

Wanderful

SWE SC2006

Group 5 Members:

Sethi Aryan

Jonathan Jiang Wenxuan

Jarel Tan Zhi Wen

Tan Wei Yuan

Dann, Wee Zi Jun

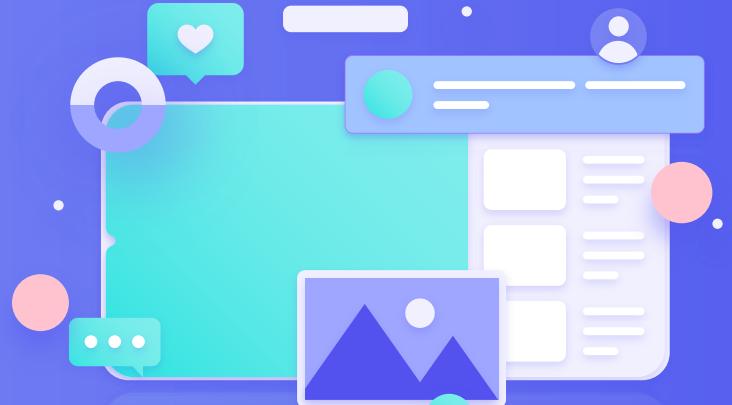




Table of contents

01 • • Introduction

04 • • Additional Features

