Example of using Leaflet Routing Machine API to navigate and calculate distance for the order to the consignee’s location

const routing = L.Routing.control({

waypoints: [

L.latLng({ lat: 10.001, lon: 105.123 }),

L.latLng({ lat: 10.001, lon: 105.567 }),

],

…

})

Example of a data that return from the API:

const data = {

summary: {

totalDistance: 63742.4,

totalTime: 3730.5,

},

instructions: [

{

directions: ‘S’,

distance: 91.1,

road: “Đường B14”,

}

…

],

…

}

3. USER INTERFACE DESIGN

The delivery system’s user interface is designed to be simple, clean and friendly for any kind of users to operate.

3.1. Login/Register page

Register page is for users to register an account to use the delivery system. Login page is for both user and manager of the system to login.

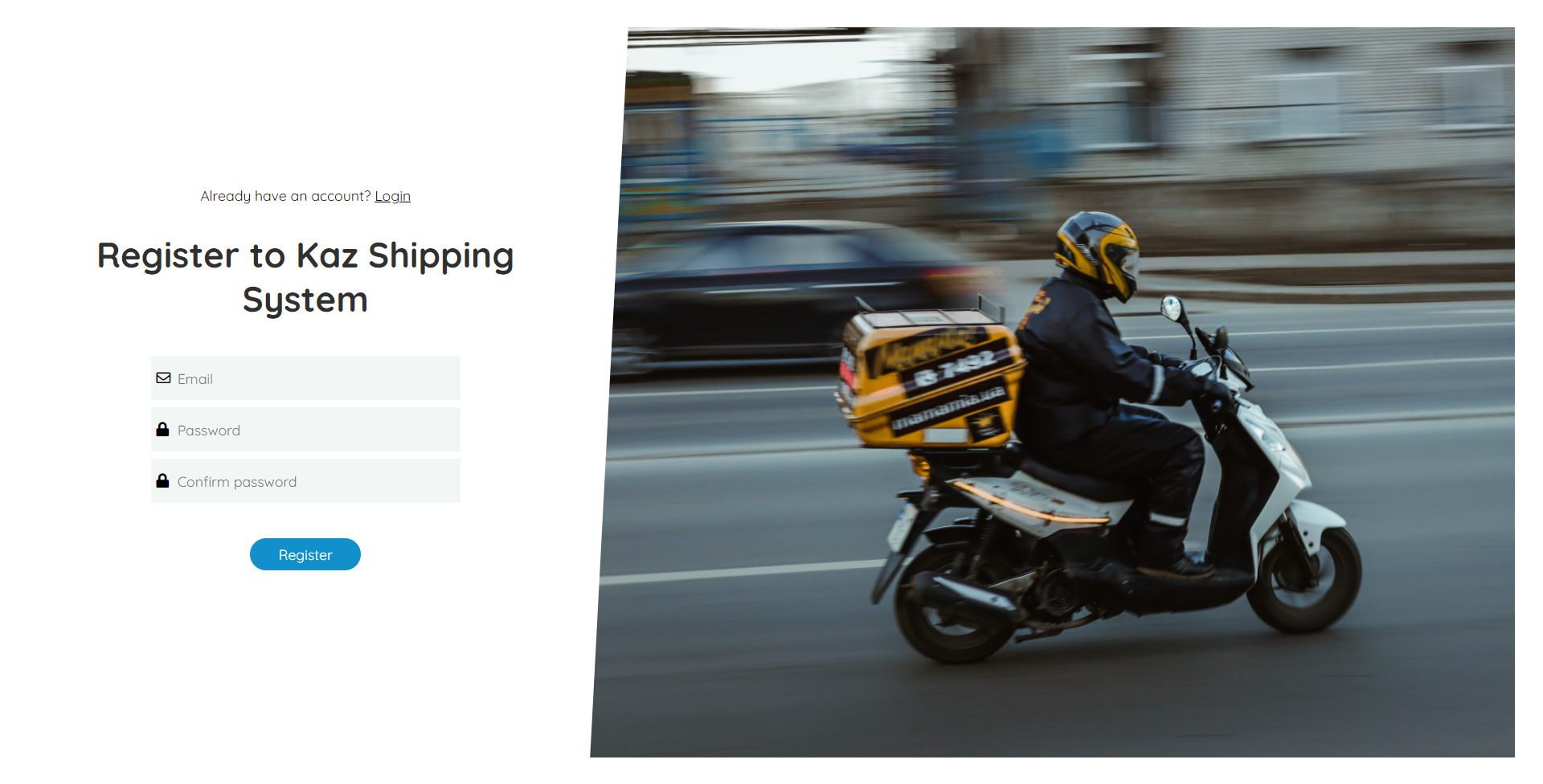


Figure 12: System Screenshot - Register page

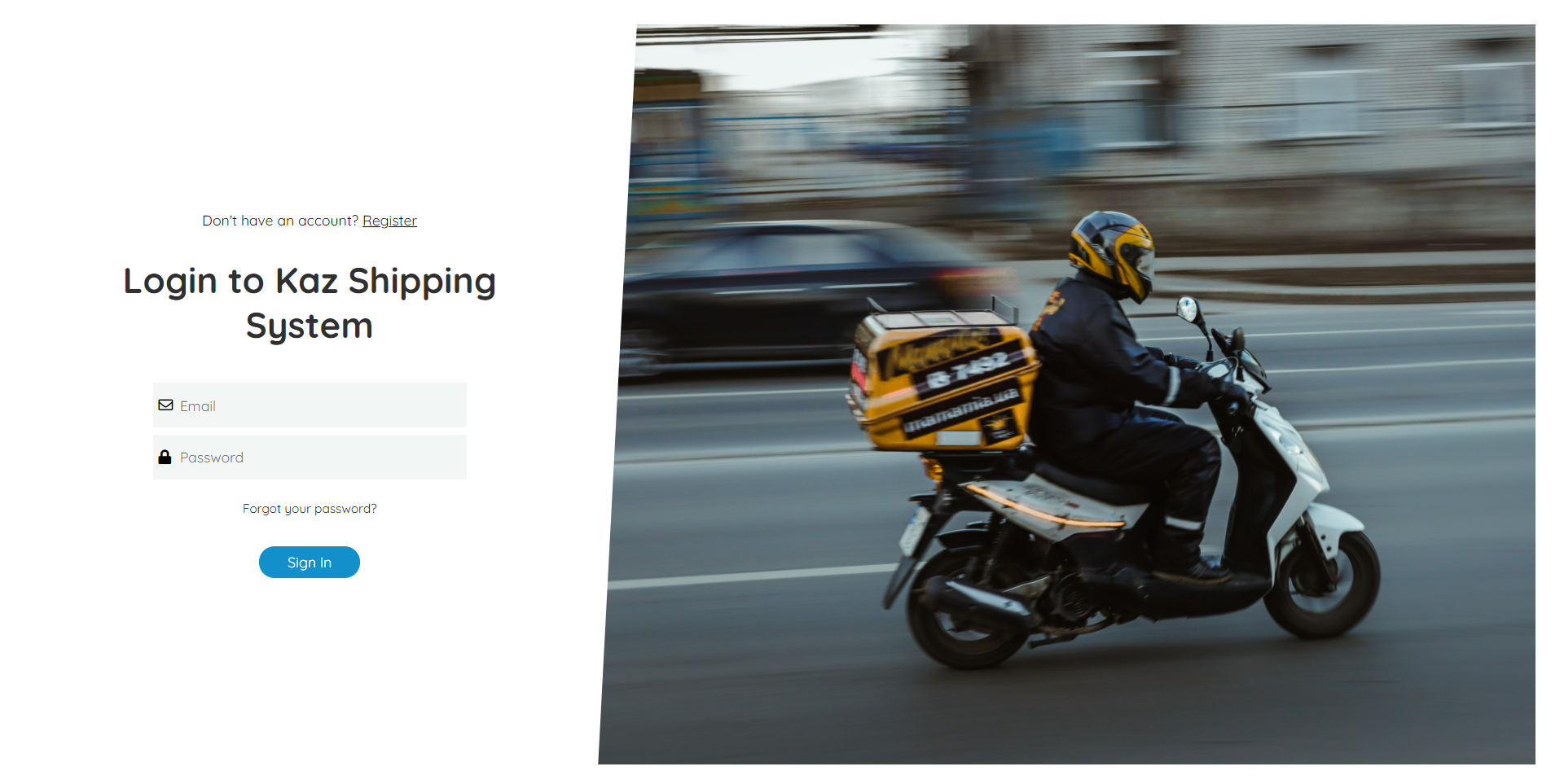


Figure 13: System Screenshot - Login page

3.2. Orders Management page

Orders management page helps users/manager to manage their orders. Managers can manage all the orders that are registered in the system. Users can only manage that they made.

