

Blue Bird

Order Management System

Black Sabbath

Amadeus Darren Leander - 2101628915

Bernard Wijaya - 2101636910

Dave Zachary - 2101671195

Ryan Adipradana - 2101626903

Benny Wiryawan - 1901497546

Computer Science Global Class – LA02

**Preface**

In this project, we created an object-oriented database system to simulate Blue Bird’s taxi ordering service. The form-based application is created using C# Windows Form Application and runs on the Windows operating system. The application involves users, drivers and also the admin who manages the database.

**Table of Contents**

[Preface](#_airrw93fl7rx) 2

[Table of Contents](#_8jhbxdijtjpb) 2

[Requirement Analysis](#_tkshfgyva3fb) 3

[Conceptual Design](#_nshonlepoqy5) 3

[Logical Design](#_842kty6s8qv0) 4

[Object Definition Language](#_dzl35wwlhzac) 5

[Report Generation and Object Manipulation Language](#_wxtsbnlx9552) 7

[Conclusion](#_1070u0aggnbr) 12

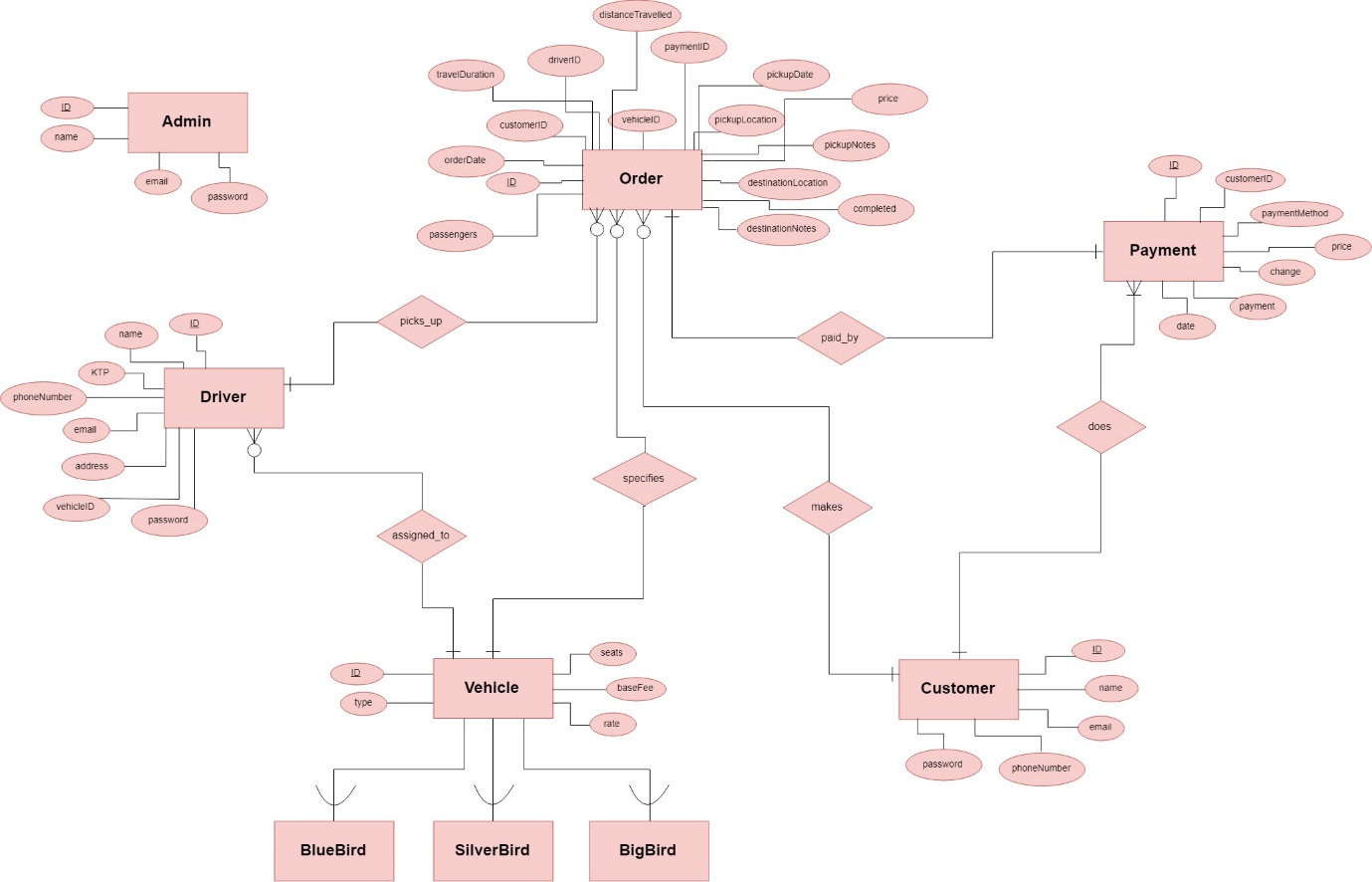
[References](#_3qyo0n6fkuwj) 12

**Requirement Analysis**

To simulate the Blue Bird’s taxi ordering process, first we must understand the individuals and organizations involved in the process. By observing the real-world domain, we concluded that the platform consists of interactions between users, drivers and system admins.   
The entire process flow starts with the user, whose role is to create an order which they can also cancel. Drivers must then accept the order and after picking up and delivering the user to their destination, they must confirm that the ride has been completed. The admin holds the most tasks in the system because they must manage the users, drivers, vehicles and transactions.

**Conceptual Design**

In this process, we model the real-world domain by creating the data model required to fulfill the business requirements. To model it we created the Entity Relationship Diagram below.



**Logical Design**

The logical design is represented by a class diagram below.