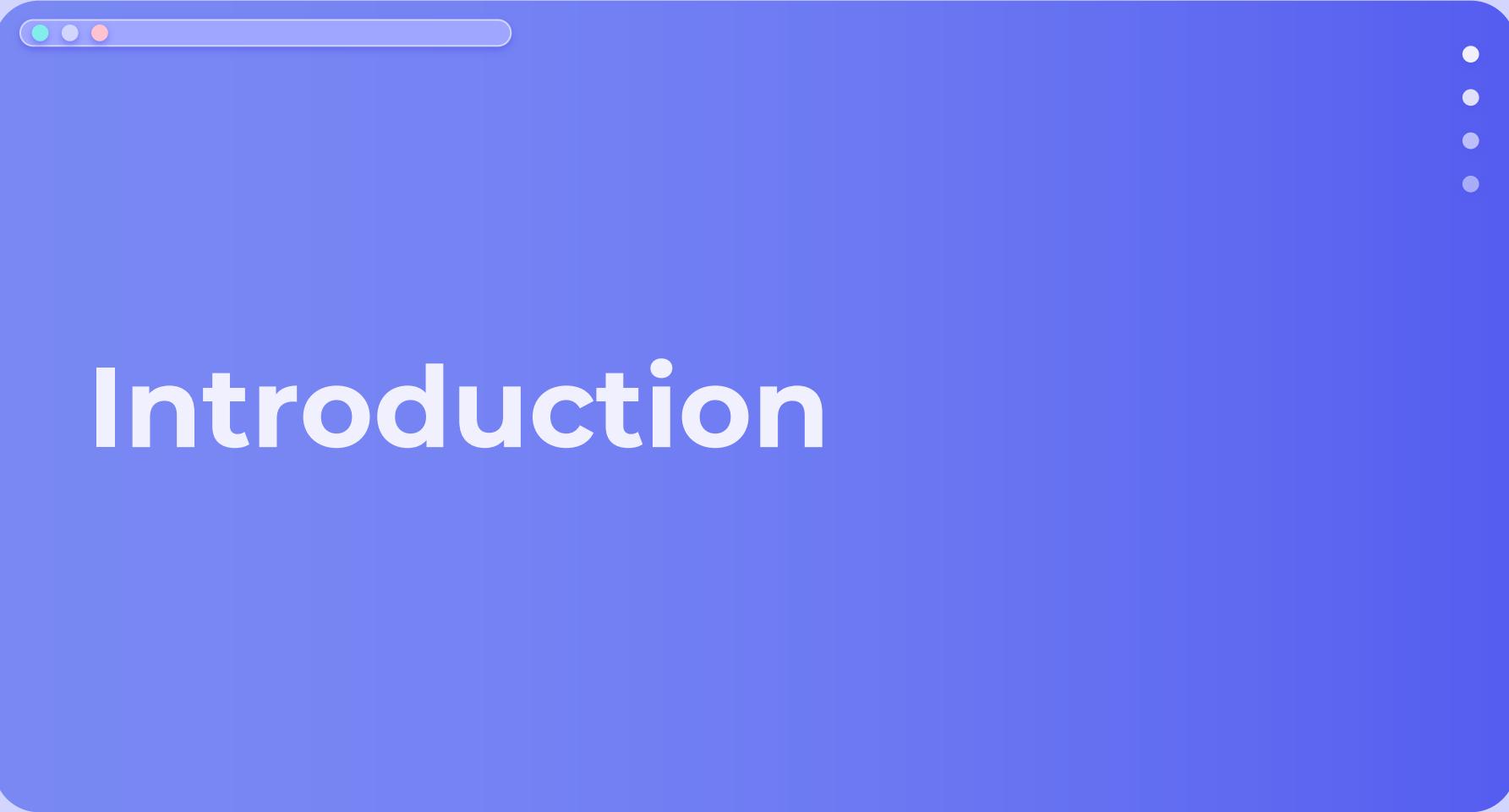
02 • • System Design

05 • • Demonstration

03 • • Use Cases

06 • • Good SE Practices

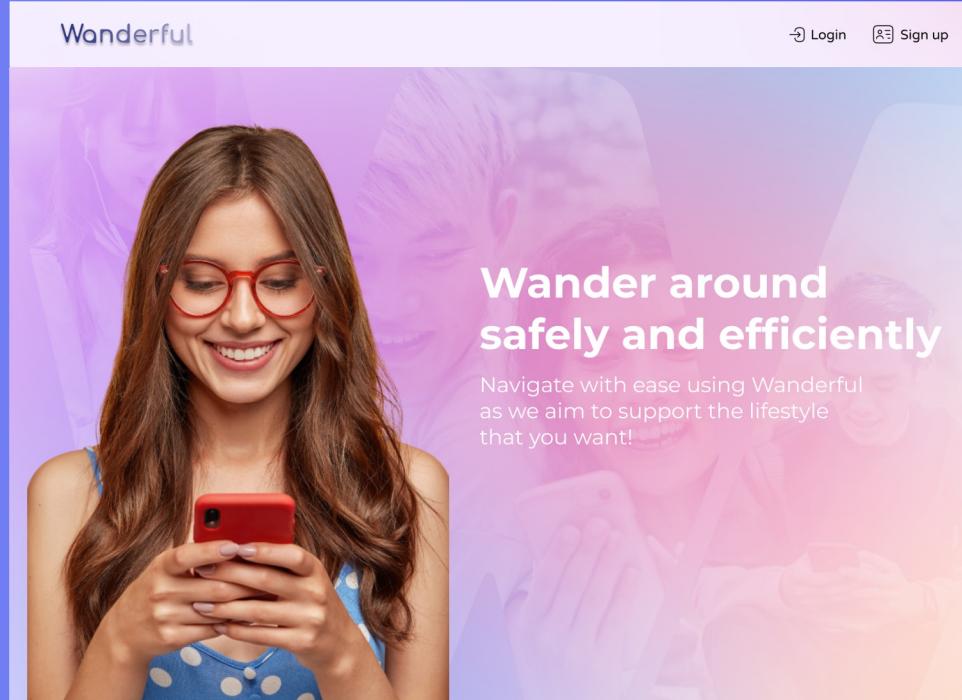




Introduction

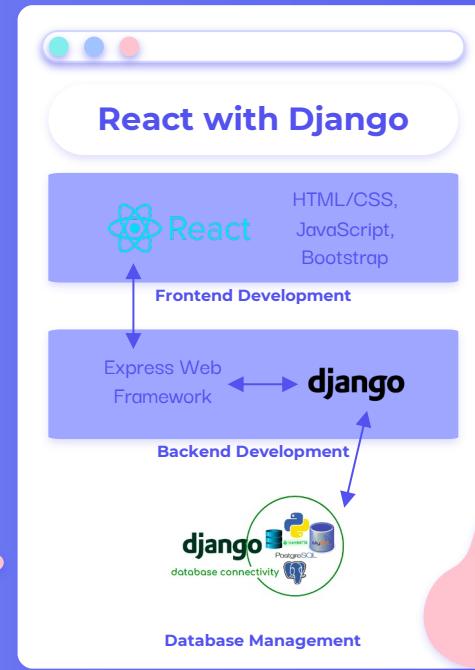
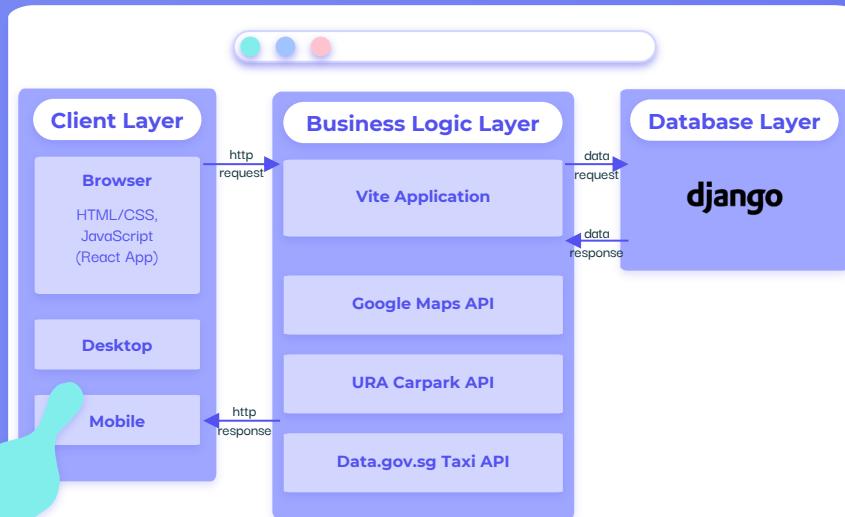
Wanderful