Orders management page provides a filter tool for users to easily manage their orders. Users can filter by the orders’ status (processing, delivering, delivered or failed), the payment method of the orders (paid by consignee or by consignor) and the period of time that the orders had been made.

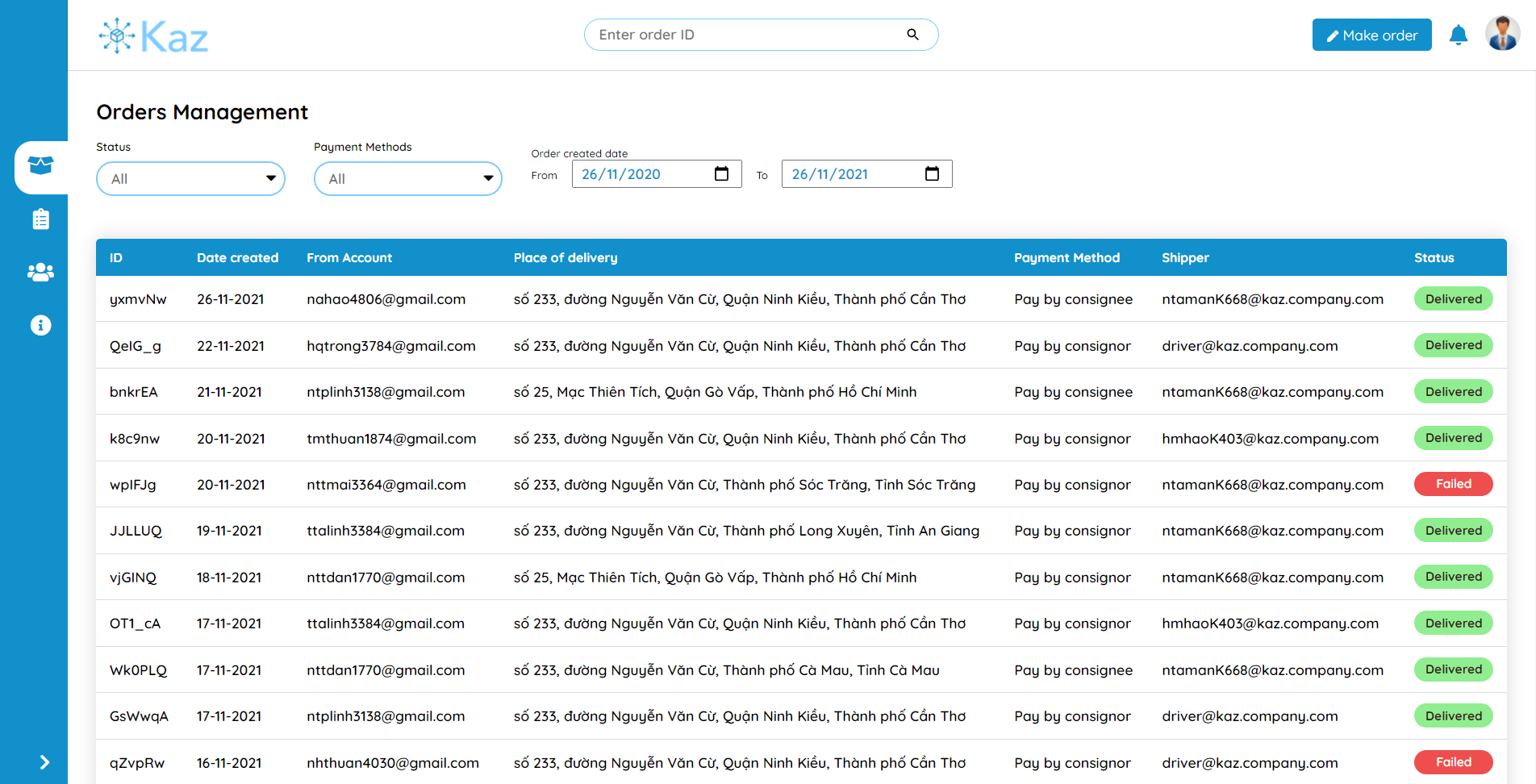


Figure 14: System Screenshot - Order management page

3.3. Employee Management page

The only-manager access page for manager to employees and their customers. There are 3 tabs (employee, consignor and consignee) for the managers to toggle, each tab will display the information correspond to its name.

For each employee, if it is a manager account, the other manager can click on the “Profile” button to view his/her profile which contains personal and work related information (name, phone number, …). Otherwise, if it is a driver account, the manager can view not only the account’s profile but also the orders which are delivering or delivered by him and the manager can also assign processing orders for that driver account by click on the “Orders” button.

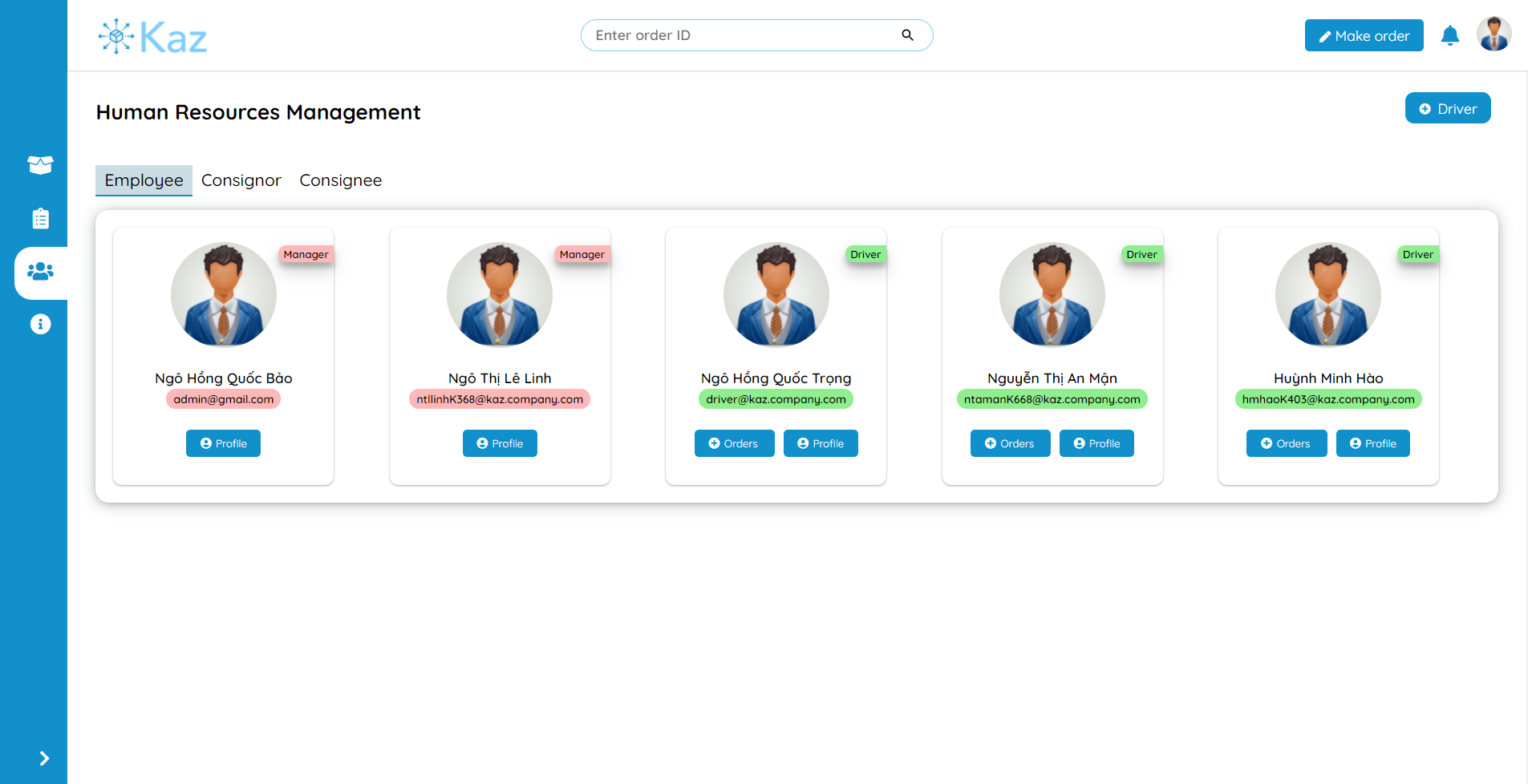


Figure 15: System Screenshot - Employee management

For the consignor tabs, the system will display a table of users that registered their account on the system. The manager can click on the row that corresponding to an account to view his/her information and the orders that he/she has placed.