**Object Definition Language**

**class** Order (**extent** Order) {

**attribute** string ID;

**attribute** DateTime orderDate;

**attribute** string customerID;

**attribute** string driverID;

**attribute** string vehicleID;

**attribute** string paymentID;

**attribute** string pickupLocation;

**attribute** string pickupNotes;

**attribute** string destinationLocation;

**attribute** string destinationNotes;

**attribute** DateTime pickupDate;

**attribute** int passengers;

**attribute** TimeSpan travelDuration;

**attribute** int distanceTravelled;

**attribute** int price;

**attribute** bool completed;

**relationship** Customer made\_by **inverse** Customer::makes;

**relationship** Driver picked \_by **inverse** Driver::picks\_up;

**relationship** Vehicle specifies **inverse** Vehicle::specified\_by;

**relationship** Payment paid\_by **inverse** Payment::pays;

}

**class** Driver (**extent** Driver) {

**attribute** string ID;

**attribute** string name;

**attribute** string KTP;

**attribute** string phoneNumber;

**attribute** string email;

**attribute** string password;

**attribute** string vehicleID;

**relationship** **list**<Order> picks\_up **inverse** Order::picked\_by;

**relationship** Vehicle assigned\_to **inverse** Vehicle::assigned\_to;

}

**class** Vehicle (**extent** Vehicle) {

**attribute** string ID;

**attribute** string type;

**attribute** int seats;

**attribute** int baseFee;

**attribute** int rate;

**relationship** Driver assigned\_to **inverse** Driver::assigned\_to;

**relationship** Order specified\_by **inverse** Order::specifies;

}

**class** BlueBird **extends** Vehicle (**extent** BlueBird) {

}

**class** SilverBird **extends** Vehicle (**extent** SilverBird) {

}

**class** BigBird **extends** Vehicle (**extent** BigBird) {

}

**class** Customer (**extent** Customer) {

**attribute** string ID;

**attribute** string name;

**attribute** string email;

**attribute** string phoneNumber;

**attribute** string password;

**relationship** **list**<Order> makes **inverse** Order::made\_by;

**relationship** **list**<Payment> does **inverse** Payment::done\_by;

}

