

**CS2102**

**Project Report Topic D – Car Pooling**

**Group 8**

1. Ang Wei Ming, A0168721B
2. Benjamin Chin Choon Kiat, A0168698B
3. Lee Yu Choy, A0177151H
4. Yeo Cheng Hong, A0168369L

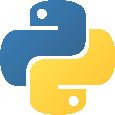
##### 6 November 2018

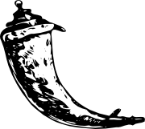
Table Of Content

**................................................................................................................................................................................................. 1**

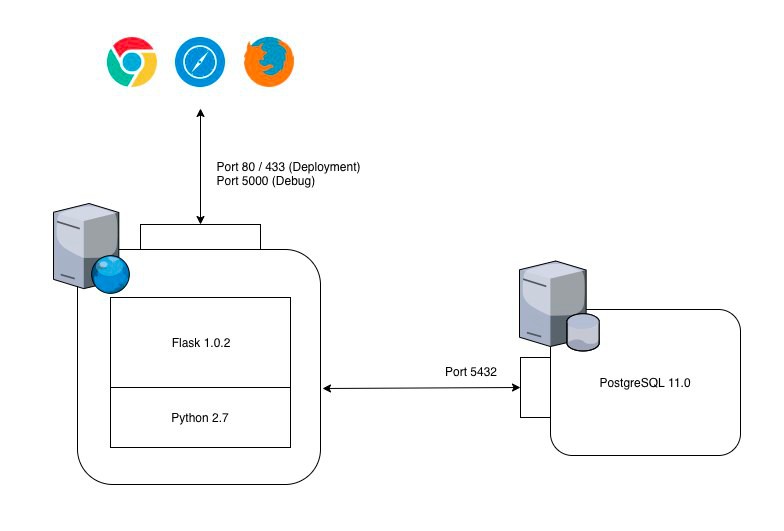
1. [GENERAL ARCHITECTURE 3](#_TOC_250029)
2. ER DIAGRAM 3
3. [DDL 4](#_TOC_250028)
   * 1. [User Schema 4](#_TOC_250027)
     2. [Car Model Schema 4](#_TOC_250026)
     3. [Car Schema 4](#_TOC_250025)
     4. [Car Ride Schema 4](#_TOC_250024)
     5. [Car Ride Bid Schema 5](#_TOC_250023)
     6. [Transaction Log Schema 5](#_TOC_250022)
   1. [TRIGGERS AND FUNCTIONS 5](#_TOC_250021)
      1. [After Approval Checks 5](#_TOC_250020)
      2. [Constraint checks before bidding. 6](#_TOC_250019)
      3. [Insert into audit table 7](#_TOC_250018)
      4. [Login / Register Stored Procedure 7](#_TOC_250017)
   2. [SAMPLE FUNCTION AND SQL 8](#_TOC_250016)
      1. [Search Ride 8](#_TOC_250015)
      2. [Login / Register with Stored Procedure 8](#_TOC_250014)
      3. [Ride History 9](#_TOC_250013)
4. [ASSERTION 10](#_TOC_250012)
   1. [REJECTION OF BID 10](#_TOC_250011)
      1. [Trigger behind 10](#_TOC_250010)
      2. [Simple Insert Query 11](#_TOC_250009)
   2. [BID APPROVAL 11](#_TOC_250008)
      1. [Trigger as shown below 11](#_TOC_250007)
      2. [on\_approval\_update\_pax function 11](#_TOC_250006)
   3. [REJECTING CAR UPDATES 12](#_TOC_250005)
      1. [Update Query is shown below 12](#_TOC_250004)
5. [SCREEN SHOTS 12](#_TOC_250003)
   1. [LOGIN PAGE 12](#_TOC_250002)
   2. [INDEX PAGE/ HOME 13](#_TOC_250001)
   3. [MENU 14](#_TOC_250000)

