RideNShare - Capstone Final Report

Linked to our demo:

Link to our demo: https://ridenshare.s3.amazonaws.com/image/12-5-1080p.mp4
GitLab - https://code.vt.edu/cs5834-fall-22/ridenshare

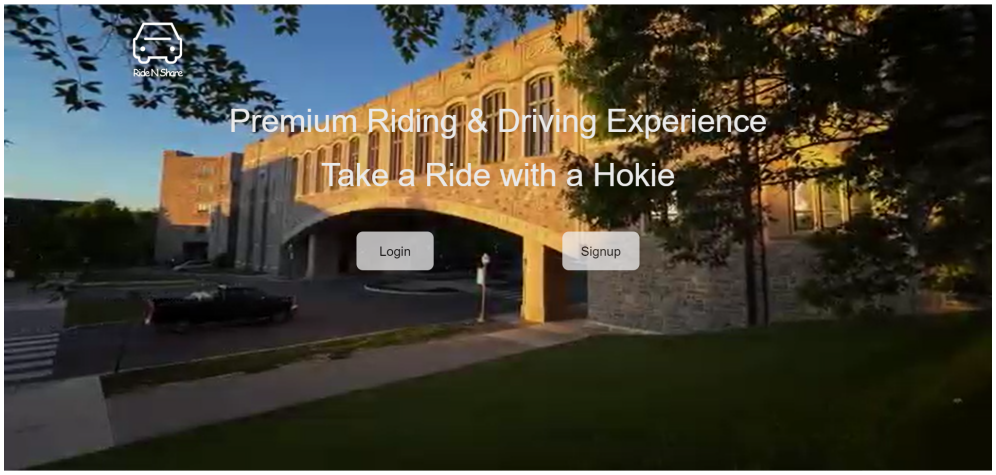
Product Description

We have developed a website for VT faculties and students, called RideNShare. Users can offer a ride or get a ride using our product. Our product solves some safety problems of traditional ride apps. All users who use our products need a VT email to register to ensure that users are our students or faculties. Besides this, users can make some money by offering rides for someone in need, and users who do not have a car can get help in need. Last but not least, sharing a ride is both economical and environmentally-friendly, improving the utilization of vehicles and reducing road congestion.

Product functionalities

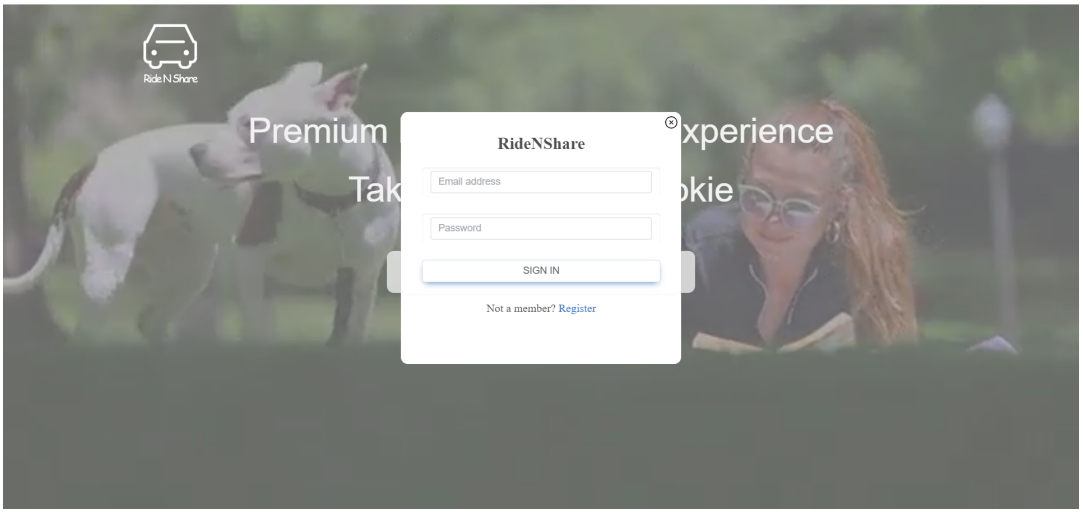
Login and Signup

The screenshot below is our home page. Users can login or sign up on this page. We are using Virginia Tech email authentication to make sure that each user is a real VTech person.

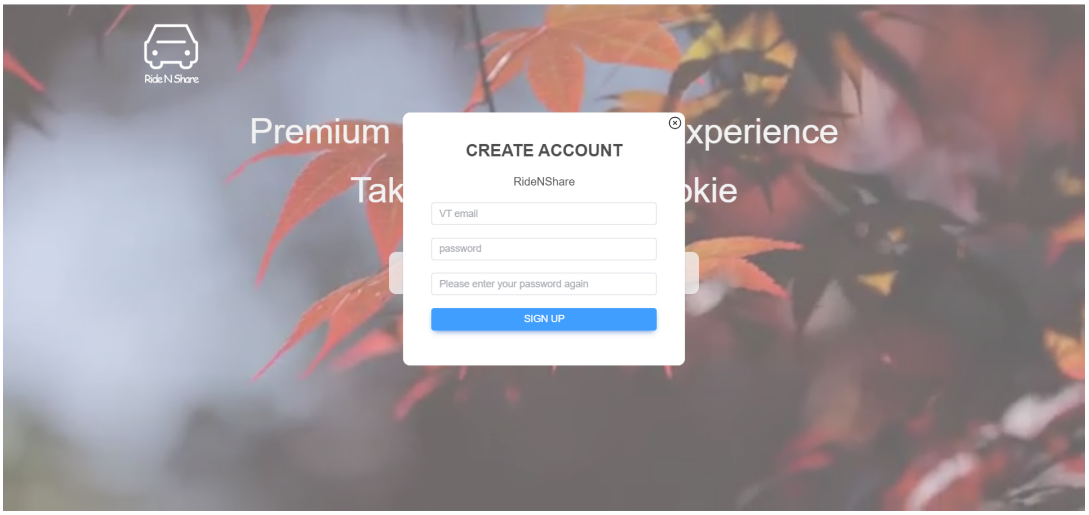


Home Page

A window will pop out if users click one of the login or sign-up buttons. Users are able to login or signup on the window.

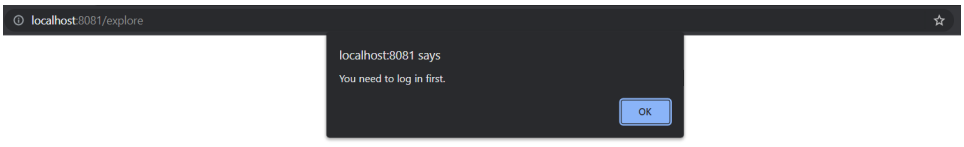


Login pop-out window



Sign-up pop-out window

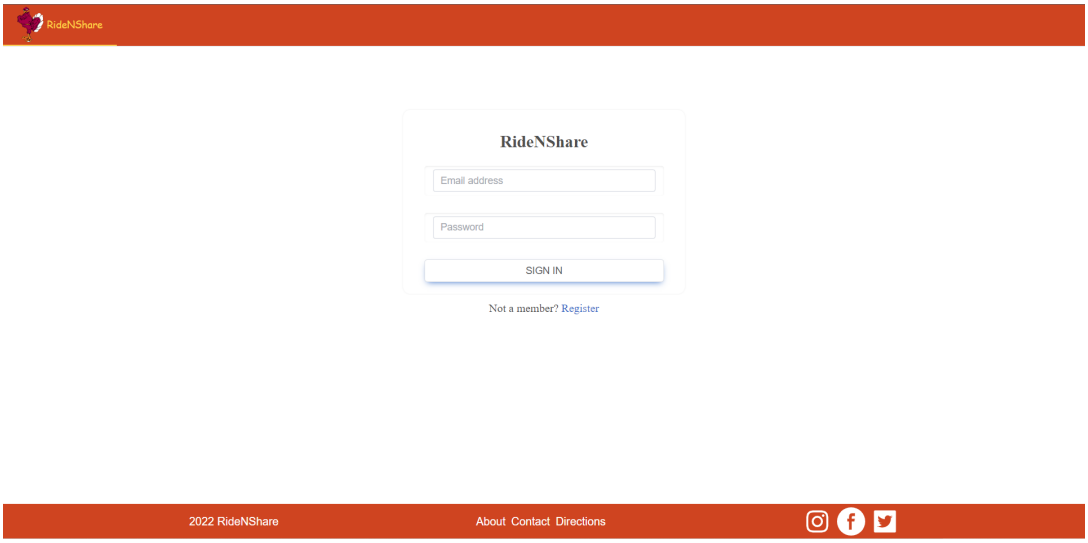
Login Process



Navigation-guards

To prevent people who have not logged in from accessing other pages (such as the post explore page) by typing the address directly, our navigation-guards will redirect them to the login page after hitting the “ok” button. Apart from login, signup and information page, other functionalities of our web application are not exposed to anonymous users.

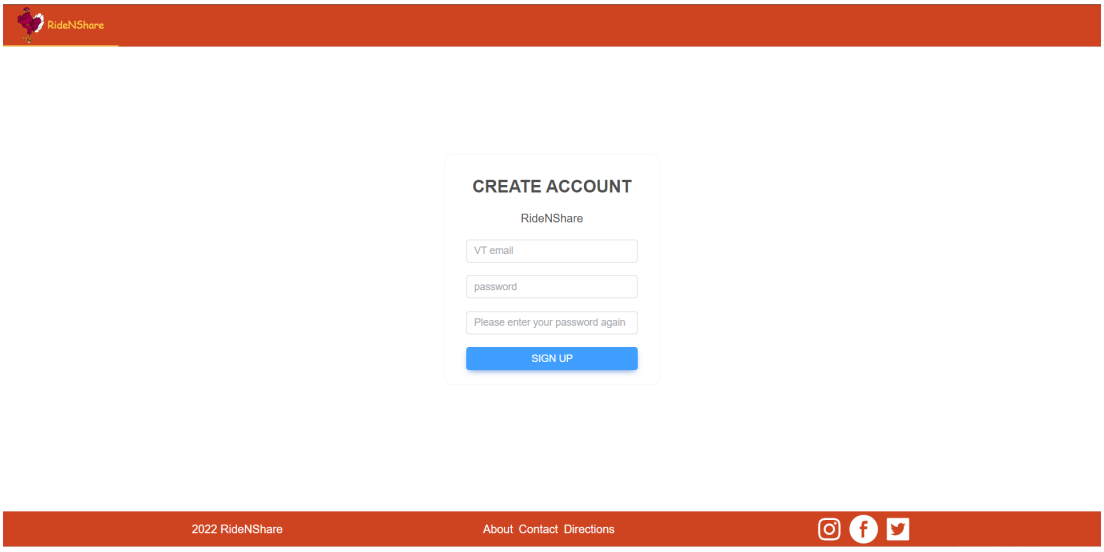
After clicking the login button, the website will lead the user to the login page.