**class** Payment (**extent** Payment) {

**attribute** string ID;

**attribute** string customerID;

**attribute** string paymentMethod;

**attribute** int price;

**attribute** int payment;

**attribute** int change;

**attribute** DateTime date;

**relationship** Order pays **inverse** Order::paid\_by;

**relationship** Customer done\_by **inverse** Customer::does;

}

**class** Admin (**extent** Admin) {

**attribute** string ID;

**attribute** string name;

**attribute** string email;

**attribute** string password;

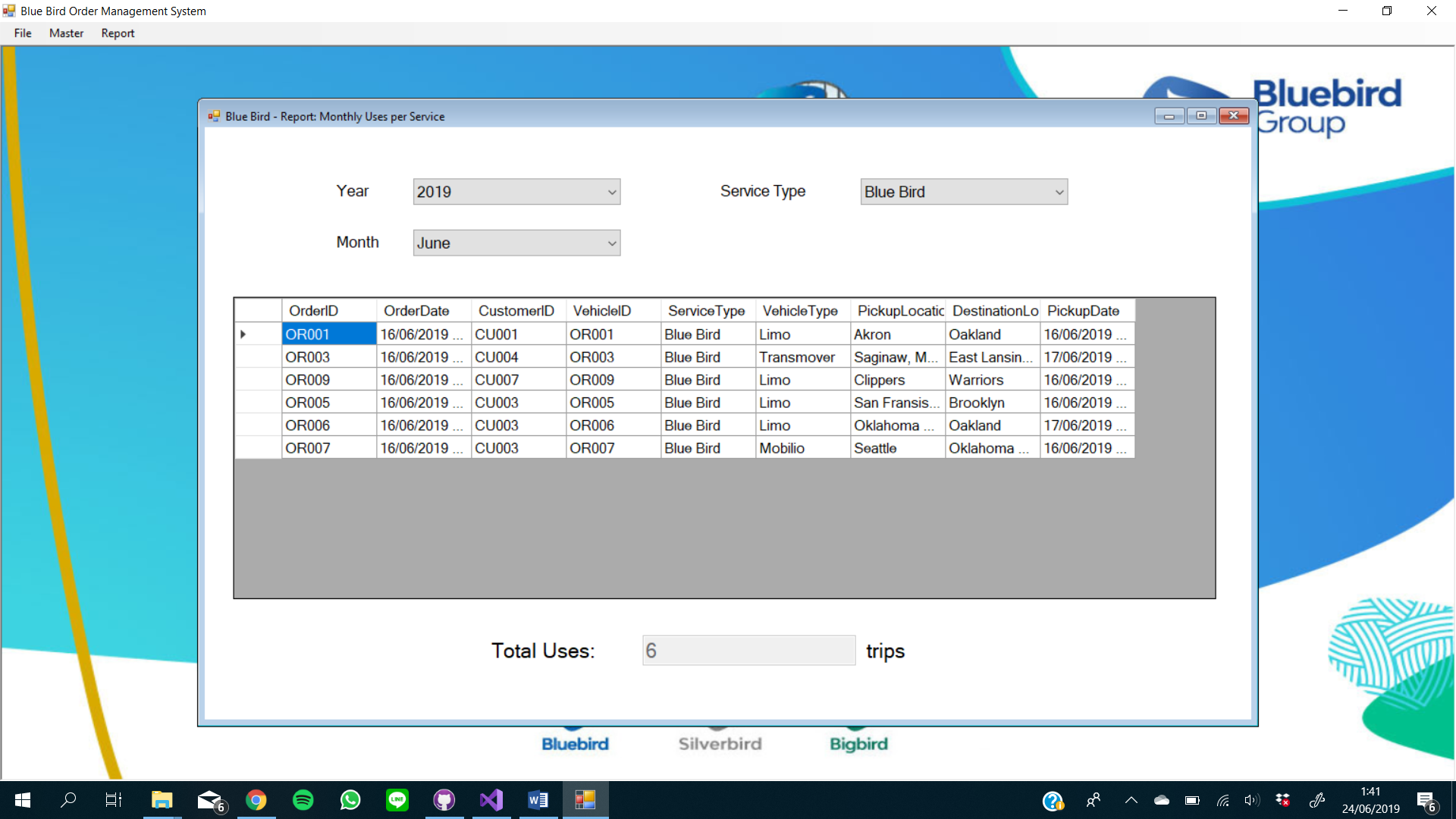
}

**Report Generation and Object Manipulation Language**

Our order management system can generate 6 different reports as listed below:

1. **Monthly uses per service type**

In this report, the admin will be able to view all of the orders, based on the service type, as well as the year and month is placed. Aside from displaying the data, the report will also show the total number of uses.



if (cbService.SelectedIndex == 1)

{

var getOrders = (from Order o in DB.conn

join BlueBird b in DB.conn

on o.vehicleID equals b.ID

where o.pickupDate.Year == year

&& o.pickupDate.Month == month

select new

{

OrderID = o.ID,

OrderDate = o.orderDate,

CustomerID = o.customerID,

VehicleID = o.ID,

ServiceType = "Blue Bird",

VehicleType = b.type,

PickupLocation = o.pickupLocation,

DestinationLocation = o.destinationLocation,

PickupDate = o.pickupDate

}).ToList();

dgvOrders.DataSource = getOrders;

txtUses.Text = getOrders.Count.ToString();

}

else if (cbService.SelectedIndex == 2)

{

var getOrders = (from Order o in DB.conn

join SilverBird b in DB.conn

on o.vehicleID equals b.ID

where o.pickupDate.Year == year

&& o.pickupDate.Month == month

select new

{

OrderID = o.ID,

OrderDate = o.orderDate,

CustomerID = o.customerID,

VehicleID = o.ID,

ServiceType = "Silver Bird",

VehicleType = b.type,

PickupLocation = o.pickupLocation,

DestinationLocation = o.destinationLocation,

PickupDate = o.pickupDate

}).ToList();

dgvOrders.DataSource = getOrders;

txtUses.Text = getOrders.Count.ToString();