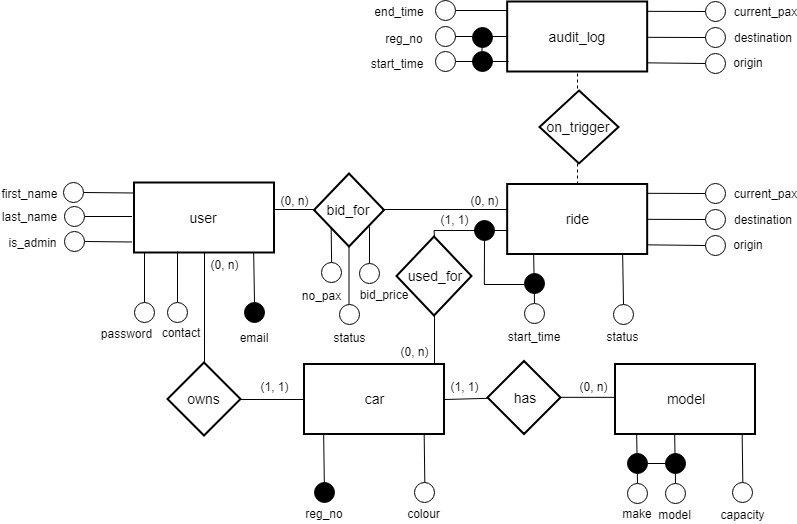
# 1 General Architecture





##### Server Language – Python 2.7

* Web Server - Python Flask 1.0.2
* Database – PostgreSQL 11.0



**2 ER Diagram**

The on trigger function for the audit\_log table copies over the entry from ride table when an entry status changes from 'in progress' to 'completed'

# 3 DDL

### User Schema

create table if not exists "user" -- `"` used because PostgreSQL use 'user' as a keyword

(

email varchar(256) not null constraint user\_pkey primary key, contact numeric(8),

first\_name varchar(50) not null, last\_name varchar(50) not null, is\_admin boolean default false not null, password varchar(512) not null

)

;

### Car Model Schema

create table if not exists model (

model varchar(256) not null, make varchar(256) not null,

capacity integer not null constraint capacity\_min check (capacity > 0), constraint model\_pk primary key (model, make)

)

;

### Car Schema

create table if not exists car (

reg\_no varchar(8) not null constraint car\_pkey primary key, colour varchar(50),

email varchar(256) not null constraint car\_email\_fkey references "user", make varchar(50) not null,

model varchar(50) not null,

constraint car\_make\_fkey foreign key (make, model) references model (make, model)

)

;

### Car Ride Schema

create table if not exists ride (

start\_time timestamp not null,

status varchar(11) not null constraint ride\_status\_type

check (((status)::text = 'in progress'::text) OR ((status)::text = 'completed'::text)), current\_pax integer not null,

destination varchar(256) not null, origin varchar(256) not null,

reg\_no varchar(8) not null constraint ride\_reg\_no\_fkey references car, constraint ride\_pkey primary key (start\_time, reg\_no)

)

;

### Car Ride Bid Schema

create table if not exists ride\_bid (

email varchar(256) not null constraint ride\_bid\_email\_fkey references "user", start\_time timestamp not null,

reg\_no varchar(8) not null,

no\_pax integer not null constraint min\_pax check (no\_pax > 0), bid\_price double precision check (bid\_price > 0),

status varchar(13) default 'pending'::character varying not null constraint bid\_status\_type check (((status)::text = 'pending'::text) OR ((status)::text = 'successful'::text) OR ((status)::text = 'unsuccessful'::text)),

constraint ride\_bid\_pkey primary key (email, start\_time, reg\_no),

constraint ride\_bid\_start\_time\_fkey foreign key (start\_time, reg\_no) references ride (start\_time, reg\_no)

)

;

### Transaction Log Schema

create table if not exists audit\_log (

start\_time timestamp not null, end\_time timestamp not null,

status varchar(11) not null constraint ride\_status\_type check ((status)::text = 'completed'::text),

current\_pax integer not null, destination varchar(256) not null, origin varchar(256) not null,

reg\_no varchar(8) not null constraint ride\_reg\_no\_fk references car, constraint ride\_pk primary key (start\_time, reg\_no)

)

;

## Triggers and Functions

### After Approval Checks

##### This trigger will runs when a bid changes from pending/unsuccessful to successful.

When it runs, it will increase the current number of passengers in the ride with the number stated in the ride bid.