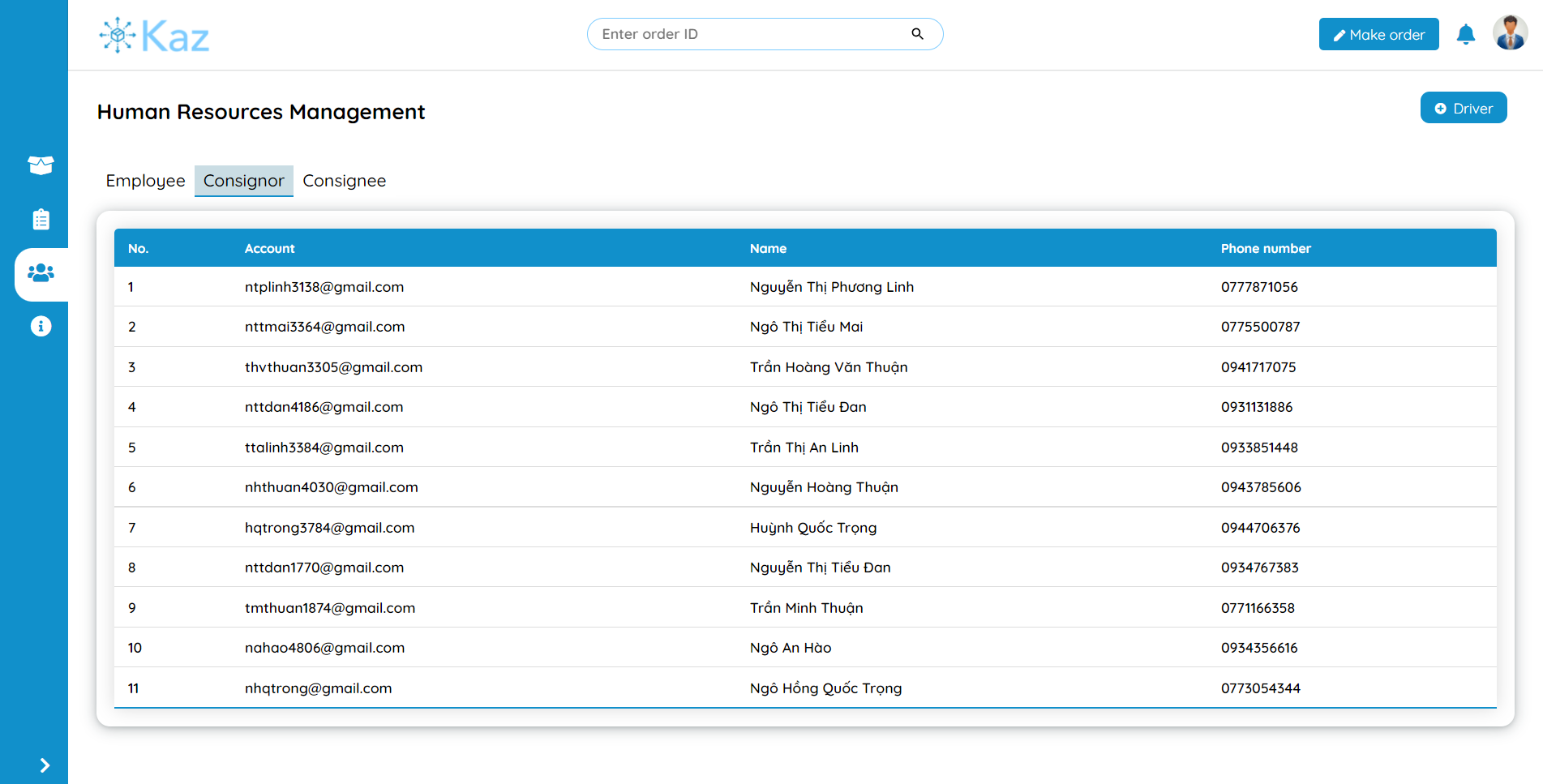


Figure 16: System Screenshot - Consignor management page

For the consignee tabs, the system will also display a table of consignee whose information has been recorded when a consignor account makes an order. Manager can click to that the row that that corresponding to a consignee to view all the information about that consignee with that his/her order.

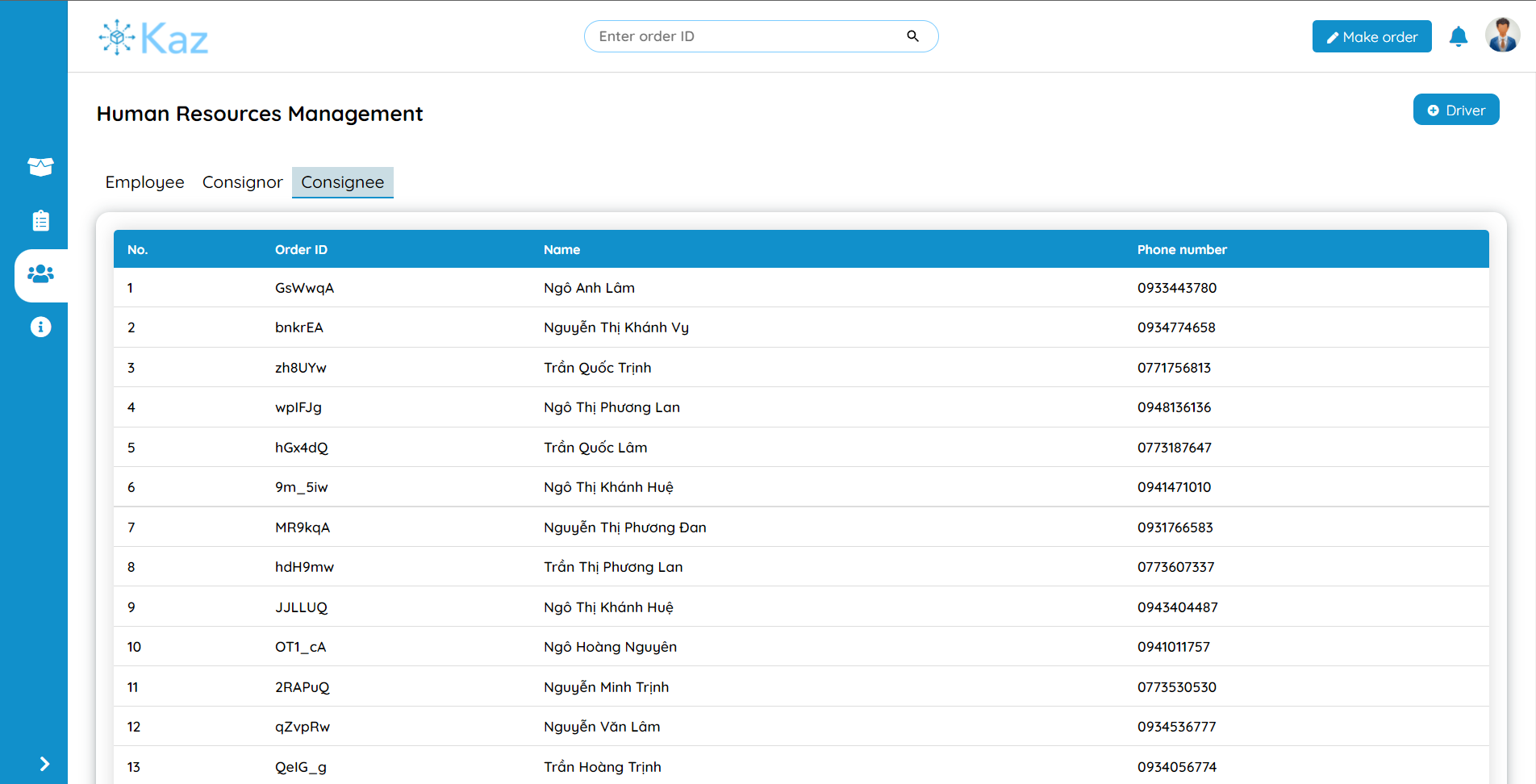


Figure 17: System Screenshot - Consignee management page

3.4. Profile page

Profile page displays the personal and work related information of an account. The user can use to the profile page to edit their own information. There is also an “Orders” tab that displays all the orders that an account has made.