}

else if (cbService.SelectedIndex == 3)

{

var getOrders = (from Order o in DB.conn

join BigBird b in DB.conn

on o.vehicleID equals b.ID

where o.pickupDate.Year == year

&& o.pickupDate.Month == month

select new

{

OrderID = o.ID,

OrderDate = o.orderDate,

CustomerID = o.customerID,

VehicleID = o.ID,

ServiceType = "Big Bird",

VehicleType = b.type,

PickupLocation = o.pickupLocation,

DestinationLocation = o.destinationLocation,

PickupDate = o.pickupDate

}).ToList();

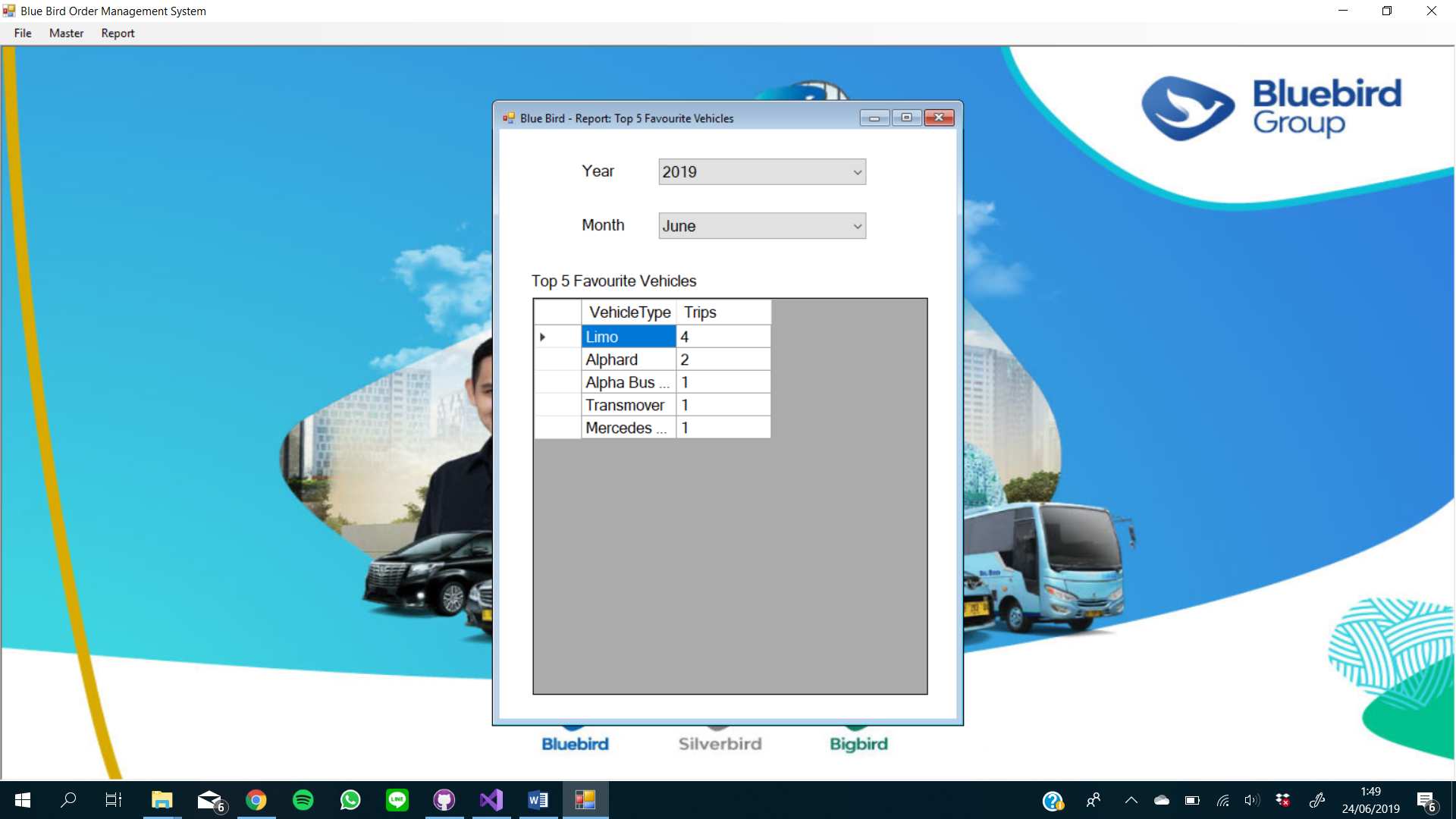
dgvOrders.DataSource = getOrders;

txtUses.Text = getOrders.Count.ToString();

