Abnormal User Operation

User Authentication

- Problem: If the user is not authenticated, he or she can visit pages we should think about how to keep users login states

  Solution: We use Django package which can be used as a decorator or a mixin, now if the user is not authenticated, he or she can only visit login and signup pages. After a user logs in the platform, he or she is able to use the account as the identity to visit all the pages. If he or she redirects to other pages, the login status should hold.

- Problem: The password created by the user can be any string. The real web application should require users to enter a strong enough password

  Solution: We add password strength validation when a new user registers. The password requirement includes:
  a. The password cannot be too similar to the user's personal information,
  b. The password must contain at least 8 characters
  c. The password cannot be a commonly used password such as qwe123456, qwert123, 123456abc, etc.
  d. The password cannot be entirely numeric
  e. The password must be the same as the password in the password confirmation field

Rider

- Problem: If the user click checks my orders, all the user's order will be displayed on the same page. The problem is we need to display the driver's order and the rider's order separately.

  Solution: we add two buttons for users to choose an identity. If a user chooses to be a rider then he or she can either launch a new order or search the existing ride orders and join one of them.

- Problem: Right now users cannot view the link to the invalid order on the list, but If the user tries to access the invalid resource use URL, the server will still redirect them to the page.

  Solution: We override get_queryset method and now if the user tries to access the invalid resource, the server will return Http 404 response,

Owner

- Problem: We implement the function to support users to create a new order and modify their own orders. If the driver confirms the order, the user can still modify the order.

  Solution: We override the get_querySet() method for our class-based view, now we can filter the confirmed orders. Also, in the previous version, we override the get_object function to return None instead of throw Http404 exception. We realized that we can catch the exception and in the catch block we redirect the user to a page providing error user-friendly information. Now, If any sharer joins a ride or the order is confirmed, the owner of the ride cannot edit the information. Otherwise, the owner should be able to change all his or her order.

- Problem: The order creator can choose unlimited number of passengers.

  Solution: The maximum number of passengers is set to 6, because most of the private vehicles can only carry 7 people including the driver. Also, our project to accommodate the situation if the requirement asks us to support vehicles with passenger capacity more than 6.

- Problem: If an order is completed and the owner of the order can still use URL to access the specific order.

  Solution: We override the get_queryset method to filter out the completed order. an order is completed and the owner of the order use URL to access the specific order, the server will return Http 404 response.

- Problem: If any sharer joined the order and the order owner uses the URL want to modify the specific order, he or she cannot modify it and receive Http 404 response.

  Solution: We override the get_queryset method to filter out the order whose sharerset is not empty. If any sharer joined the order and the order owner uses the URL want to modify the specific order, he or she cannot modify it and receive Http 404 response.

Sharer

- Problems: The sharer can change everything in the order and after the sharer joins the order, the order owner can still modify the order. We should avoid the sharer changing order information such as destination, start time, etc.

Solution: We do not allow sharer modifying the order and after any user joins the order, the owner cannot modify the order anymore.

- Problems: the share can view all the existing orders. If the user clicks one of them then the user can join the order. We should add a search feature and avoid user join orders using URL

  Solution: We ask sharer to fill in a search form before he or she can view the order list. The sharer is able to search the existing rides by specifying the destination, the number of passengers with him, expecting the ride start time and the special requirement. Here we do not search according to the start time instead of finish time. We think it is more reasonable to search according to the finish time when it is a long trip. For a short trip, for example, from Heights apartment to Perkins library, specifying start time is more likely to fit the real situation. But we can change it easily by using a range query on the finish time.

- Problem: After shares fill in the passenger number in the search form, they can still modify the number of passengers when they fill in the join information form. We should record the number of passengers.

  Solution: Sharer's passenger number is specified when he or she filled in the search form. After filling in the form, the passenger number can no longer be modified.

- Problem: If a sharer tried to join a ride again, Django would throw an error.

  Solution: We override the get_queryset method to filter out the order that the sharer has joined in. If a sharer tried to join a ride again, he or she will be redirected to a page notifying that they cannot join the ride again.

- We expect sharer join one ride per search, because sharer only specifies the number of passengers with him or her when filling the search form, so we think it is better for users to fill in the search form again. If the user uses the back button of the browser to join a ride again without filling the new search form, he or she would be redirected to a page notifying he or she cannot do it.

- To avoid a sharer join an invalid order such as a completed or confirmed order, an order owned by himself, we add the filter condition to get rid of these problems

Further Thought:

- If multiple users operate the order at the same time, whether or not our project can maintain consistent data depends on Django's and Postgre's implementation of database operation.

Driver

- If the driver

Service Failure

1. If the Nginx service fails, the request cannot be sent to our server and users cannot visit any pages.
2. If the database service fails, Django would throw exceptions according to the reason why the database is down. In this situation, since the server cannot run normally, users cannot communicate with the server.