If a manager account visits the “Profile” page of a driver account, the manager can also have a tab to assign orders to a driver.

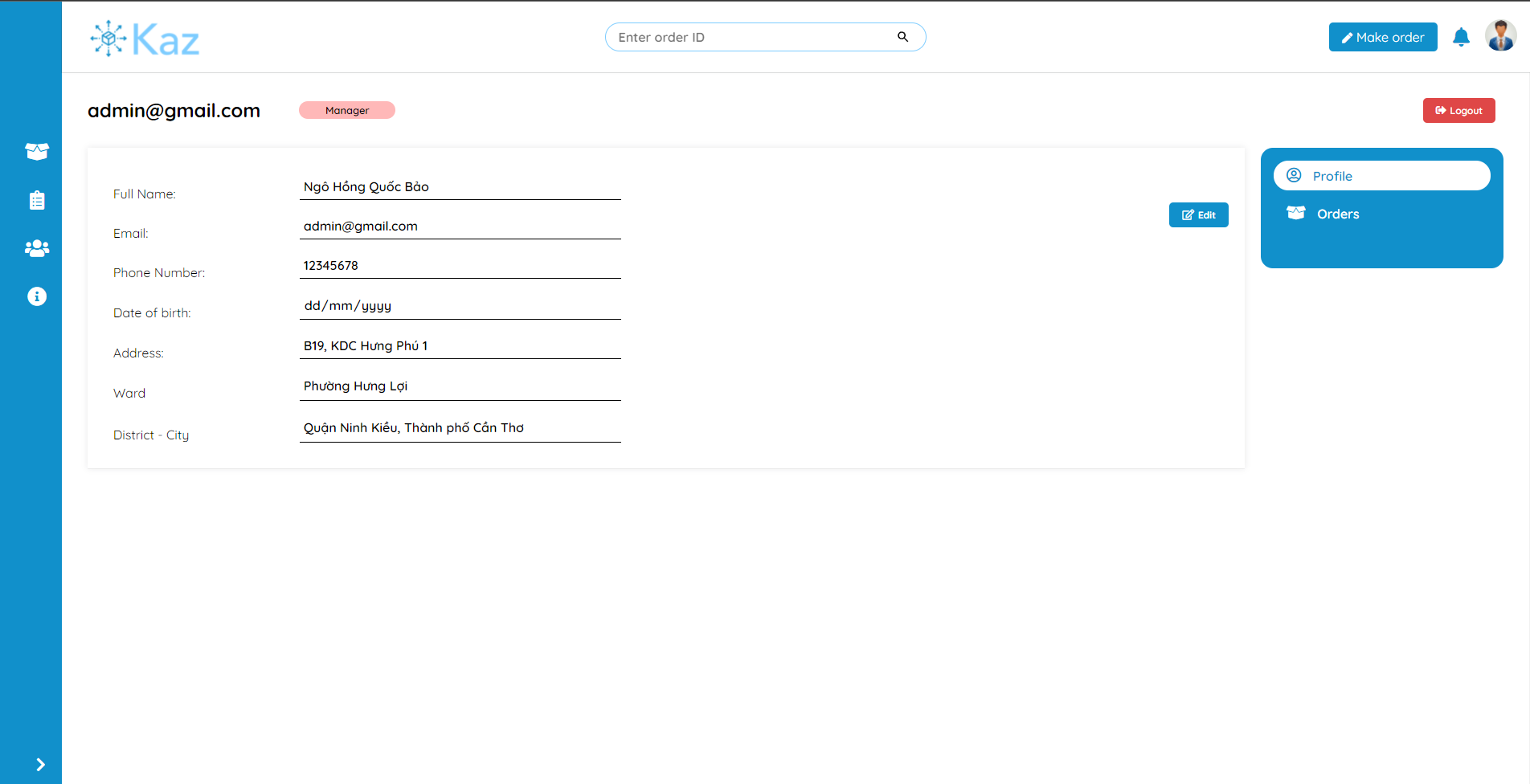


Figure 18: System Screenshot - Profile page

3.5. Pricing list management page

Pricing list page displays the prices correspond to each range of distance that a driver has travel to deliver an order.

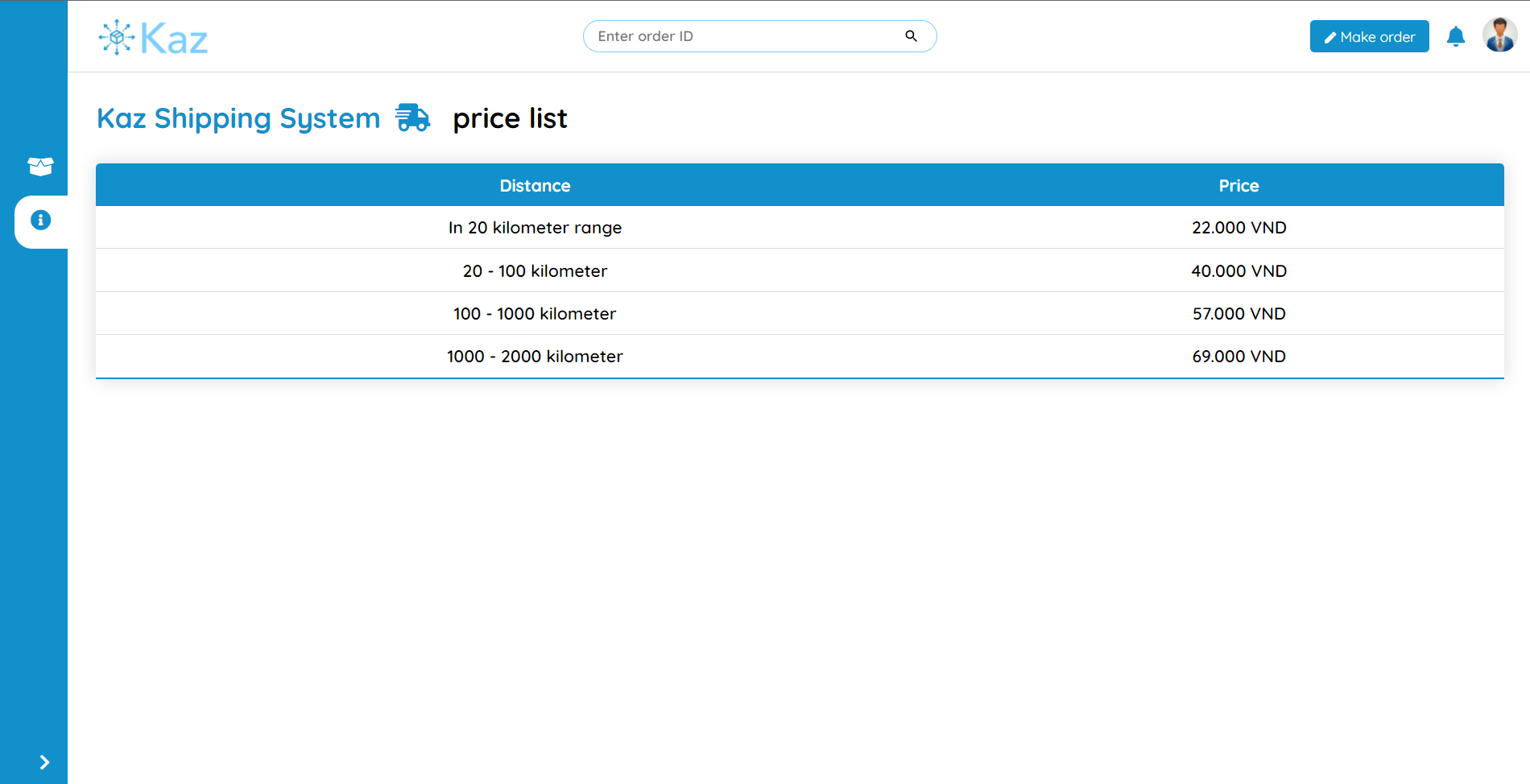


Figure 19: System Screenshot - Pricing list page on user view

For a manager account, he/she can also edit, delete or create a new price tag.