}

1. **Top 5 favorite vehicles**

This report will show the most used vehicles on trips that were done on a user specified year and month.



var getData = (from Order o in DB.conn

join Vehicle v in DB.conn

on o.vehicleID equals v.ID

where o.pickupDate.Year == year

&& o.pickupDate.Month == month

group v.type by v.type into g

orderby g.Count() descending

select new

{

VehicleType = g.Key,

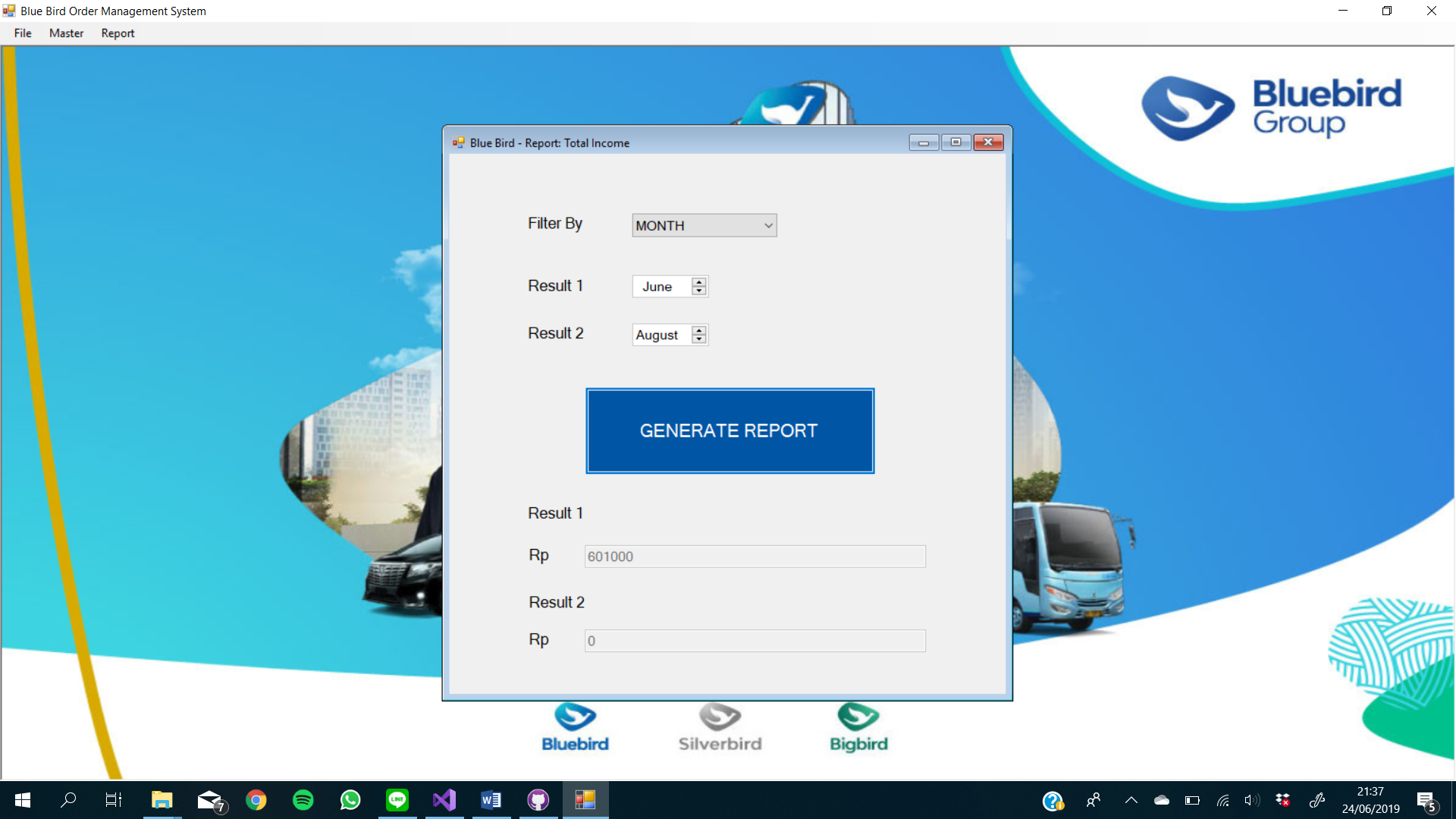
Trips = g.Count()