A One-Stop platform that enables all
commuters in Singapore to
conveniently carpool together



The image shows a screenshot of the Wanderful mobile application's landing page. At the top, there is a navigation bar with three colored dots (green, white, and orange) on the left, and 'Login' and 'Sign up' buttons on the right. The main header 'Wanderful' is centered above a large, semi-transparent background image of a woman smiling while looking at her red smartphone. To the right of the image, the tagline 'Wander around safely and efficiently' is displayed in bold white text. Below the tagline, a smaller text block reads: 'Navigate with ease using Wanderful as we aim to support the lifestyle that you want!'.

System Design



Frontend, Backend and Database



React.js

Frontend



Material UI

Frontend



Vite

Frontend



Django

Backend



SQLite

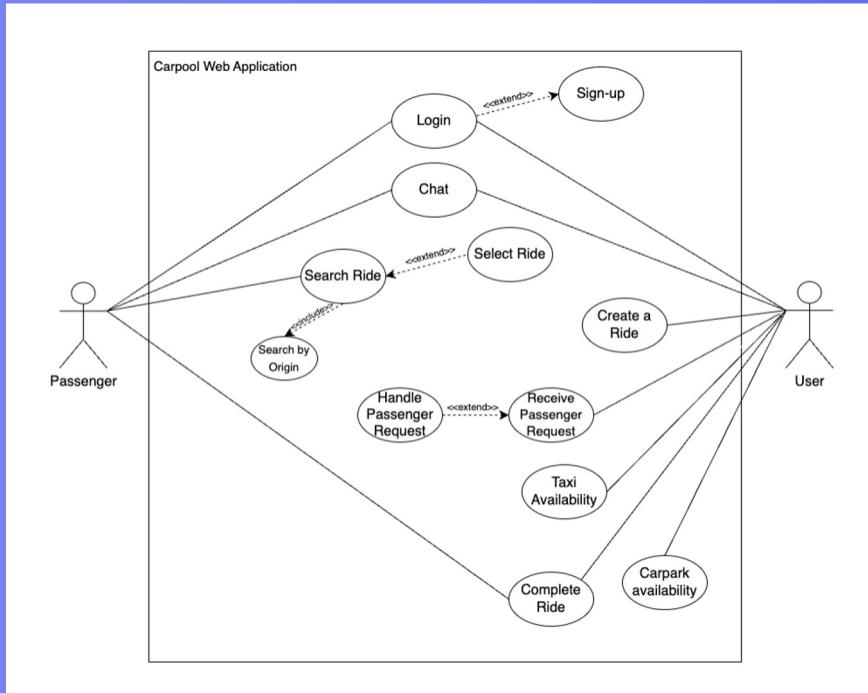
Database





Use Case Diagram

Use Case Diagram



Use Case #1 - Create Ride

Wanderful

Create Ride Search Ride My Rides Carparks

Welcome John

Drive Ride

Enter your drive details

Add a pick-up location Verify

Enter your destination Verify

10/4/2023 0:53

Recurring One-Time 0

Submit

127.0.0.1:5174/main

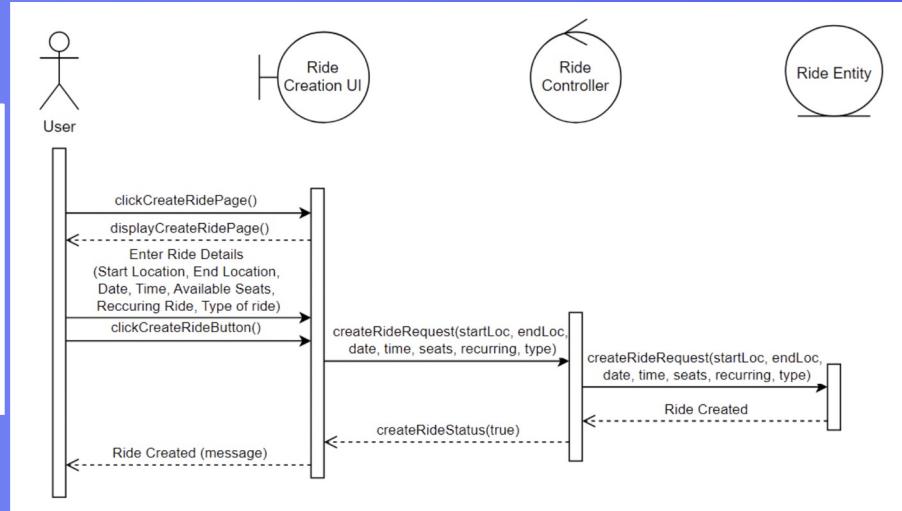
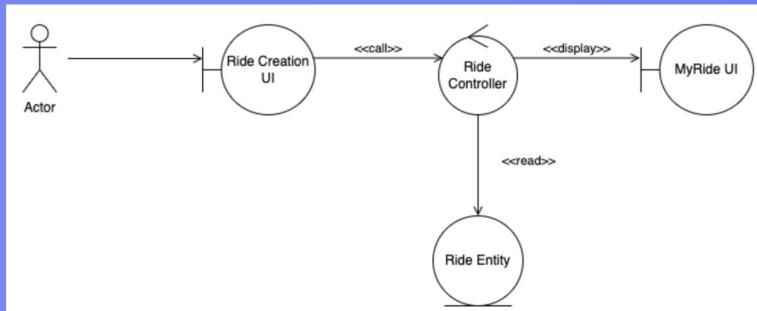
Keyboard shortcuts Map data ©2023 Google Terms of Use Report a map error

Use Case #1 - Create Ride

Use Case ID:	4.0		
Use Case Name:	Create Ride		
Created By:	Sethi Aryan	Last Updated By:	Sethi Aryan
Date Created:	07/02/2023	Date Last Updated:	21/02/2023

Actor:	Initiated by <i>User</i>
Description:	<i>User</i> creates a ride as a <i>Driver</i> or as a <i>Passenger</i> . To do so, <i>User</i> has to fill up the required details of the ride such as, start and end location, date & time of ride, number of available seats on the ride, indicates if the ride is a recurring ride or a one time ride, and indicates if the user has a personal car to use or if they want someone to share a taxi with.