Following which, all ‘pending’ bids which have their number of passengers greater than the remaining seats will be rejected.

create or replace function on\_approval\_update\_pax() returns trigger language plpgsql

as $$ BEGIN

IF NEW.status = 'successful' and OLD.STATUS <> 'successful' THEN

UPDATE ride

SET current\_pax = current\_pax + NEW.no\_pax WHERE reg\_no = NEW.reg\_no

AND start\_time = NEW.start\_time; UPDATE ride\_bid rb

SET status = 'unsuccessful' FROM ride r, car c, model m

WHERE r.reg\_no = rb.reg\_no

AND r.start\_time = rb.start\_time AND r.reg\_no = c.reg\_no

AND c.make = m.make AND c.model = m.model

AND rb.reg\_no = NEW.reg\_no

AND rb.start\_time = NEW.start\_time AND rb.status = 'pending'

AND rb.no\_pax > (m.capacity - r.current\_pax); END IF;

RETURN NULL; END

$$

;

create trigger approval\_update after update

on ride\_bid for each row

execute procedure on\_approval\_update\_pax()

;

### Constraint checks before bidding.

This function will reject insertion/update if the bids have its numbers of passengers exceed the remaining seats.

It will also reject insertion if the bids is bid by the driver himself.

create or replace function capacity\_checker() returns trigger

language plpgsql as $$

BEGIN

IF (SELECT (r.current\_pax + NEW.no\_pax <= m.capacity)

FROM ride r

inner join car c on r.reg\_no = c.reg\_no

INNER JOIN model m on c.make = m.make and c.model = m.model

AND r.reg\_no = NEW.reg\_no

AND r.start\_time = NEW.start\_time)

THEN

--

-- Do nothing ELSE

RAISE EXCEPTION 'Exceeded maximum capacity, please reduce your number of passenger'; END IF;

IF (SELECT (1)

FROM ride r

INNER JOIN car c on r.reg\_no = c.reg\_no where r.reg\_no = NEW.reg\_no

AND r.start\_time = NEW.start\_time AND c.email = NEW.email)

THEN RAISE EXCEPTION 'Cannot Bid for Own Ride'; ELSE

RETURN NEW; END IF;

END

$$;

create trigger cap\_check before insert OR update on ride\_bid

for each row

execute procedure capacity\_checker()

;

### Insert into audit table.

##### This trigger is used to insert into a similar table which help in querying transaction history. The end time of the ride will also be updated upon completion.

This extra table although is redundant, but it can help to maintain integrity of the logs, as there will not be any updates in this table. (Only Insertion)

create or replace function audit() returns trigger language plpgsql

as $$ BEGIN

IF NEW.status = 'completed' THEN INSERT INTO

audit\_log(start\_time,end\_time,status,current\_pax,destination,origin,reg\_no) VALUES

(OLD.start\_time,now(),NEW.status,OLD.current\_pax,OLD.destination,OLD.origin,OLD.reg\_no)

;

END IF;

RETURN NEW;

END;

$$

;

create trigger to\_audit before update

on ride

for each row

execute procedure audit()

;

### Login / Register Stored Procedure

##### This function will be called when the user register or login into our web application.

**NOTE** : Need to run CREATE EXTENSION as postgres to install the pgcrypto extension.

CREATE EXTENSION pgcrypto; -- Have to run as postgres / super user. Run only once.

create or replace function register(email character varying, contact numeric, first\_name character varying, last\_name character varying, password character varying) returns boolean

language plpgsql as $$

DECLARE

success BOOLEAN; hashPassword varchar(512); BEGIN

SELECT INSERT RETURN END;

$$

;

encode(digest($5, 'sha256'), 'hex') INTO hashPassword;

INTO "user" VALUES ($1,$2,$3,$4,false ,hashPassword) Returning 1 into success; success;

create

or replace function login(email character varying, password character varying)

returns SETOF "user" language plpgsql

as $$

DECLARE

hashPassword varchar(512); BEGIN

SELECT encode(digest($2, 'sha256'), 'hex') INTO hashPassword;

RETURN QUERY SELECT \* FROM "user" where "user".email = $1 and "user".password = hashPassword;

END;

$$

;

## Sample Function and SQL

### Search Ride

#### If origin and destination is NULL, return all rides

SELECT u.first\_name,u.email,r.origin,r.destination,r.status,r.reg\_no, r.start\_time, r.current\_pax, (m.capacity - r.current\_pax) as pax\_left,

EXISTS (SELECT c1.email FROM car c1 WHERE c1.reg\_no = c.reg\_no AND c1.email = %s) as is\_driver,