}).Take(5).ToList();

dgvFavourites.DataSource = getData;

1. **Total income earned**

In this report, the admin will be able to discover the amount of total income earned, filtered by month or by year. The admin will also be able to view the total income of two different months/years, so it will be easier for them to compare.



date1 = dateTimePicker1.Value.Month;

date2 = dateTimePicker2.Value.Month;

resultByMonth1 = (from Order x in DB.conn

where x.completed == true && x.orderDate.Month == date1

select x.price);

resultByMonth2 = (from Order x in DB.conn

where x.completed == true && x.orderDate.Month == date2

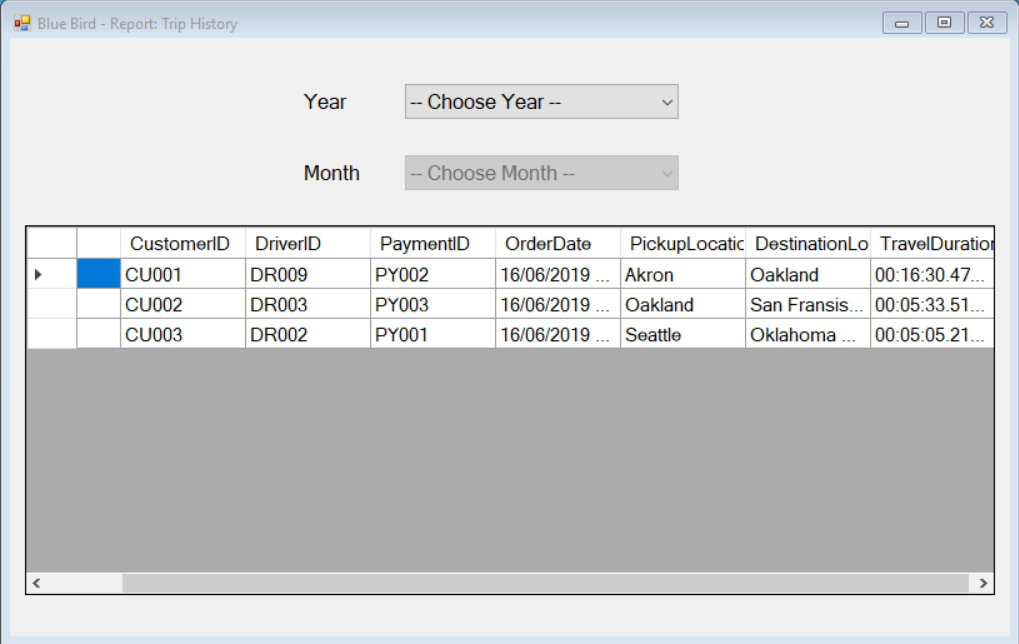
select x.price);

result1Tb.Text = resultByMonth1.Sum().ToString();

result2Tb.Text = resultByMonth2.Sum().ToString();

1. **History of complete trips**

In this report, the admin can see all completed trips done by the user and is filtered by month or by year