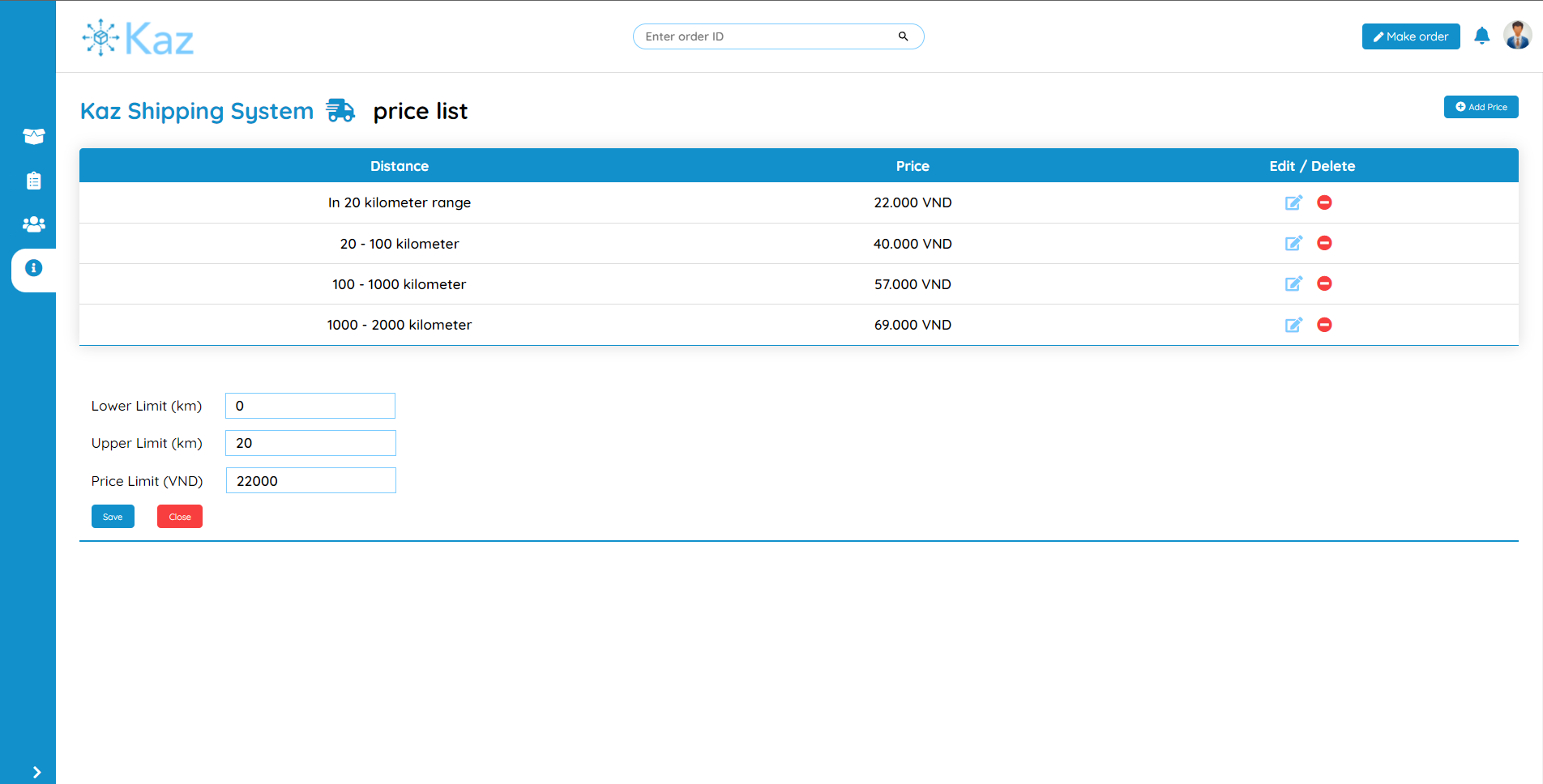


Figure 20: System Screenshot - Pricing list page on manager view

3.6. Reports page

Report page displays information about all the orders which are placed by customers in different periods of time (today, last 7 days, last month and all time) and also the calculated revenue from them.

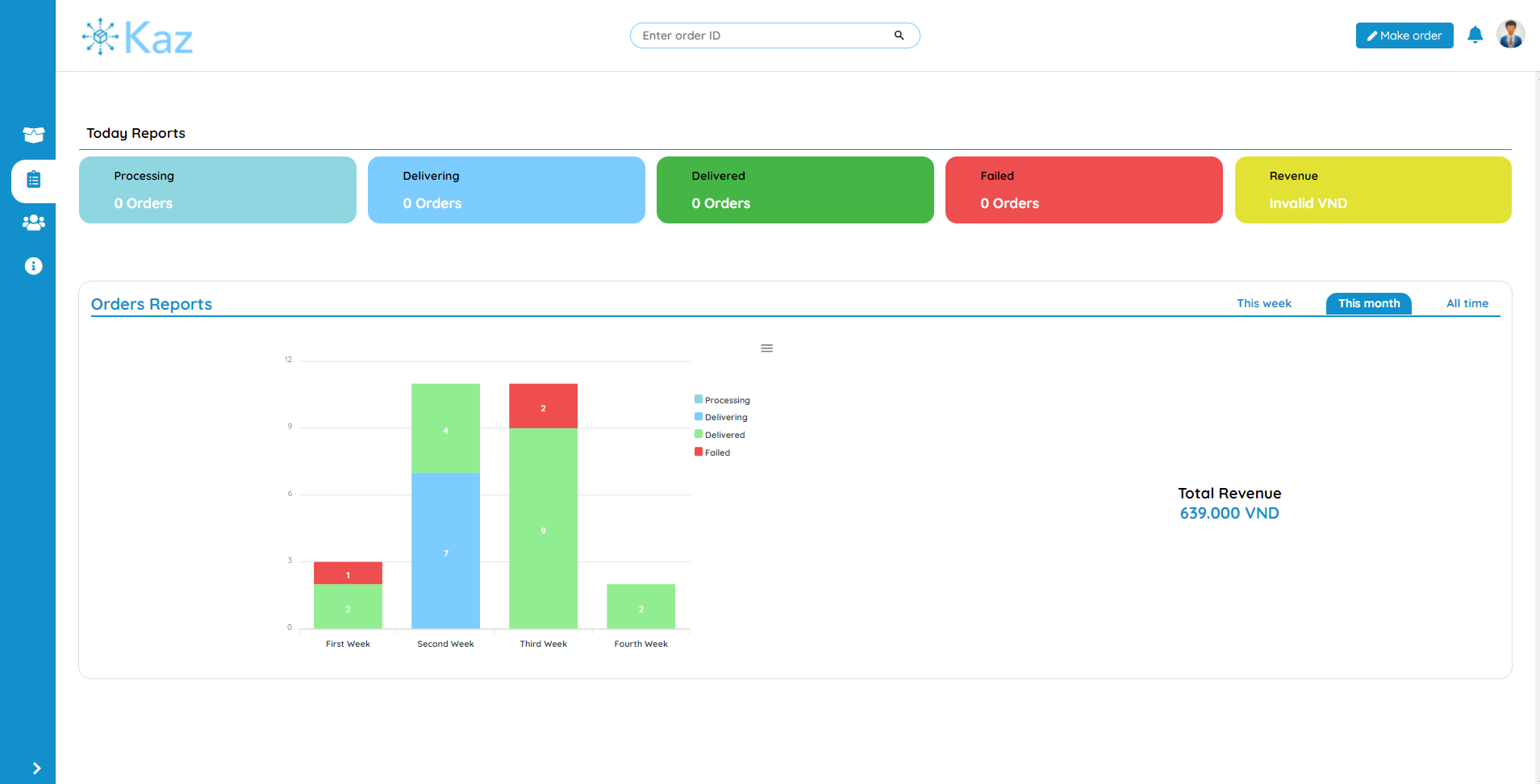


Figure 21: System Screenshot - Report page

3.7. Order Detail page

Order detail page displays information of an order. This information includes consignor and consignee’s information, the status, note, payment method and the date created of the order, and the package information from that order.