EXISTS (SELECT rb.email FROM ride\_bid rb WHERE rb.reg\_no = c.reg\_no AND rb.email = %s AND rb.status = 'successful' AND rb.start\_time = r.start\_time) as has\_success\_bid, EXISTS (SELECT rb.email FROM ride\_bid rb WHERE rb.reg\_no = c.reg\_no AND rb.email = %s AND rb.status = 'unsuccessful' AND rb.start\_time = r.start\_time) as has\_unsuccessful\_bid,

EXISTS (SELECT rb.email FROM ride\_bid rb WHERE rb.reg\_no = c.reg\_no AND rb.email = %s AND rb.status = 'pending' AND rb.start\_time = r.start\_time) as has\_pending\_bid

FROM ride r, "user" u, car c,model m WHERE r.reg\_no = c.reg\_no

and c.email = u.email

and LOWER(r.origin) LIKE LOWER(%s) and LOWER(r.destination) like LOWER(%s) and r.status = 'in progress'

and c.make = m.make and c.model = m.model

ORDER BY r.start\_time ASC

### Login / Register with Stored Procedure

##### The following is the SQL for register and login respectively

SELECT REGISTER(%s,%s,%s,%s,%s)

SELECT \* from login(%s,%s)

*3.3.2.1 Stored Procedure for login and register*

##### The stored procedure is used so that the password is hash on the database side before inserting into the table, which improve the security factor.

CREATE OR REPLACE FUNCTION register(email varchar(256),contact numeric(8),first\_name varchar(50), last\_name varchar(50),password varchar(256))

RETURNS BOOLEAN AS $$ DECLARE

success BOOLEAN; hashPassword varchar(512); BEGIN

SELECT encode(digest($5, 'sha256'), 'hex') INTO hashPassword;

INSERT INTO "user" VALUES ($1,$2,$3,$4,false ,hashPassword) Returning 1 into success; RETURN success;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION login(email varchar(256),password varchar(256)) RETURNS setof "user" AS $$

DECLARE

hashPassword varchar(512); BEGIN

SELECT encode(digest($2, 'sha256'), 'hex') INTO hashPassword;

RETURN QUERY SELECT \* FROM "user" where "user".email = $1 and "user".password = hashPassword;

END;

$$ LANGUAGE plpgsql;

### Ride History

This allow us to keep track of when is a ride completed. With another table in place, it can help to prevent accidental update. As there are no other query except this trigger that is link to the table.

* + - 1. Trigger

create or replace function audit() returns trigger language plpgsql

as $$ BEGIN

IF NEW.status = 'completed' THEN INSERT INTO

audit\_log(start\_time,end\_time,status,current\_pax,destination,origin,reg\_no) VALUES

(OLD.start\_time,now(),NEW.status,OLD.current\_pax,OLD.destination,OLD.origin,OLD.reg\_no)

;

END IF;

RETURN NEW;

END;

$$

;

create trigger to\_audit before update

on ride

for each row

execute procedure audit()

;

##### Simple Update Query

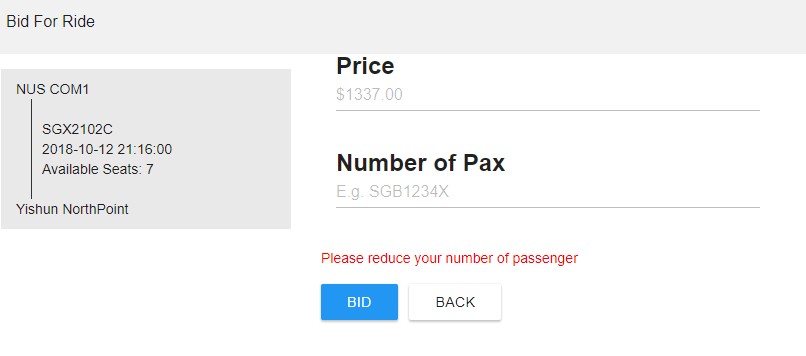
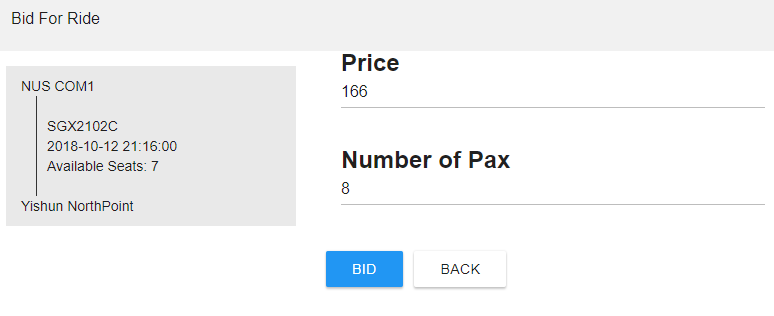
UPDATE ride