var data = (from Order o in DB.conn

join Driver d in DB.conn on o.driverID equals d.ID

join Customer c in DB.conn on o.customerID equals c.ID

join Payment p in DB.conn on o.paymentID equals p.ID

where o.completed == true

select new

{

OrderID = o.ID,

CustomerID = c.ID,

DriverID = d.ID,

PaymentID = p.ID,

OrderDate = o.orderDate,

PickupLocation = o.pickupLocation,

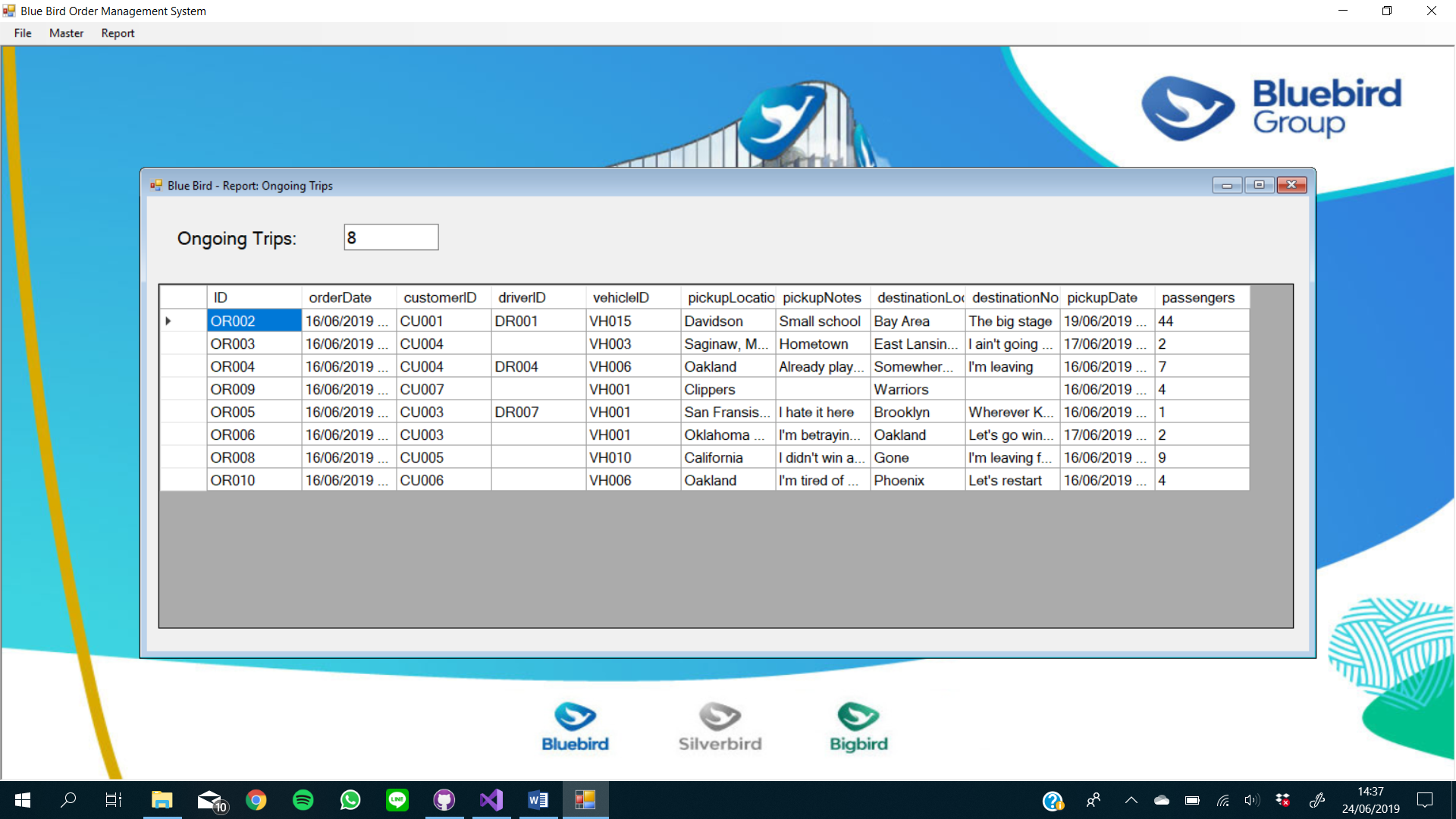
DestinationLocation = o.destinationLocation,

TravelDuration = o.travelDuration

}).ToList();

1. **List of currently ongoing trips**

This report will show all of the orders that are currently ongoing and have not been completed yet.