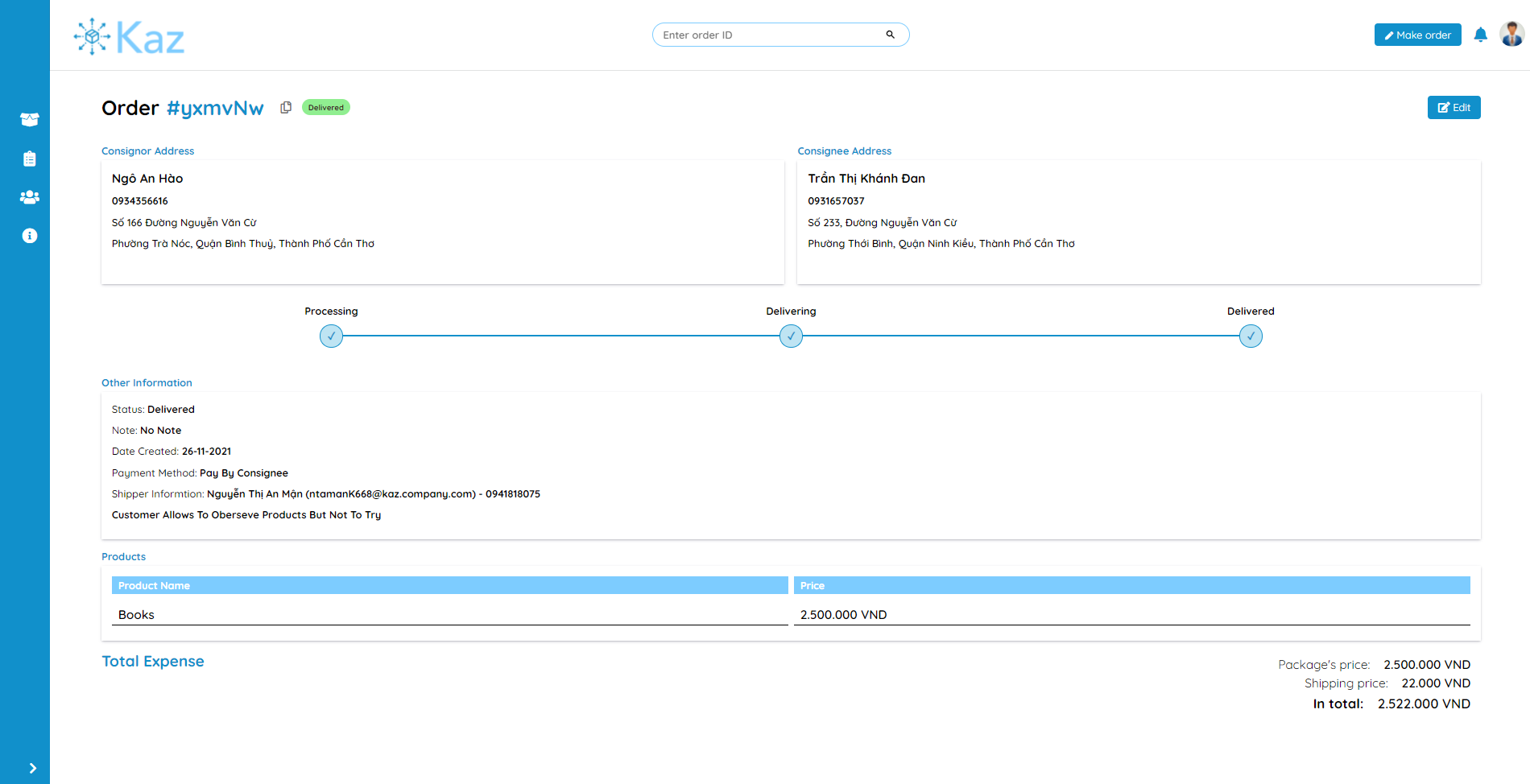


Figure 22: System Screenshot - Order detail page

For an order that is delivering, there is also a map displays on the page for user to easily track the current location of that order. User can freely manipulate the map (zoom, move around, …). There is also a “Focus on your order” button to focus the map to zoom in to the order location.

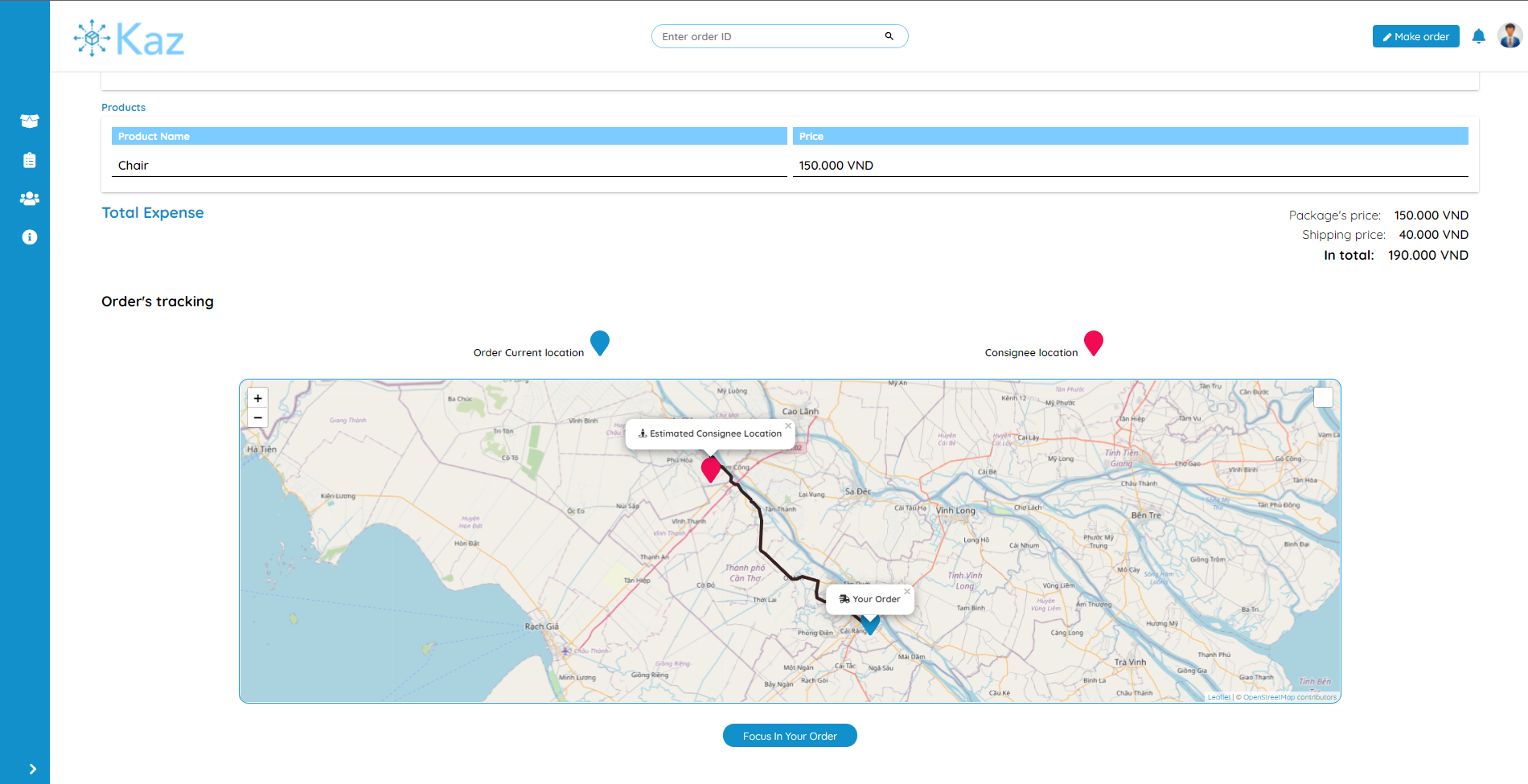


Figure 23: System Screenshot - Tracking order in Order detail page

3.8. Order placement page

A page for customer to place an order. The fields that requires to place an order include consignor and consignee’s addresses, information of the package and other information regarding the delivering process such as the payment method, how the consignee can preview the product, or any note for the package.