Login page

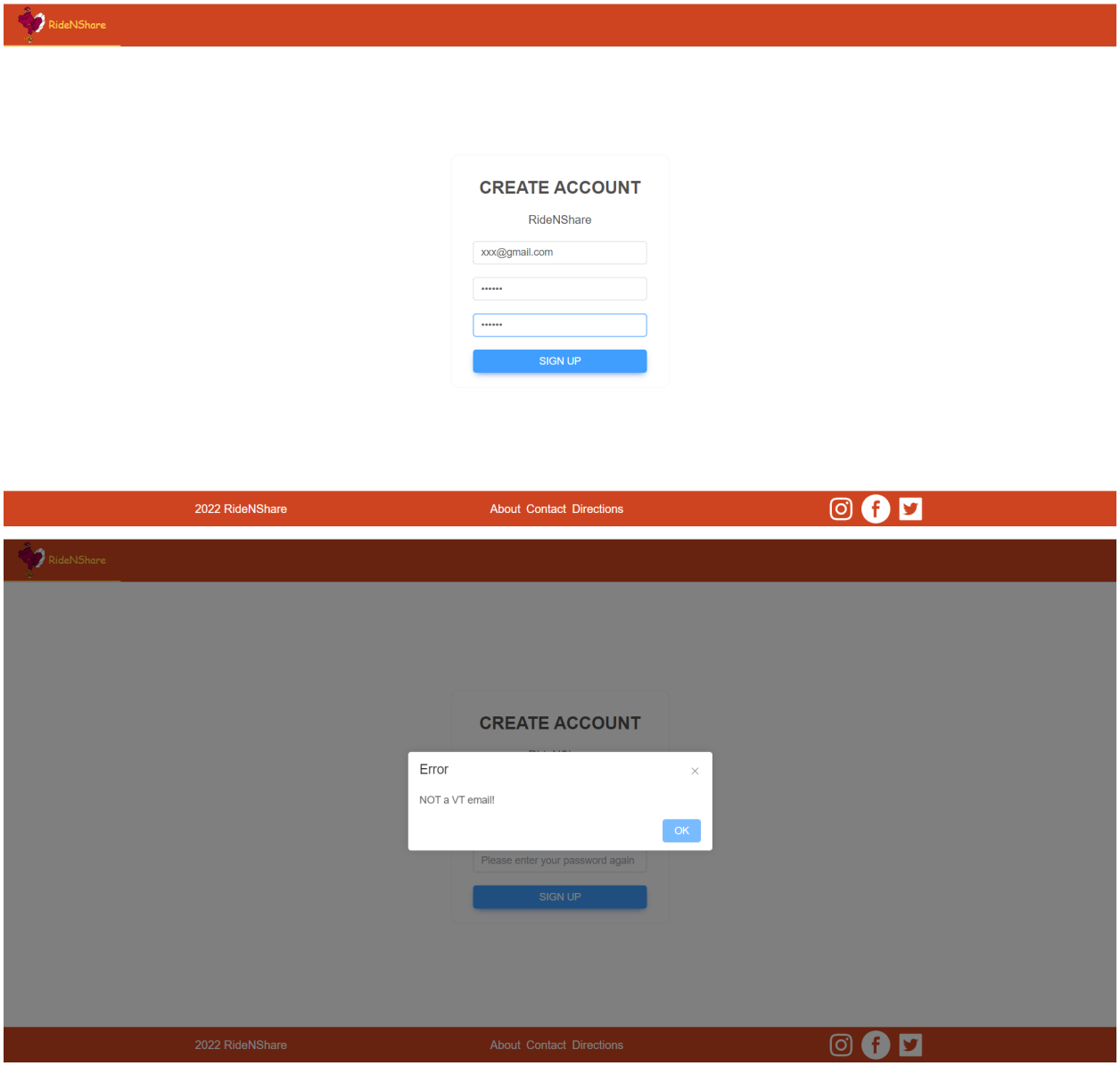
Signup Process

If users are new to our website, they will have to sign up using their Virginia Tech email. The signup page is attached below.



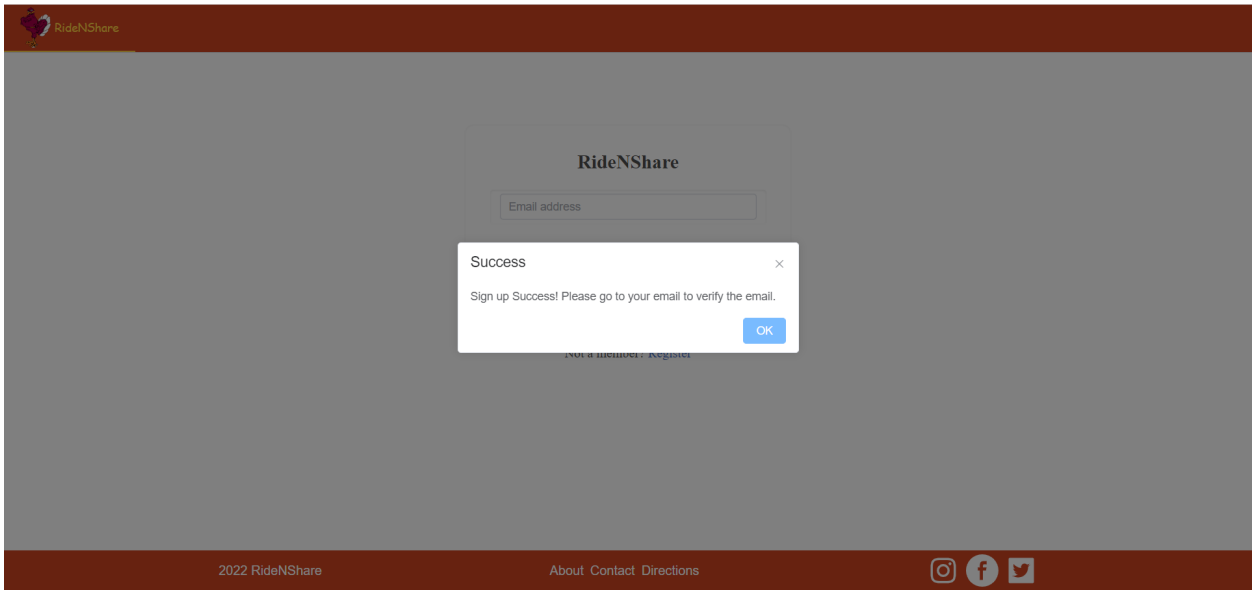
Signup page

We only accept VT email. Users will receive a notification that says VT email address only.



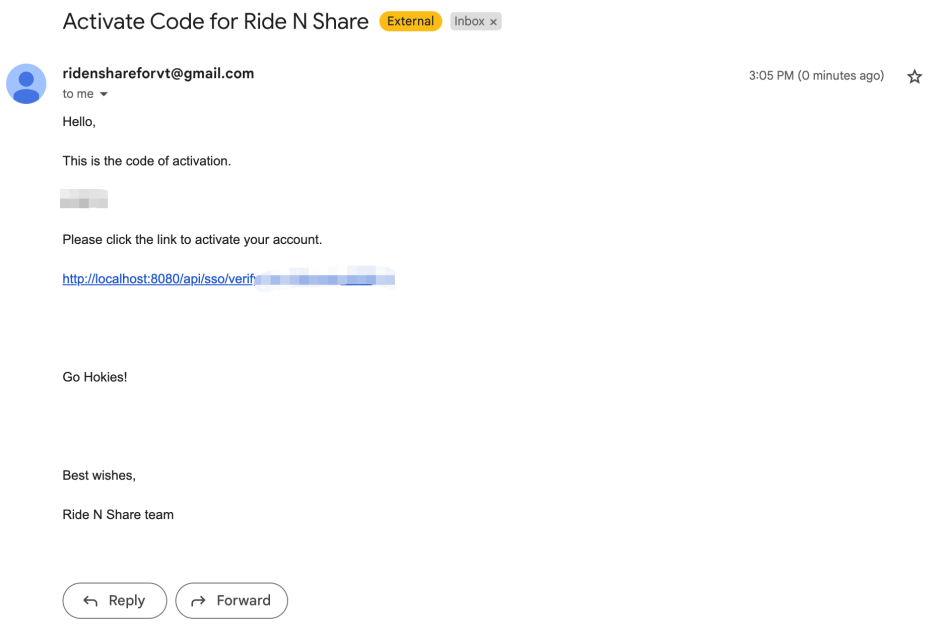
Only accept VT email

Once the users sign up with their VT email address, they will receive a notification that says sign up successfully. However, verification is necessary in order to explore our main functionalities.



Need to verified first

The screenshot above shows the verification link. Users will have to click on that link to activate their user accounts. This is a way to ensure that the VT email address is real and valid. The link will expire in 2 hours.