var getTrips = (from Order x in DB.conn

where x.completed == false

select new

{

x.ID,

x.orderDate,

x.customerID,

x.driverID,

x.vehicleID,

x.pickupLocation,

x.pickupNotes,

x.destinationLocation,

x.destinationNotes,

x.pickupDate,

x.passengers

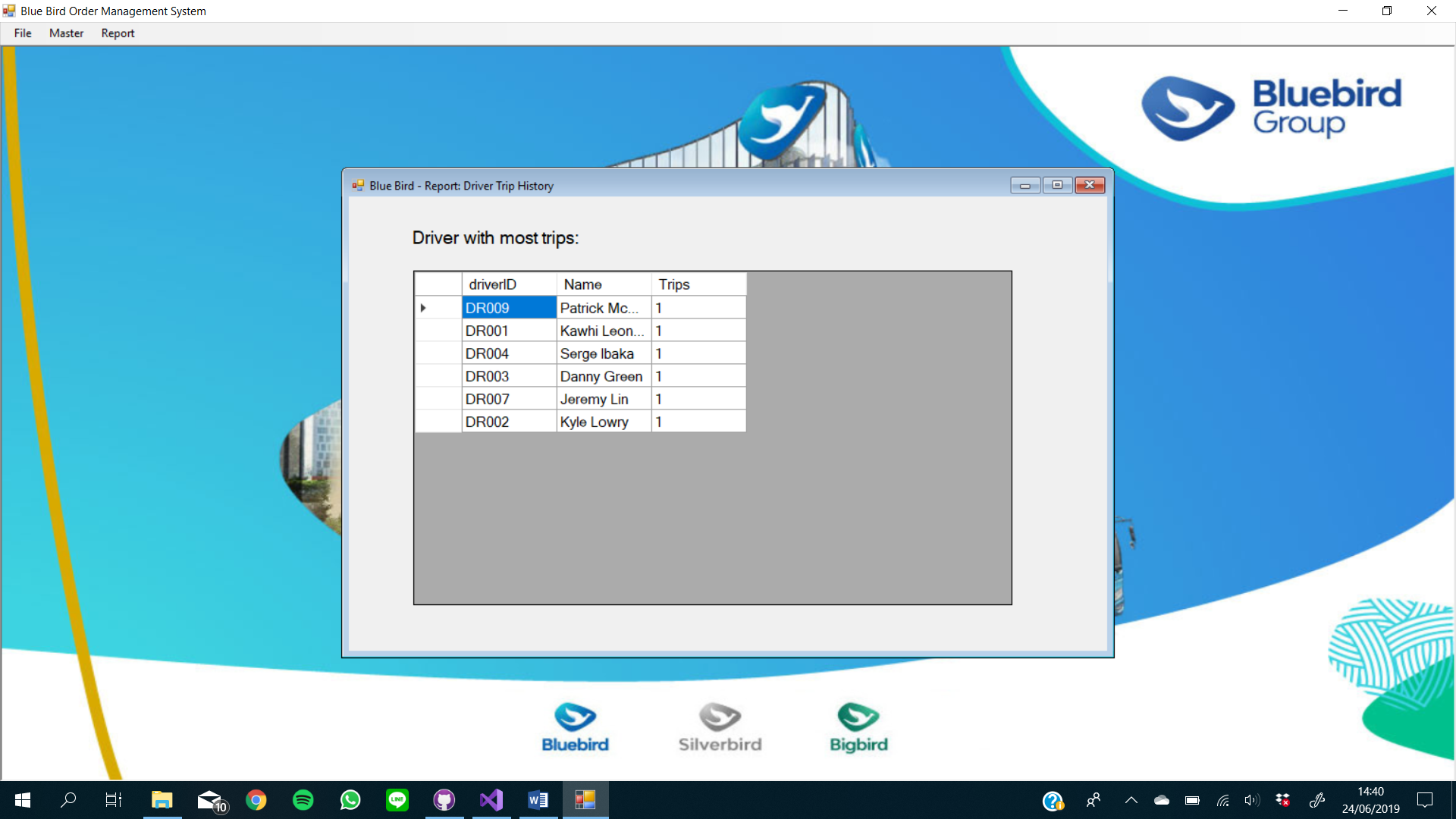
}).ToList();

dgvOngoing.DataSource = getTrips;

txtOngoing.Text = getTrips.Count.ToString();

1. **Driver trip history**

In this report, we show the number of trips each driver has completed all-time.



var getDriver = (from Order o in DB.conn

join Driver d in DB.conn

on o.driverID equals d.ID

group o by new { o.driverID, d.name } into g

let count = g.Count()

orderby count descending

select new

{

g.Key.driverID,

Name = g.Key.name,

Trips = count

}).Take(10).ToList();

dgvDriver.DataSource = getDriver;

**Conclusion**

The conclusion of this project is to simulate and make a prototype of BlueBird taxi ordering system which has its own database. The prototype has several services and features such as:

* Customer features
  + Making orders
  + Finishing orders
* Driver features
  + Pick up orders
* Admin features
  + Manage Customers
  + Manage Drivers
  + Manage Vehicles
  + View Orders
  + Manage Admins
* Report generation
  + Total number of users per service
  + Top 5 most used vehicles
  + Total income earned per month
  + History of completed trips
  + List of ongoing trips
  + Driver trip history

All users must either login or register first before using the application. Due to security reasons, registering through the application is only available to customers.

This Object-Oriented Database System’s project uses C# Windows Form Application as the main programming language and uses db4o, especially LINQ, to create the ODLs and OQLs which are the main components to access and create databases based on our C# code.

**References**

* https://www.bluebirdgroup.com/
* https://www.cnnindonesia.com/ekonomi/20190422203127-92-388640/tarif-taksi-blue-bird-listrik-sama-dengan-konvensional
* https://www.cnbcindonesia.com/news/20181114100359-4-42047/mana-lebih-mahal-tarif-baru-taksi-online-atau-blue-bird-cs
* https://www.bluebirdgroup.com/wp-content/uploads/2019/01/About-Us-BB-Group.jpg
* https://bluebirdgroup.com/wp-content/uploads/2018/11/bluebird-logo-new.png