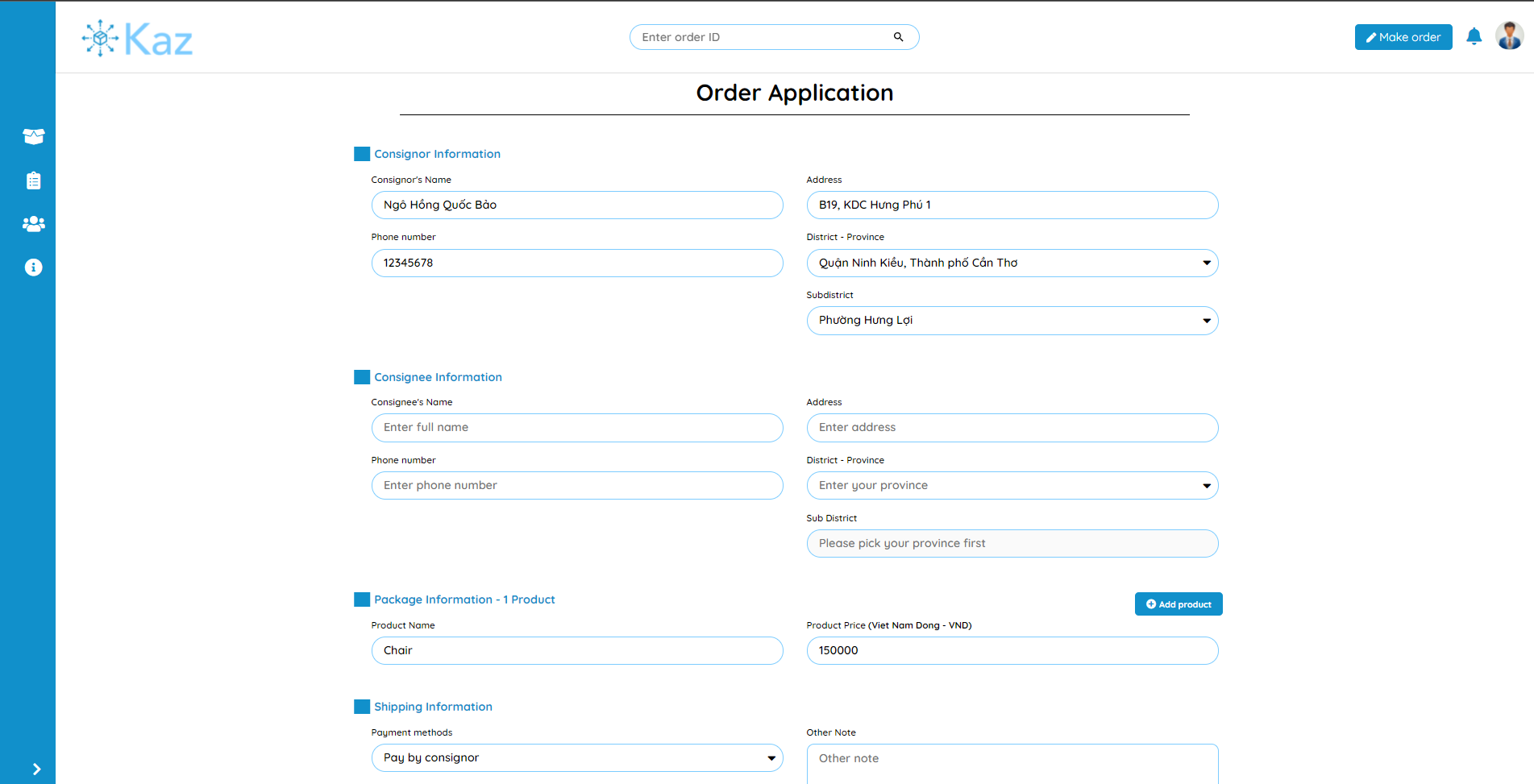


Figure 24: System Screenshot - Order placement page

When the all the fields are filled and valid, and the customer hits the “Submit” button. The system will make a couple of API calls:

* Post requests to Open Route Service API to retrieve the location of the consignee.
* Post requests to Leaflet Routing Machine API to retrieve the estimated distance and time to travel from the company location to the consignee location.
* Post requests to the backend API to retrieve the service price depends on the distance.

If there are any errors from any of these API calls, an error message will be display to inform the customer to try again later. Otherwise, a pop-up will appear to display the information from the API calls and re-confirm all the inputs from the customer, if everything is valid, customer can place an order by click the “Place order” button.

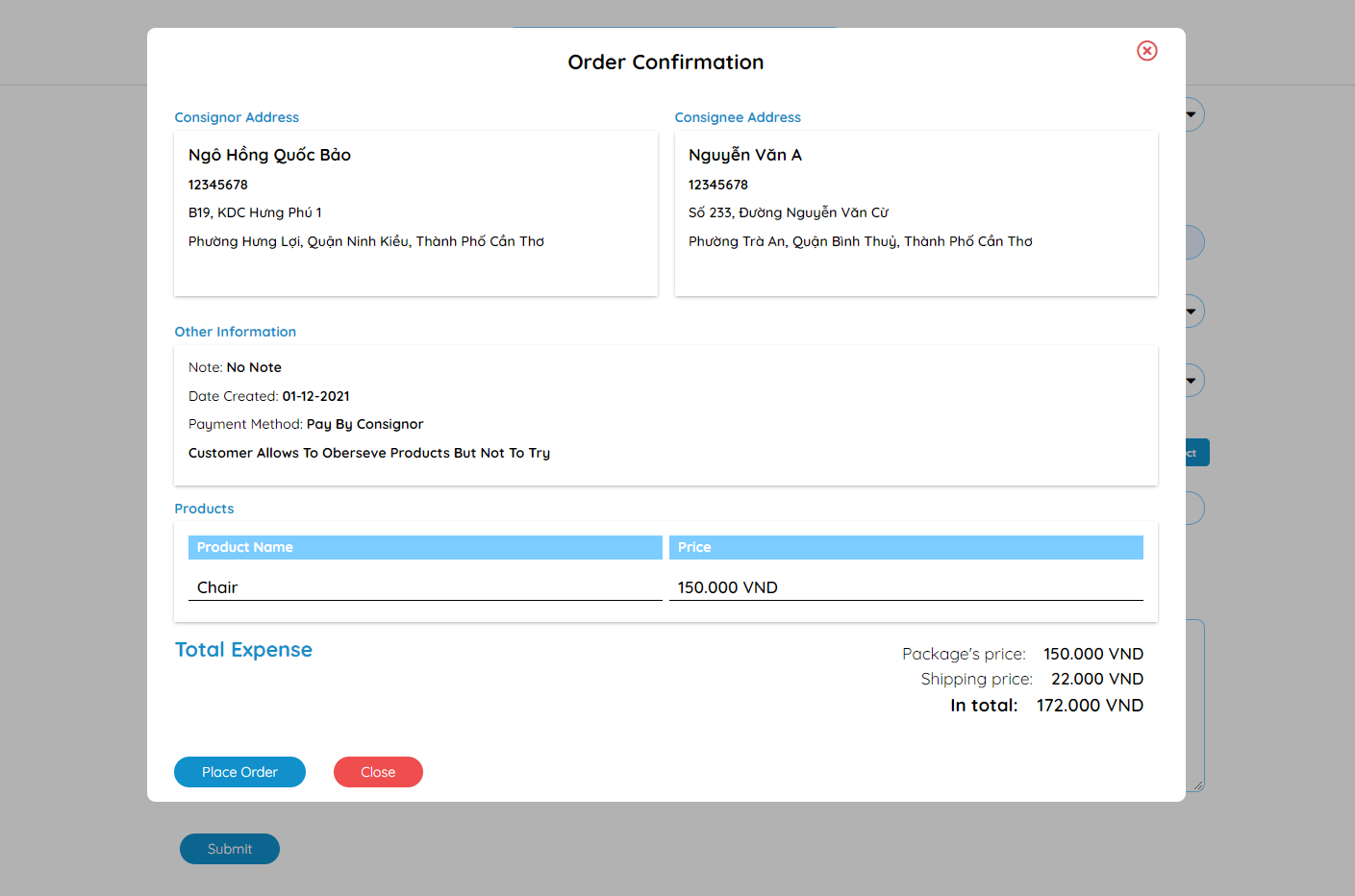


Figure 25: System Screenshot - Order confirmation

1. TESTING AND EXPERIMENTAL RESULTS

1. The purposes of testing

To certain that the final form of the system satisfies all the basis required features. Testing will be helpful to estimate how the system’s features work, the scope of the features, the progress of the building the features. Therefore, the testing results can be served for maintenance and development of the system.

2. Testing explanation

2.1. Details of the testing plan

For the testing plan, I will prioritize to display the test cases’ results from some of the system’s main features. Similar features will not be tested and displayed. I will also clearly present the criteria and variation of the test cases.