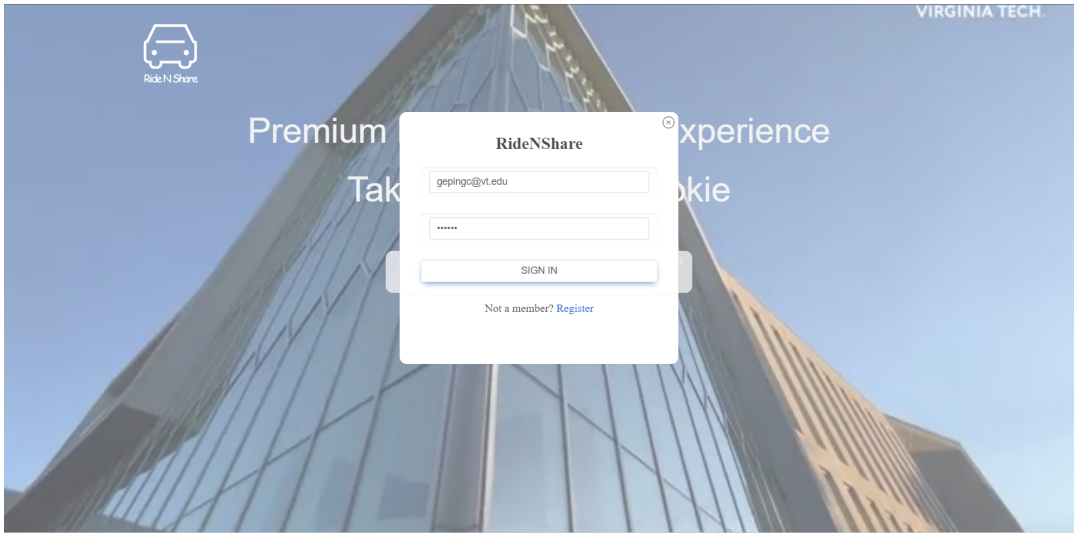


Activation Email page

The screenshot below shows the current user account has been activated. We grant this user authorization to access our page without limitations.

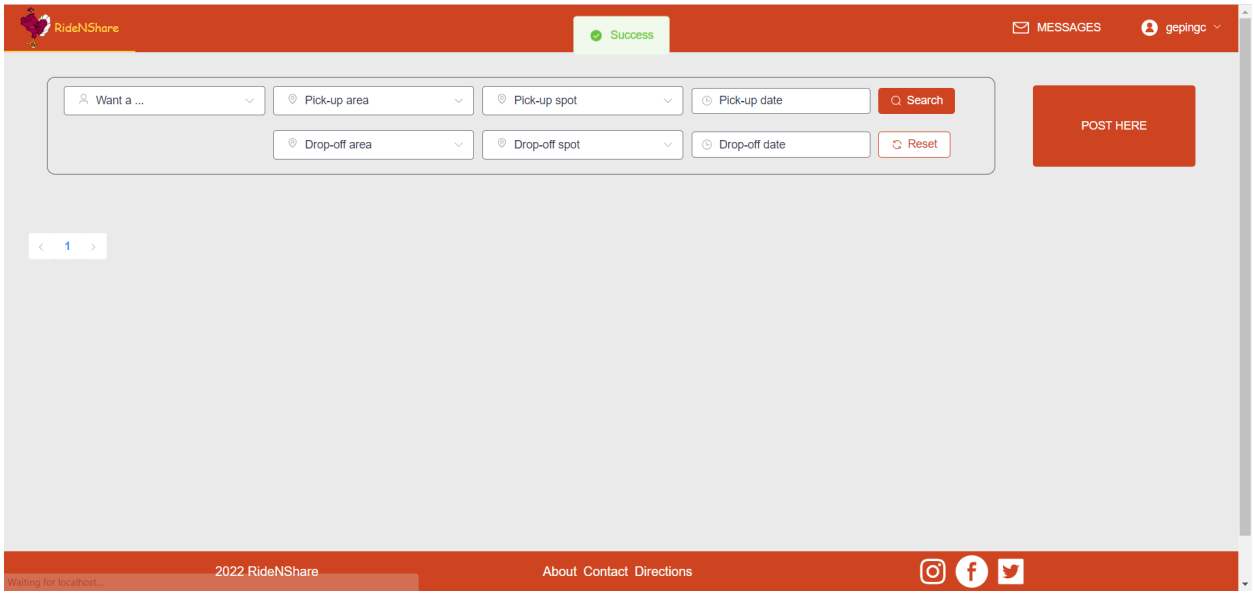
Your account has been activated! Please return to login page to log in.

Successfully Activated



Successfully registered

Now the activated user can hit the “login” button and login with the email address and password entered when registered.

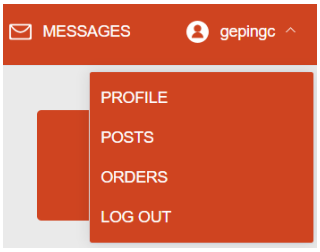


Login Successfully

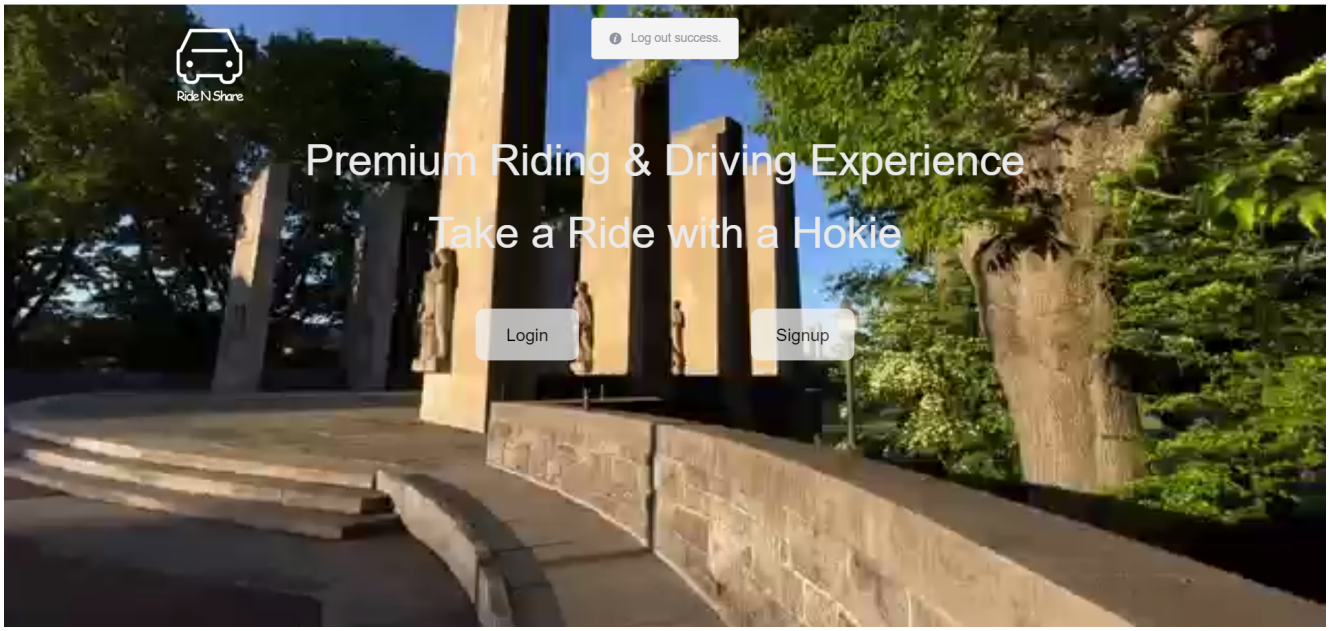
The users will receive a pop up notification showing login successfully. Clicking on “ok” will lead them to the explore page.

Logout Process

Users can log out by hovering over the username on the upper right corner of the page and clicking on “LOGOUT” to safely clear the session and back to the home page.

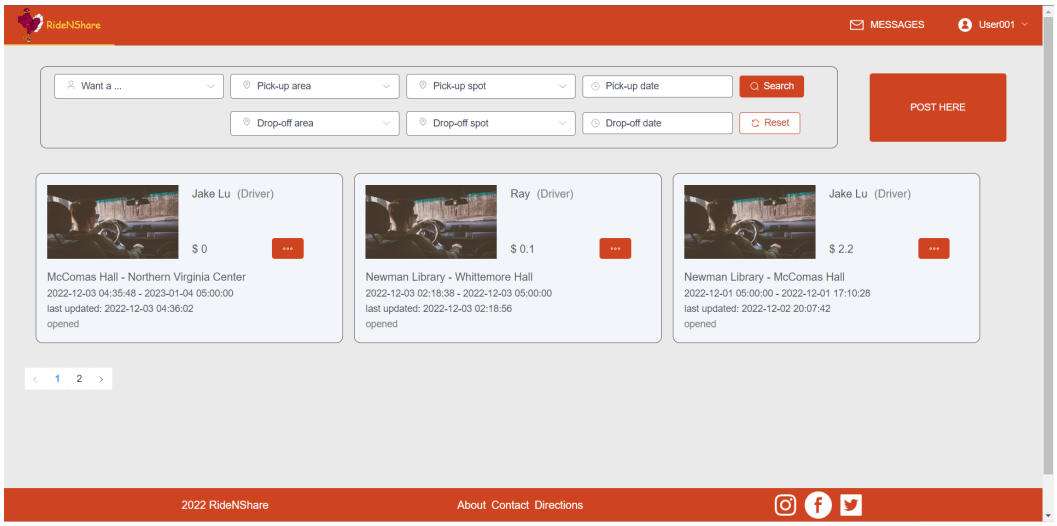


Log out

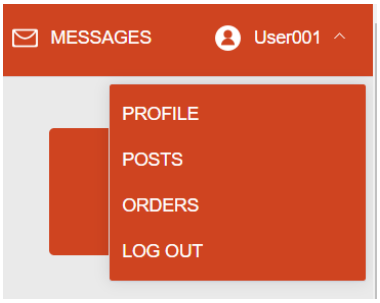


Back to Home page

Profile

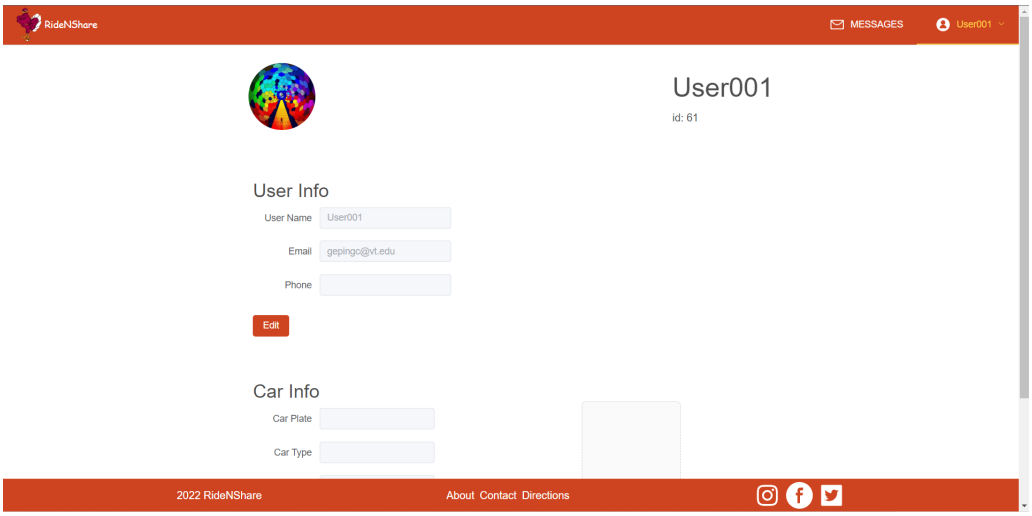


Explore Page



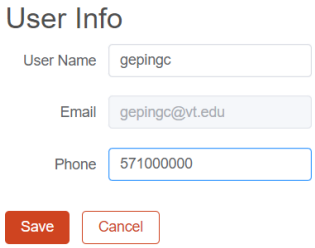
Link of the Profile System

On the explore page, we have the username displayed on the upper right corner of the header. When hovering over the username, users will see four options. Clicking on “PROFILE” will lead the users to the profile page to update their personal info.