SET origin = %s, destination = %s, status = %s WHERE reg\_no = %s AND start\_time = %s

# Assertion

## Rejection of bid

#### When the user is bidding for seats which is greater than current available seats, the system will reject the transaction via trigger



### Trigger behind

create or replace function capacity\_checker() returns trigger language plpgsql

as $$ BEGIN

IF ( SELECT (r.current\_pax + NEW.no\_pax <= m.capacity)

FROM ride r

inner join car c on r.reg\_no = c.reg\_no

INNER JOIN model m on c.make = m.make and c.model =m.model AND r.reg\_no = NEW.reg\_no

AND r.start\_time = NEW.start\_time) THEN

RETURN NEW;

ELSE

RAISE EXCEPTION 'Exceeded maximum capacity, please reduce your number of passenger'; END IF;

END

$$

;

create trigger cap\_check before insert

on ride\_bid for each row

execute procedure capacity\_checker()

;

### Simple Insert Query

INSERT INTO ride\_bid (email,start\_time,reg\_no,no\_pax,bid\_price) VALUES (%s,%s,%s,%s,%s)

## Bid Approval

#### For every successful ride bid that is approved by the driver , there will be a trigger to update the current capacity of the ride and reject all pending bid which exceed the current available seats.

### Trigger as shown below

create trigger approval\_update after update

on ride\_bid for each row

execute procedure on\_approval\_update\_pax()

;

### on\_approval\_update\_pax function

create or replace function on\_approval\_update\_pax() returns trigger language plpgsql

as $$ BEGIN

IF NEW.status = 'successful' and OLD.STATUS <> 'successful' THEN

UPDATE ride

SET current\_pax = current\_pax + NEW.no\_pax WHERE reg\_no = NEW.reg\_no

AND start\_time = NEW.start\_time;

-- The following part update all current biddings which have their number of passenger above the current available seats

-- for example total pax is 4, current pax is 2

-- all bidding which is >2 will be automatically changed to unsuccessful UPDATE ride\_bid rb

SET status = 'unsuccessful' FROM ride r, car c, model m WHERE r.reg\_no = rb.reg\_no

AND r.start\_time = rb.start\_time AND r.reg\_no = c.reg\_no

AND c.make = m.make AND c.model = m.model

AND rb.reg\_no = NEW.reg\_no

AND rb.start\_time = NEW.start\_time AND rb.status = 'pending'

AND rb.no\_pax > (m.capacity - r.current\_pax); END IF;

-- return NULL as trigger is called after update RETURN NULL;

END

$$

;

## Rejecting Car Updates

##### There cannot be any updates to the car entry because if there is a ride in progress with full capacity, changing to another car model might result in not enough seats for the passengers.