2.1.1. Testing features

The table below displays all the features that were tested with their expected results.

| **No** | **Feature** | **Overview** | **Cases** | **Expected Results** |
| --- | --- | --- | --- | --- |
|  | Login | Customer or Manager logins to the system | * Valid inputs * Wrong email * Wrong password | * Successfully login * Display error corresponds to the input |
|  | Register | Customer registers his/her email to the system | * Valid inputs * Invalid inputs (email format, insecure password) | * Successfully register * Display error corresponds to the input |
|  | Add/Edit pricing list | Manager adds or edits a price in the pricing list | * Valid inputs * Invalid inputs (empty fields, invalid price format) | * Successfully add/update price. * Display error corresponds to the input |
|  | Search order | Manager or customer search for an order by using its id | * Valid order id * Invalid order id (non-existence id or order not belongs to customer) | * Successfully view the detail of an order * Display error corresponds to the input |
|  | Find consignee location | Calculate the coordinate by using the Open Route Service API of the consignee location base on the input | * Calculate the location from the address | * Find the geolocation of the consignee’s address successfully * Display errors when no location is found. |
|  | Navigate and calculate distance | Calculate distance and routes from the consignee and the order’s location | * Distance from the company to the consignee’s location when the order is first placed. * Distance from the driver to the consignee when the order is being delivered. | * Successfully calculate distance and routes to display. * Display error when no routes are found. |

Table 15: Features require to be tested

2.1.2. The goal of testing

* Successful test case: Test case is considered success if the experimental results of that test case meet the expected results. If there is an error which are expected, the message for that error must be clear and explicit.
* Failed test case: Test case is considered failed if the experimental results are unexpected compared to the expected results or there are errors appear without mentioning before that affect negatively to how the system work.

2.1.3. Result from the test cases

* Plan for the testing.
* All the cases for the testing.
* Reports of the test results.

2.2. Test environment

Hardware:

* Personal computer with stable internet connection.
* CPU Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz; RAM 8.00 GB; SSD 256GB.

Software:

* Windows 10 operating system.
* Google Chrome version 96.0.4664.45 (official version) (64 bit).

2.3. Test cases

From the list of features that are displayed from in Table 8, I began to build the plan for testing each feature. Every test case will have the input, experimental results along with the evaluation. The plans for each feature are describe in the tables below.

2.3.1. Login

Test case ID: TC001

Conditions: Testing device is connected to internet and tester is required to have an account which is authenticated by the system.

Account for test case:

* Email: admin@gmail.com
* Password: kaz123

| **No** | **Plan** | **Inputs** | **Expected result** | **Experimental**  **result** | **Evaluation** |
| --- | --- | --- | --- | --- | --- |
| TC001-01 | No information filled | No input | Display error message that all information needs to be displayed | Expected | Success |
| TC001-02 | Invalid email or invalid password | Email is not registered in the system  Wrong password | Display “Invalid email or password” message | Expected | Success |
| TC001-03 | Login successfully | All inputs are invalid | Login successfully and redirect to the management page according to user’s role | Expected | Success |

Table 16: Feature test cases - Login

2.3.2. Register

Test case ID: TC002

Conditions: Testing device is connected to the Internet

| **No** | **Plan** | **Inputs** | **Expected result** | **Experimental**  **result** | **Evaluation** |
| --- | --- | --- | --- | --- | --- |
| TC002-01 | No information filled | No input | Display error message that all information needs to be displayed | Expected | Success |
| TC002-02 | Email is already taken | Email is already registered in the system  Any password | Display “Invalid email” message | Expected | Success |
| TC002-03 | Register successfully | All inputs are invalid | Register successfully and redirect to login page | Expected | Success |

Table 17: Feature test cases - Register

2.3.3. Add/Edit pricing list

Test case ID: TC003

Conditions: Tester is login to the system and tester account is an admin/manager account.

| **No** | **Plan** | **Inputs** | **Expected result** | **Experimental**  **result** | **Evaluation** |
| --- | --- | --- | --- | --- | --- |
| TC003-01 | No information filled | No input | Display error message that all information needs to be displayed | Expected | Success |
| TC003-02 | Price is not valid (under 10.000 VND) | Any upper and lower limit  Under 10.000 VND price | Display “Invalid price tag” message | Expected | Success |
| TC003-03 | Invalid lower limit and upper limit | Lower limit is bigger than upper limit  Any price that bigger than 10.000 VND | Display “Invalid lower and upper distance” message | Expected | Success |
| TC003-04 | Add/Edit successfully | Lower limit is smaller than upper limit  Any price that bigger than 10.000 VND | Add/Edit successfully | Expected | Success |

Table 18: Feature test cases - Add/Edit pricing list

2.3.4. Search for order by its id

Test case ID: TC004

Conditions: There is at least one order in the database.

Testing accounts:

* Manager account: admin@gmail.com
* Customer account: baobao@gmail.com

| **No** | **Plan** | **Inputs** | **Expected result** | **Experimental**  **result** | **Evaluation** |
| --- | --- | --- | --- | --- | --- |
| TC004-01 | Search order by its id without login to the system | Valid order id | Redirect to the Order Preview page. | Expected | Success |
| TC004-02 | Search order by its id with the manager account | Valid order id | Redirect to the Order Detail page. | Expected | Success |
| TC004-03 | Search order by its id with the customer account | Valid order id and the order is owned by the customer | Redirect to the Order Detail page. | Expected | Success |
| TC004-04 | Search un-related order by its id with the customer account | Valid order id but the order is not owned by the customer | Redirect to the Unauthenticated page. | Expected | Success |
| TC004-05 | Search order with invalid id (non-exist order) | Order id is not recorded in the database. | Redirect to the 404 page. | Expected | Success |

Table 19: Feature test cases - Order searching

2.3.5. Find consignee information

Test case ID: TC005

Conditions: Tester is login to the system.

Scenario: In this test, I will test the system by feeding the Open Route Service API a lot of valid addresses, which spread across all three part of Vietnam.

Expected results: The Open Route Service API return a geolocation (latitude and longitude) for each address.

Input format: The address format will only contain the ward name, the district name and the province/city name.

Example of an address: “Hưng Lợi, Cái Răng, Cần Thơ”

| **Number of cases** | **Successful case** | **Success rate** |
| --- | --- | --- |
| 50 | 37 | 74% |

Table 20: Feature test cases - Find consignee location

2.3.6. Navigate and calculate distance

Test case ID: TC005

Conditions: Tester is login to the system.

Scenario: In this test, the inputs are from the cases which the system had successfully get the geolocations (of the consignee addresses) from test TC005.

Expected results: The Leaflet Routing Machine API returns the path and the distance from the company location to the consignee location.

Input format: Geolocation of the company location and geolocation of the consignee address.

| **Number of cases** | **Successful case** | **Success rate** |
| --- | --- | --- |
| 37 | 33 | 89.19% |

Table 21: Feature test cases - Navigate and calculate distance

CONCLUSION

1. Final results

Throughout the time of research and producing this thesis with the “Delivery management system” topic, I had an opportunity to expose myself with a lot of web development technologies and services. I am also able to build a full functional delivery system that matches all the basis criteria. My system can help the manager to manage the employees, the pricing list and the orders, track their locations and the customer can place an order, track the order. The system’s user interface is built with user experiment in mind and met the goal of being user friendly, easy to use and comprehensible.

2. Future works

In this topic, due to limited time and resources I mainly focused in developing and finalizing the basis and most important features of a delivery system. In other hand, I also prepare some future paths to improve the system in the future.

* Using VISA cards to use the Google Map API for better finding and routing between locations.
* Implementing online paying methods such as PayPal, MoMo for the consignor to pay the delivery fee right after he/she confirms the order placement.
* Researching and implementing the “Return goods” policy for customers to return the products that they are not happy with.
* Developing a fully functional mobile app for company’s drivers to update their location and route their way to the consignee location.

REFERENCES

1. Giao hàng tiết kiệm, “Dịch vụ Giao hàng tiết kiệm” from <https://giaohangtietkiem.vn/dich-vu-giao-hang-ghtk>.
2. Giao hàng nhanh, “Dịch vụ giao hàng nhanh” from <https://ghn.vn/pages/dich-vu-giao-hang>.
3. Andrew Smith, “Building Management System (BMS)”, *City of Melbourne*.
4. MDN Web Docs October 8th 2021, accessed November 1st 2021, <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>.
5. Rohan Vats, “Introduction To Django REST Framework: Development with Examples”, *upGrad*, 10/05/2021.
6. Vuejs.org, “Vue.js Introduction” from <https://vuejs.org/v2/guide/>.
7. Wikipedia December 5th 2021, accessed October 20th 2021, <https://en.wikipedia.org/wiki/Leaflet_(software)>.
8. Per Liedman, Leaflet Routing Machine 2015, accessed October 20th 2021, <http://www.liedman.net/leaflet-routing-machine/>.
9. Open Route Service, accessed October 20th 2021, <https://openrouteservice.org/>.