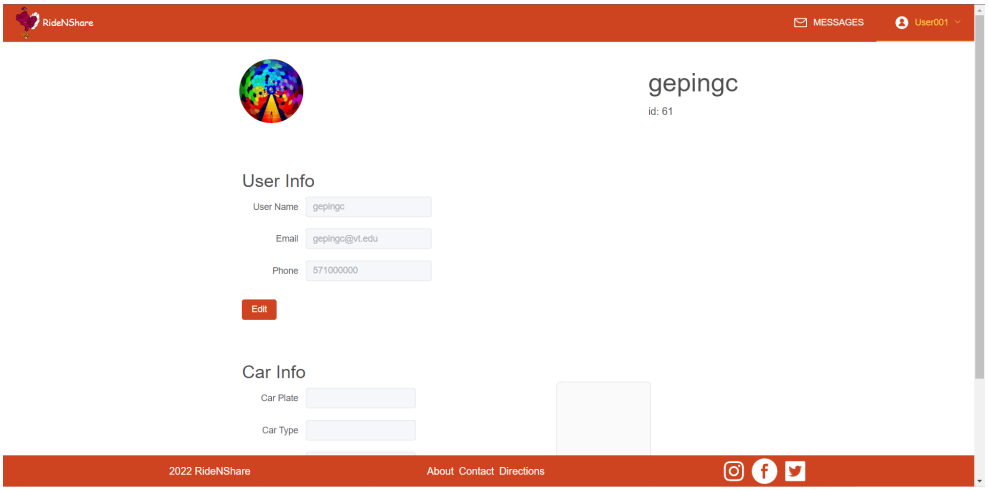


User Profile Page

Users can update their username, phone number as well as adding a car to their profile by clicking the edit button on each section.

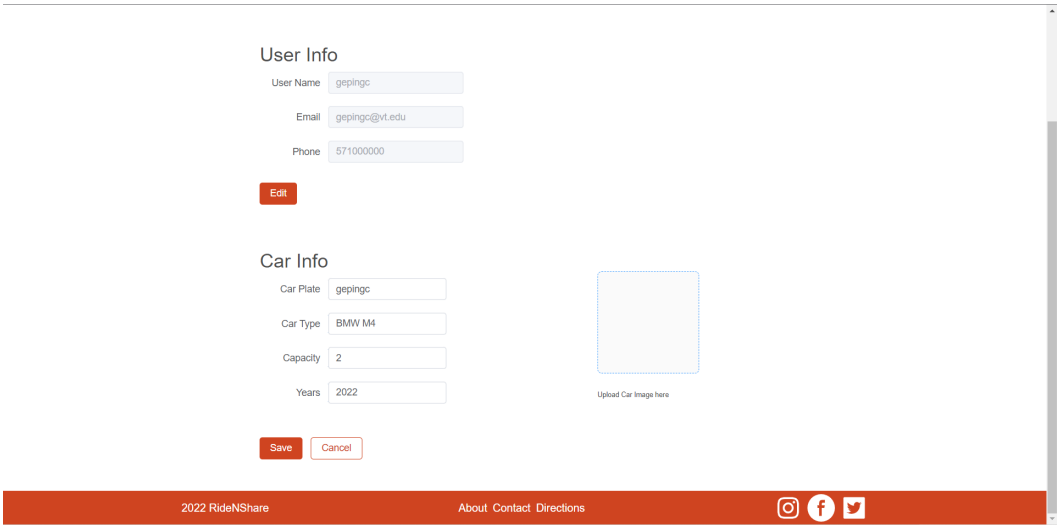


Edit user profile

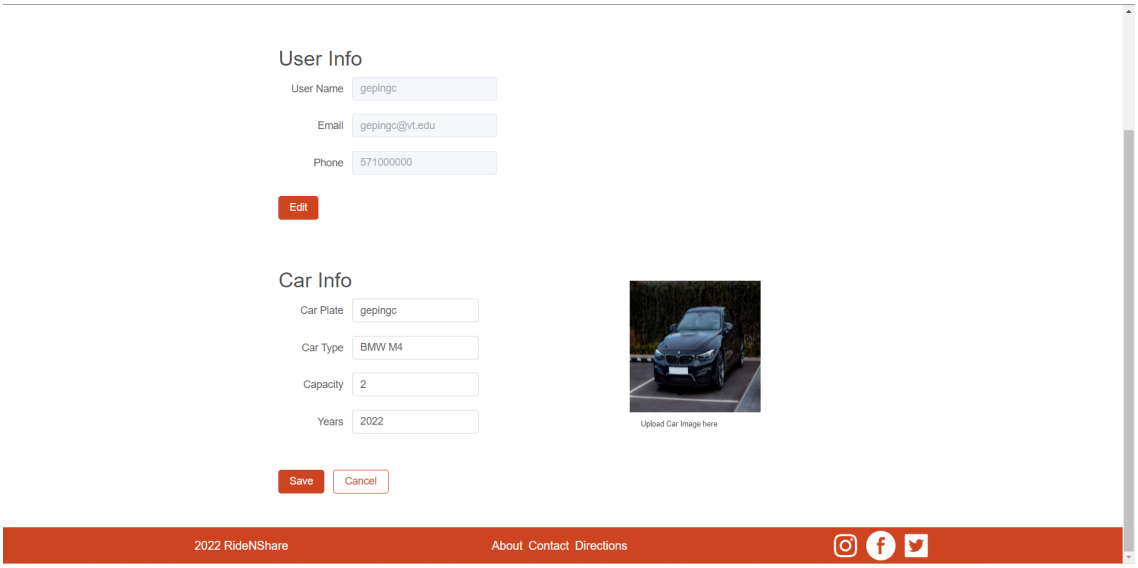


Profile changed

We can see that the message changed after we edit the profile.



Edit car information

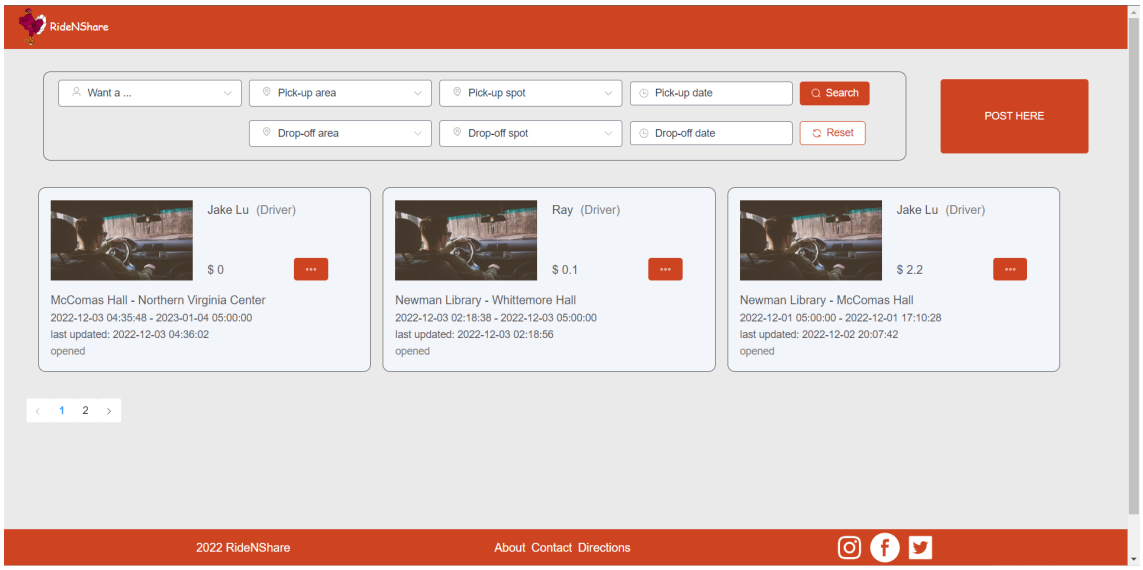


Upload car image

Users can edit their car information, and upload their car image to register as a driver.

Post System

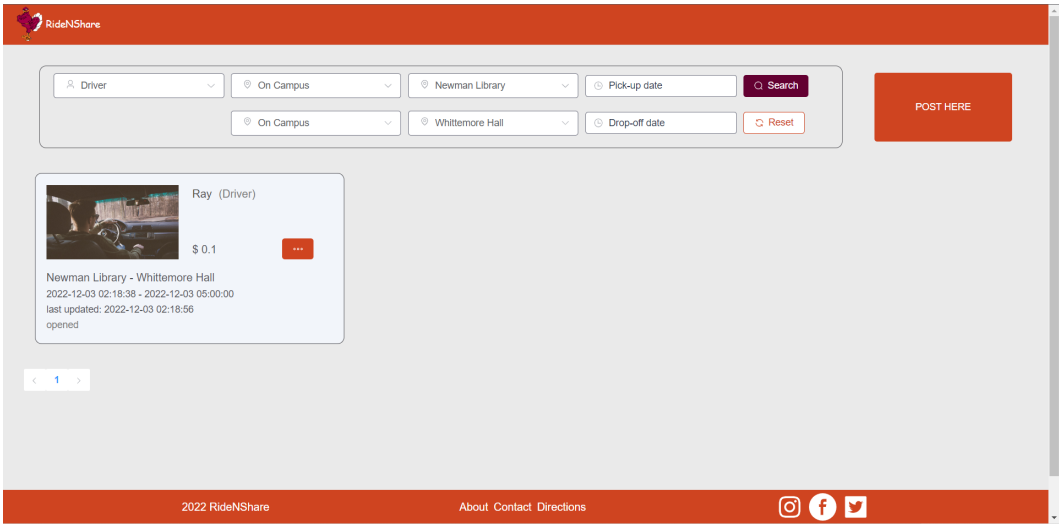
On the explore page, users can search the post they want or create a post.