### Update Query is shown below

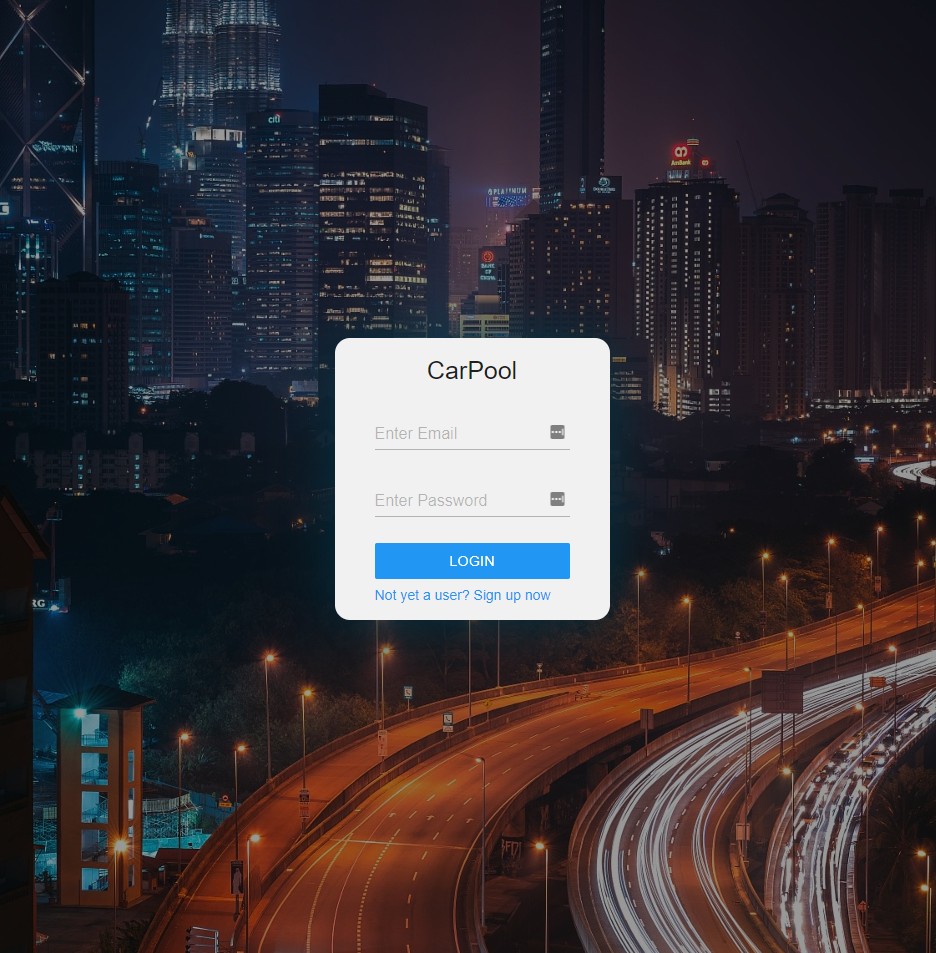
UPDATE "car" SET "make" = %s, "model" = %s, "colour" = %s WHERE "reg\_no" = %s AND NOT EXISTS (

SELECT \* FROM "ride" r WHERE r."status" = 'in progress' AND r."reg\_no" = %s

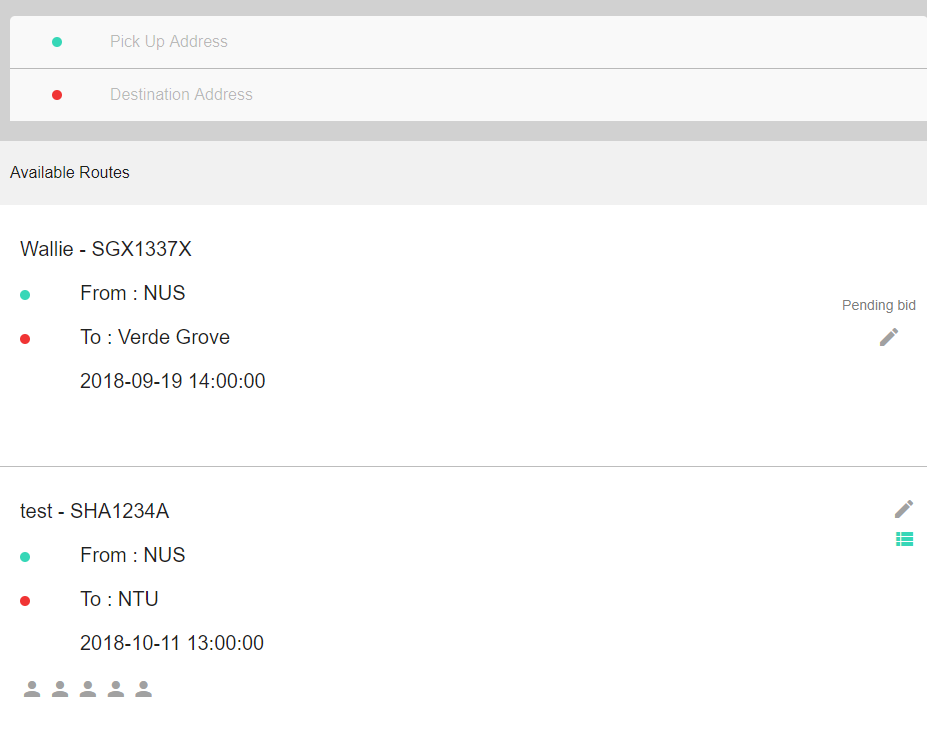
) RETURNING "reg\_no"

# Screen shots

## Login Page:



## Index Page/ Home



## Menu