Explore Page

Search

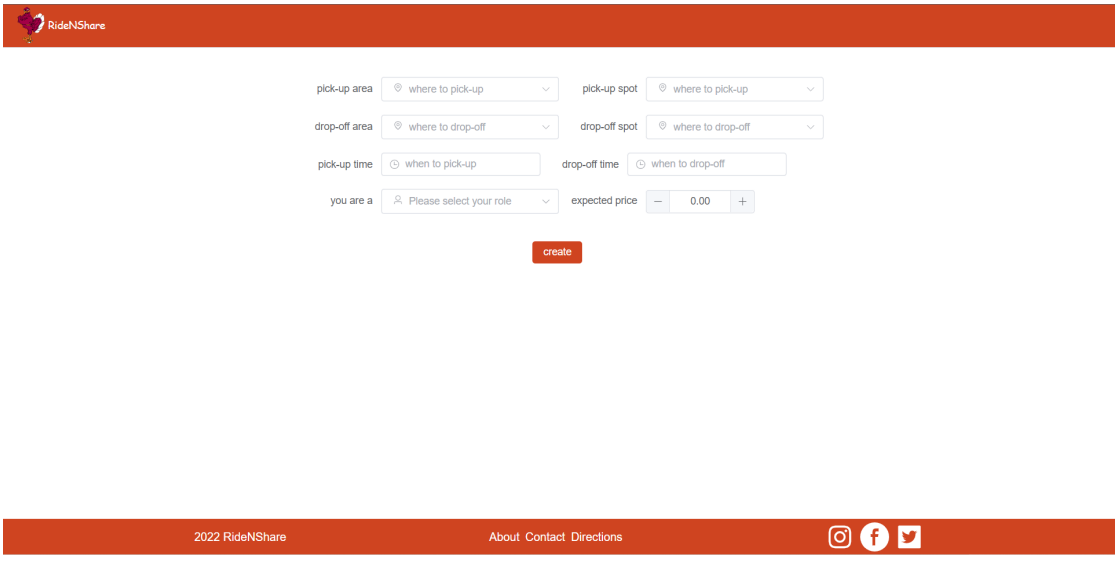
Users can choose certain filter criteria to search for a ride. After hitting the “search” button, the result will be displayed according to the filter.



Post Page According to the Filter

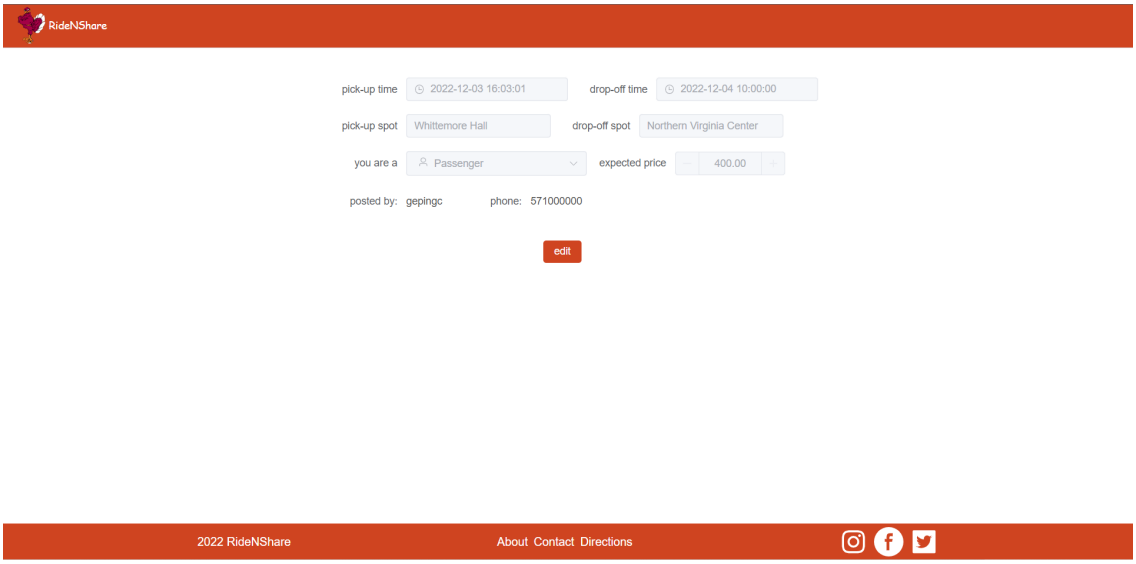
Publish new post

The screenshot shows the search results. If no result meets the criteria, users can simply create a post to express their needs by hitting the big “POST HERE” button. And the Post page will be shown.



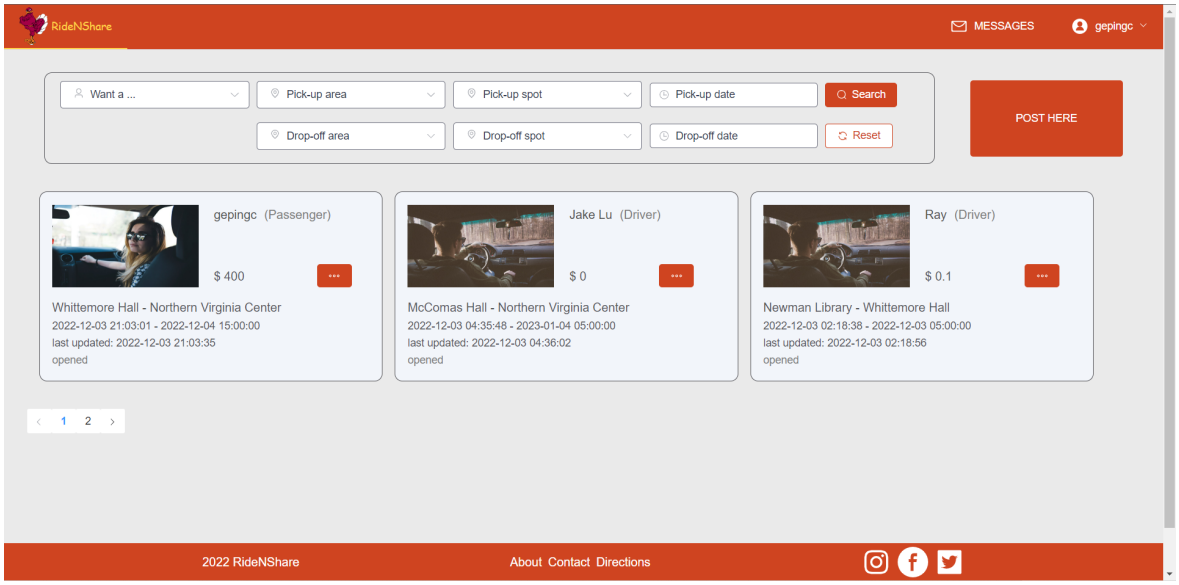
Create Post

The screenshot below shows the post the user just created. This post will also be displayed on the explore page and can be reached through searching by other users. We also support editing the post. Users can hit the “edit” button to update the information such as location, time and expected price they are willing to pay or take.



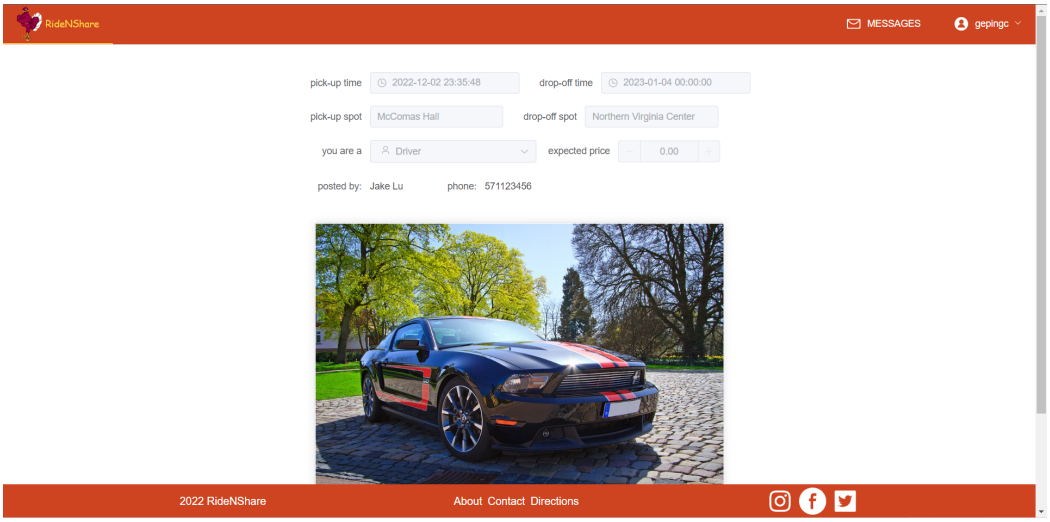
Post Detail Page

Sorting



Posts sorting based on updated time

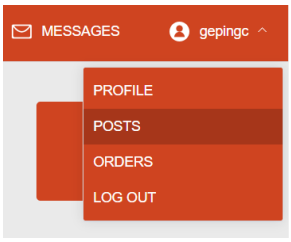
The default order of the posts are sorted according to the last updated time of the post to make sure the posts are potentially valid and are explored in a timely manner.Users can click on the “...” (More) button to view the post detail. Post detail page will be shown after clicking the button.



Post Detail Page

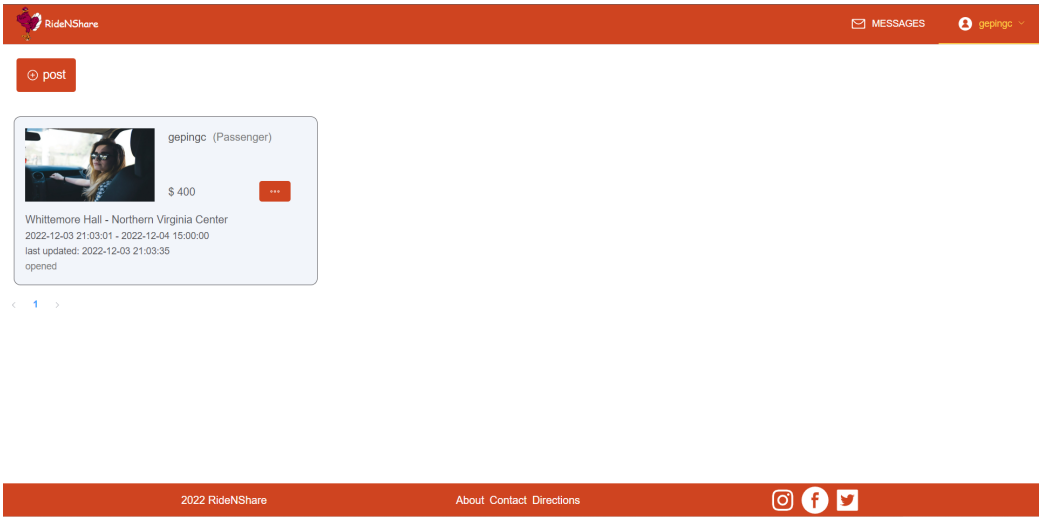
My post

Users can also access their own posts by hovering over the upper right corner of the page and clicking on the “POSTS” button.



User’s own post

All of the posts whenever it has been opened, completed or canceled will be shown here. By clicking on the “...” (More) button, users will be able to access their own post detail to view or edit this post.

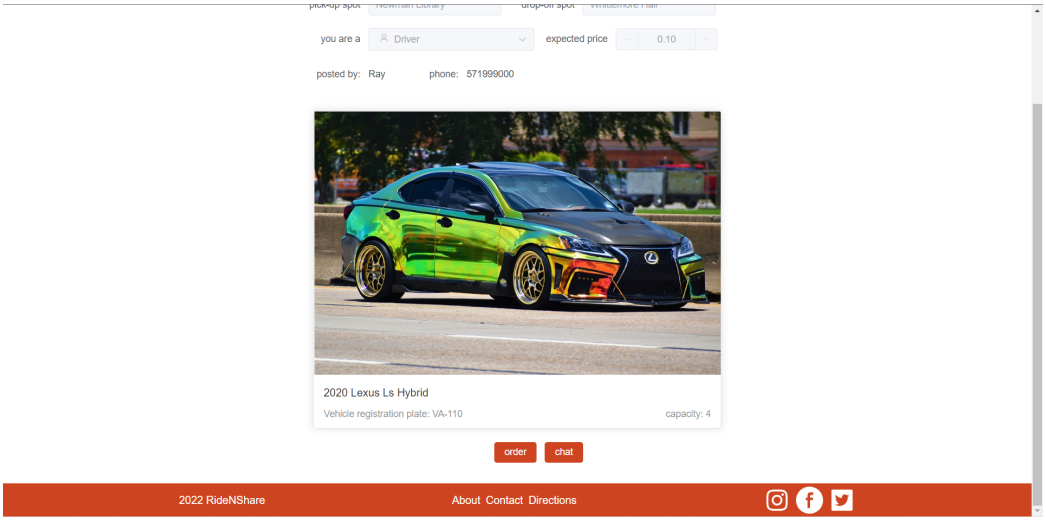


My Post

Chat

There are two ways to view the chat page:

1. On the post detail page, users can click the “chat” button to chat with the other user who publishes this post to reach an agreement before placing the order.

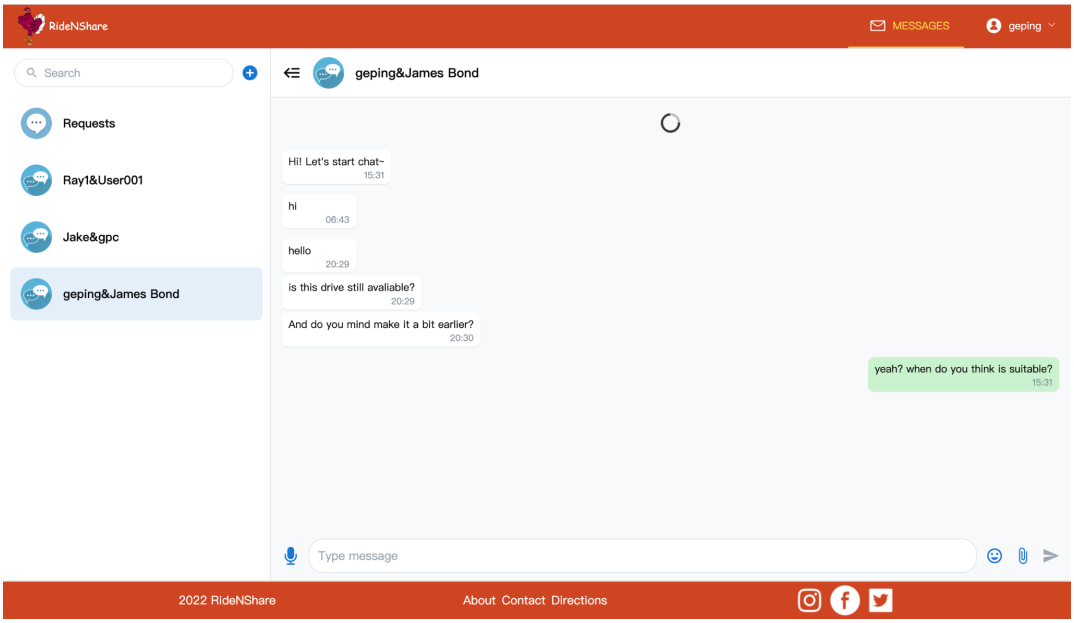


Chat with the driver/passenger before making order

2. Click the “MESSAGES” button in the navigation bar on the top

View all chat rooms

Both of these two ways can take you to the chat room page. The screenshot below shows chatting between two users: geping and James Bond.

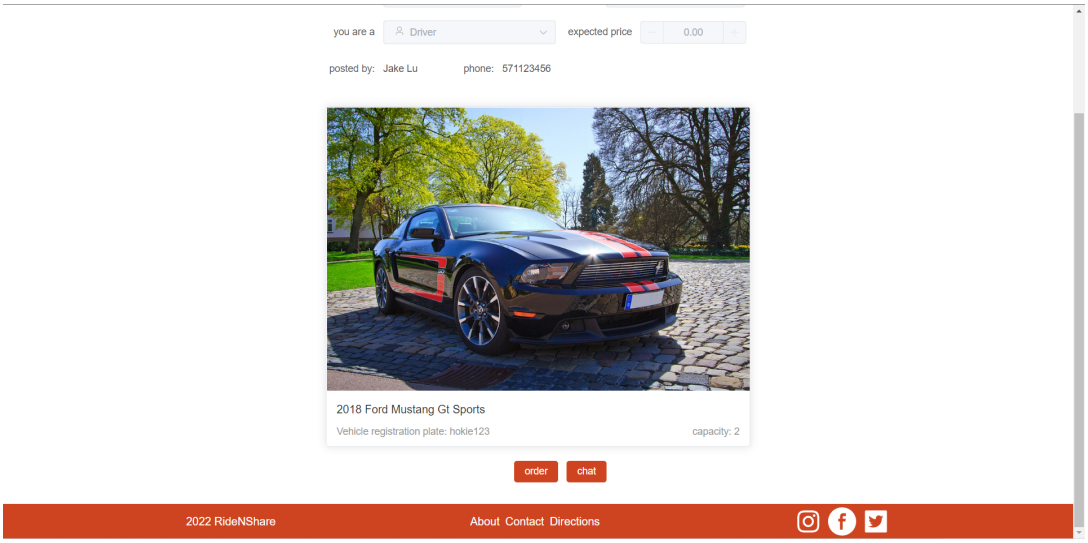


Chat room

When the user is redirected from the post detail page and he/she has never chatted with the post owner before, a new chat room will be generated between two users, and the welcome message “Hi! Let’s start chat~” will be displayed on the chat window.
If the two users have chatted before, the message history will show on the chat room page.

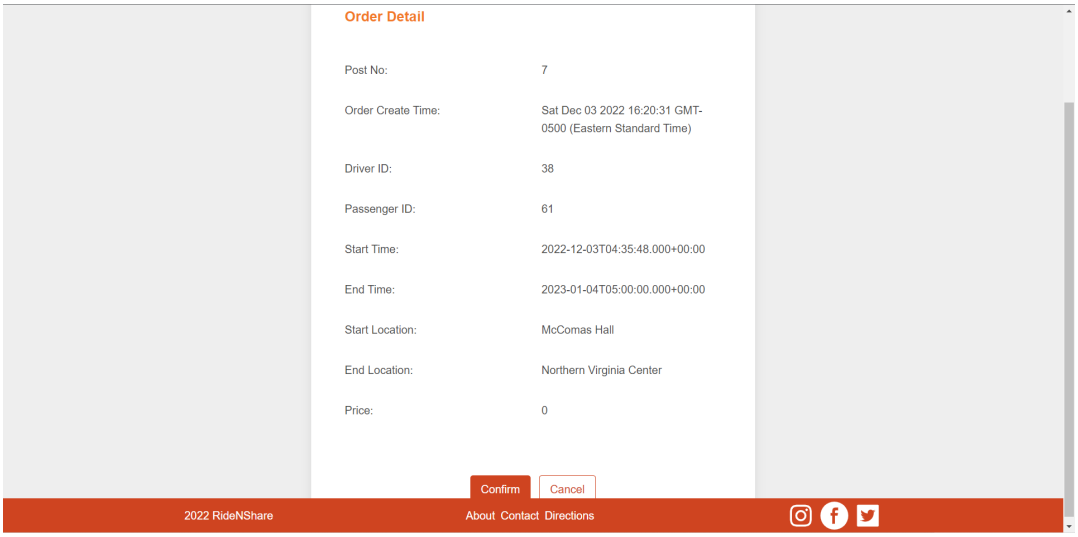
Order System

On the post detail page, users can place an order if they think they want the ride by hitting the “order” button.

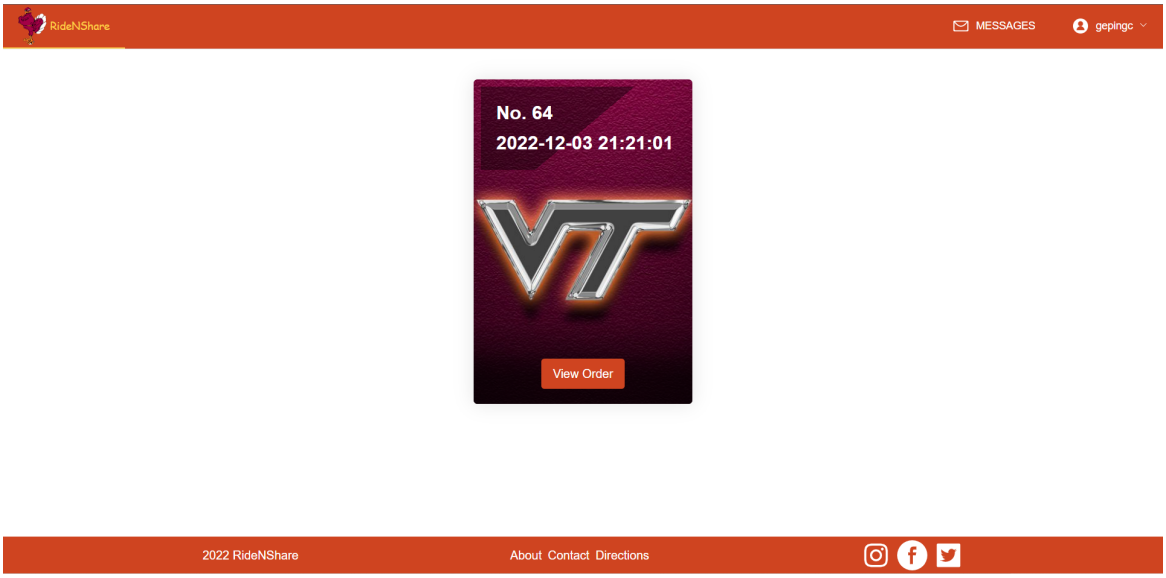


Order by clicking on the post page

After clicking on the order button, the order will be generated. The website will take you to the order details page, users can see the details of the ride. When users are on the ride, they can confirm their order after clicking on the confirm button. If the order is confirmed, it will not be able to be canceled.

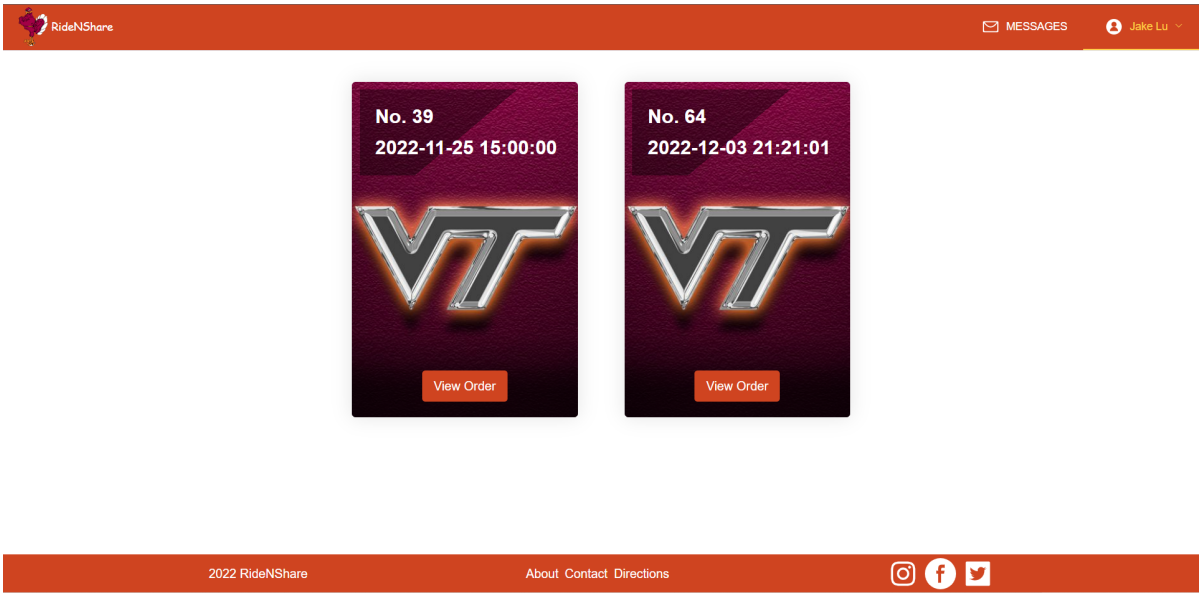


Order detail page



My order

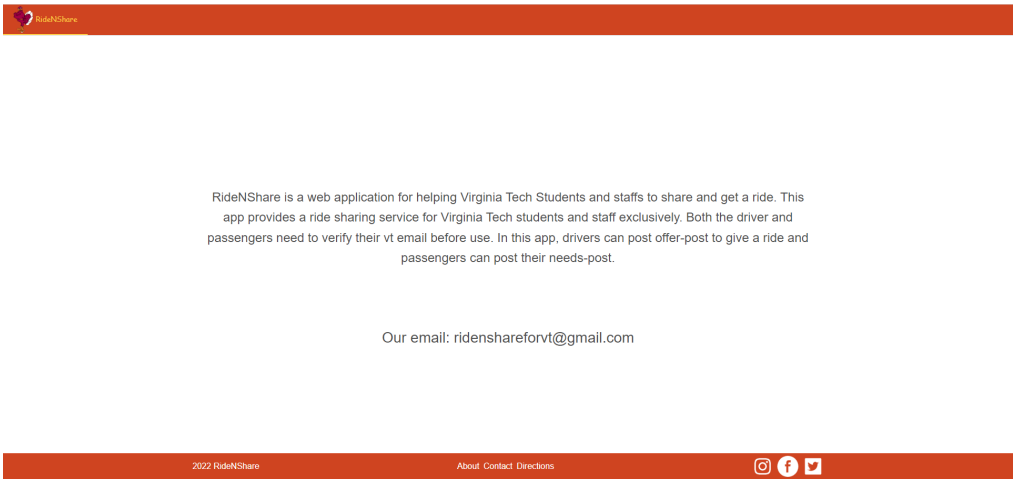
After placing the order, the other user who posted this will be able to see the order by hovering over the upper right corner of the header and clicking on the “ORDERS”.



Order page

The other user can also view the order detail by clicking on the “View Order” button.

Information Page

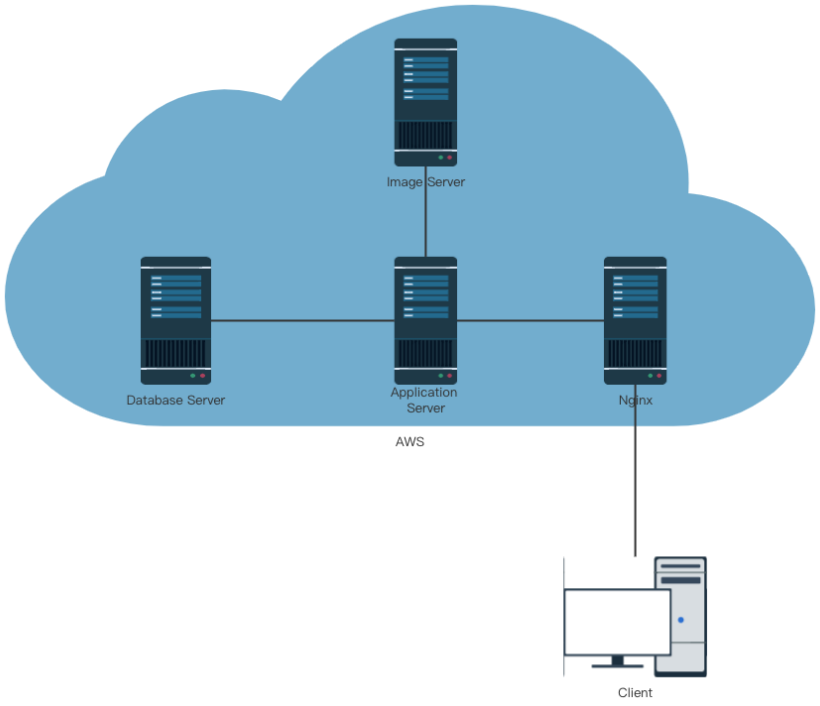


About page

This page provides a simple demonstration of the application and an administrative email has been given to the user, too.

Design description

Architecture



Architecture

Our system is deployed on the AWS platform. We use the AWS RDS as our database server. And we choose MySQL as our database. In order to relieve the pressure of the database server, we use the AWS S3 server to store all the static resources. And we use the Elastic Beanstalk service on AWS to deploy our backend application. In order to balance the request, Nginx is used here as a Proxy server to balance the requests.

Technologies used

	Technologies
Frontend	Vue.js, ElementUI, HTML, CSS, Typescript, js, Session, Cookie
Database	MySql
Backend	Spring boot, Java, Node.js, Mybatis
Tools & Platform	Design Phrase: Figma UML Tools: Astah UML Version control: Gitlab Project management: Jira Deployment: AWS. AWS S3 as image Server, AWS RDS as database, AWS Elastic Beanstalk as server. Nginx as proxy server for request balancing

Functionality Design

General Problem Design

Session Technology: We used the session to store the user's information. But when we deployed on cloud, a problem occurred because the session is stored in the server side and there are many servers provided by AWS service, this will result in cookie loss. In this case, **Spring session** technology is used to solve the problem, all the sessions are stored in the database, and all the servers can share the session data.

Interceptor: It is used to intercept requests, and is supported by **Spring Framework**. For all the requests that require a login state, the interceptor will intercept the request and check if the user is logged in or not. The interceptor will first check if the session form the user exists or not, and if the session is expired or not. If the session is valid and not expired, the system can know the current user of the client.

ThreadLocal: It is used to relieve pressure on the database, it is a multi thread technology provided by **Java**. If the number of the users is not large, the product works fine. If many users are using our service, the system would continue accessing the database for user information, this will result in great pressure on the database. Besides this, we may face the situation that many users are changing the same variables. The ThreadLocal technology in Java can be used here to manage the user in each thread using a map. In this case, every time when a user needs to access a resource that needs to be logged state, the server can know which thread belongs to each user.

AOP Technology: In order to better debugging, **AOP technology** is used here for logging. Spring framework supports AOP to solve this problem. We only need to define a new annotation, and apply it on each function that needs a log. And this will help to reduce a lot of code.
Navigation Guards: It controls the visit of users at the front-end of the application. With navigation guards, users will be forbidden to visit any page that requires logged in state. This module controls the users' visiting from the front-end and notify people when they are logged out and try to visit pages that need logged in .

Profile System
In this system, Users can see their detailed information on the profile page, and also register a car to be a driver. The system gets the user information through session, accessing the database for more information and passing it back to the frontend.
Upload: On the profile detailed page, users may want to upload his or her image of car and avatar. When users increase, the server may not suffer this large traffic of access. In this case, we use an image server to balance the traffic of our database server. We use the AWS S3 as our image server. What we need to do is to generate the key to create a credential of the AWS service and use the api provided by the S3 server. When the image is uploaded, the service will return the url of the image, we can access the public image through this url.

SSO System
The SSO system is in charge of processes of login and registration. It is a back-end system which supports the login and registration logic. It generally has five functions. It can add new users to the database, which supports the registration component. It can generate the activated link and email context, then send an email with an email sending module to the user's mailbox so that the account can be verified. In addition, it controls the login and logout logic.
Login Component: The Login Component is being used on the homepage and login page. It requires users to fill in their account email and password. Once the log-in button is clicked, the data will be posted to the backend of the SSO system, and if the user is existing and valid in our database. If the account has not been verified, the system will not let the user login but send a notification. After the user has logged in, the system will store the user's information into the Session storage at the front-end so users' name can be shown on the Navigation bar and be got at anywhere else.
Registration Component: Registration Component is being used on the homepage and registration page. Users are required to use theri vt emails to register an account of RideNShare. The front-end of the component will verify the format validation of the vt email. Also, users need to input their passwords twice to avoid mistakes. The component will verify this information and also make sure the password is at least 6 characters long. Then this new user will be posted to the backend of the SSO system, temporarily setting the activated status of the user to unactivated. The SSO system will verify this information, ensuring that the account has not been created before (if the account has been activated). When the user information passes the SSO system verification, the SSO system will give a pass signal for the registration component. The registration component will then let the user know the registration is successful.

Post System
In this system, Users can see all the in-progress posts sorted by the last update time in descending order on our explore page. Users can also search for the post they want by selecting the filter

criteria, such as pick-up and drop-off locations, time and the post type (what post they are looking for, e.g. driver's or passenger's post). Besides, users are able to see their own posts and edit the post. The system gets the search criteria as the input, accessing the database for more information and passing it back to the frontend.

Displaying posts

Users can get all the in-progress posts directly from the explore page once they log in. The client will asynchronously send a request to the server. Then the backend will process the request, query all the posts from the database, get the car information, encapsulate a vo object suitable for display, handle pagination and pass the paging object back to the client.

Searching

Once the explore page is created, the client will asynchronously send a GET request to the server to get the area list. Then the backend will pass back all the areas stored in our database. Users then will be able to see the areas options when clicking on selecting pick-up or drop-off areas. Once the users selected an area, the client will send another GET request with the area ID as the request parameter (path variable) to the server to get the locations of that area. The backend will retrieve the related locations stored in our database by a multi-table SQL query, accessing the area and spot table, as well as the intermediate table named area_spot_map. To make it more efficient, we write code to store primary keys in a List structure and perform a single SQL query to avoid too much database IO. Once the users have entered part or all of the filter criteria (locations, time, post type) and hitted the search button, the client will send a final POST request to the server. The request body is a JSON object. Then the backend will get all the fields of that JSON object, ignoring the empty and null values, querying the database, encapsulating a list of post vo objects, do pagination, and pass the paging and post information back to the client.

Post Detail, Creating and Editing

During the frontend development, we reused the post detail page. The page can serve as a post detail displaying page, a post create page or a post edit page. We will check the post status as well as the url to decide which section to display (whether it is a driver's post with a car or a passenger's post without the car, whether the form is disabled or not). As for creating and editing posts, once the form has been filled by the users, the client will send a post request to the server. The JSON object contains the post detail information, such as pick-up and drop-off locations and time, as well as the post type (which role the user wants to be, e.g. driver, passenger) and the expected price. Then the backend will access the post, user and car tables, and give back a post detail of the object containing the information users need.

Order System

Order Component: For the Order component, it receives parameters once a user chooses a post and decides to make an order. The post id is used to perform a multi-table associated query for the Post, Driver, Passenger, and Car tables. Then, put that post information into a new order. After insertion, a new order will be created and shown on the Order page with other orders belonging to the current user.

Communication System

In this system, Users can create a new chat room with the other user, type messages to other people and view their chat history.

Create new chat room:

When clicking the “chat” button in the post detail page, the id and name of the post owner will pass to the chat page through the router path. The system gets the current user’s id and name through session, and gets the other user’s information through the url, and stores all the information in a JSON object. Then the client will asynchronously send a POST request to the server. The server uses two user’s id to search if there is an existing chat room between them. If this room does exist, nothing will change, otherwise the database will create a new room item to store the information in the room table.

View chat room list:

When the current user is redirected to the chat page, the system gets the current user information through session, accesses the room table in the database, and searches all rooms that related to the current user’s information, passes the rooms list back to the frontend and shows the names of rooms as a list on the left side of the window.

Chat messages view:

When choosing one chat room by clicking this room on the left side list. The client will asynchronously send a GET request with the unique id of the current chat room to the server to find all related messages that store in the message table, then all the related message information will be sent back to the frontend. If the sender id is the same as our current user id, the message content and send time will be displayed on the right side of the chat room view, otherwise this message will be displayed on the left side because it was sent by other people.

And if the messages list we get from the backend is empty, then a welcome message will be displayed automatically on the chat room view “Hi! Let’s start chat~”.

When a user types a new message, the chat room view will show new messages, and the message content, send time, sender name, sender id and current room id will be stored in a JSON object, and the client will send a POST request to the server. This object will be stored in our message table with all the information it contains.

Database Design

Retrospection

During the whole semester, we built this full-stack project from scratch, with full documentation. The technologies we chose were new to most of the group members so learning has always been the process taking most of the time.
As course requirements, Using project management tools or platforms like Jira to document the project is also a new experience for us, which leads us to an industrial consideration of the project design and maintenance. Based on the sprints spirits, we were supposed to develop the whole project with the subdivided sprints. The art of managing a proper sprint size and deciding the sequence of the implementation of the tasks is good knowledge for our future careers.
During the development stage, we showed our knowledge of coding and especially database design, as we decided to use relational databases instead of the currently popular choice, the non-relational database, with the consideration of query times and the manipulation of the Multi-table Associated Query. The relationship between each table and the attributes inside was well designed in the early stage so that we avoided making big changes to optimize the project in the very late stage.
Apart from that, we decided to use Spring boot as the backend supports because its (IoC) inversion of control and (DI) dependency injection help to decouple the project. We used Vue.js to establish the frontend page, and extract code with components to avoid redundant work.
At the final stage, deployment was a big challenge to us since plenty of factors came out and led to conflicts with our current design. For example, we used Session to store information in the client-server, but after deploying the backend on the AWS cloud server, we can not get the same Session id due to the Nginx load balancing, which assigns each distributed server with a different session id. Deployment took us a lot of time to handle, but we are glad to learn the knowledge during the process.

Recommendations for future

In the future, we will optimize the project by adding some attractive features like online payments, driving-route mapping and demonstration, etc.
Besides, Caching will be introduced into our project to store the value between client and server. We decided to use Redis with AWS, Ngnix to support this problem.
Machine learning is also an optimization way to enhance our functionality. Such as using machine learning to drive-route mapping to find a passenger a faster driver.
Apart from that, an administration subsystem can be added to provide better support to the users.
For us, with the experience of Jira project management, we will be able to assign tasks more reasonably.
Besides, deployment could be more efficient, as factors that may influence the code will be considered in advance.

Team contribution

Name	Contribution
Geping Chen	● Profile System: Frontend and Backend design ○ Users can see/edit their detail information ○ Users can register/edit their car information, upload images. ● SSO System ○ Frontend design: login/register page design ○ Backend design ■ Add Interceptor for better accessing DB to get users ■ Implement logout api. ● Deployment ○ In charge of all the services on cloud(AWS), including:AWS RDS as database, AWS S3 as image server. ○ Deployed our app on AWS ● General Problem: Log control, session management, threadlocal … ● Homepage
Guangrui Wang	● Order System: Frontend and Backend design ○ Users can see their orders in the order page. ○ Users can make an order after confirming a post. Both users in this ride should confirm the order before the ride. ● Database Design ○ In charge of all design and maintenance of the database. ● Modules and Structure Design ● Homepage
Tianbo Lu	● Post System: Frontend and Backend design ○ Users can search all the posts on the square or by filter based on their needs. ○ Users can post their needs on the square and can see the post details to place an order. ● Database Design ○ In charge of all design and maintenance of the database. ● Modules and Structure Design ● Main (explore) Page
Yechen Xu	● Communication System: Frontend and Back end design ○ Users can chat on the communication system for details on the ride. The message history is displayed on the page. ○ Users are provided with a list of all related chat rooms, named by two users of the chat room. ○ Design and maintain the database of the communication system. ● Modules and Structure Design ● Frontend Design ○ In charge of design of all pages' user Interface.
Jiayue Zhou	● SSO System: Frontend and Backend design ○ Users can register and login to our website with their VT emails. ○ Our website will send emails to verify users. ○ Add navigation-guards to filter out users who are not logged in.

	● Modules and Structure Design ● Information and About page ● Homepage