Preconditions:	<ol style="list-style-type: none">1. <i>User</i> has an account2. <i>User</i> has successfully Logged into <u>account</u>
Postconditions:	1. The database containing all the rides has been updated with a new record
Priority:	High
Frequency of Use:	High
Flow of Events:	<ol style="list-style-type: none">1. <i>User</i> clicks on the "Create a Ride" Button2. <i>User</i> is prompted to enter ride details:<ol style="list-style-type: none">a. Starting Point: (Start location)b. Ending Point: (End location)c. Date & Time: (eg. 8 Feb 2023 0900hrs)d. Available Seats: (Number of Seats)e. Recurring ride: [Yes / No (One time ride)]f. Type of Ride: (Personal Car / Taxi)3. <i>User</i> clicks on the Create Ride Button4. The System will add the details of the ride into the database5. UI displays the message "Ride Created Successfully"
Alternative Flows:	-
Exceptions:	-
Includes:	-
Notes and Issues:	-

Use Case #1 - Create Ride



Use Case #1 - Create Ride (Entity)

```
class Rides(models.Model):
    creator = models.ForeignKey(User, related_name='creator', on_delete=models.CASCADE)
    origin = models.CharField(max_length=100)
    destination = models.CharField(max_length=100)
    CHOICES = (
        ('Taxi', 'Taxi'),
        ('Personal Car', 'Personal Car'),
    )
    types = models.CharField(max_length=15, choices=CHOICES, default='Personal Car')
    date_time = models.DateTimeField()
    recurring = models.BooleanField()
    seats = models.IntegerField()
    start_lat = models.CharField(max_length=50, null=True)
    end_lat = models.CharField(max_length=50, null=True)
    CHOICES = (
        ('Open', 'Open'),
        ('Closed', 'Closed'),
        ('Completed', 'Completed')
    )
    status = models.CharField(max_length=10, choices = CHOICES, default='Open')

    class Meta:
        verbose_name = ("Ride")
        get_latest_by = ["date_time"]

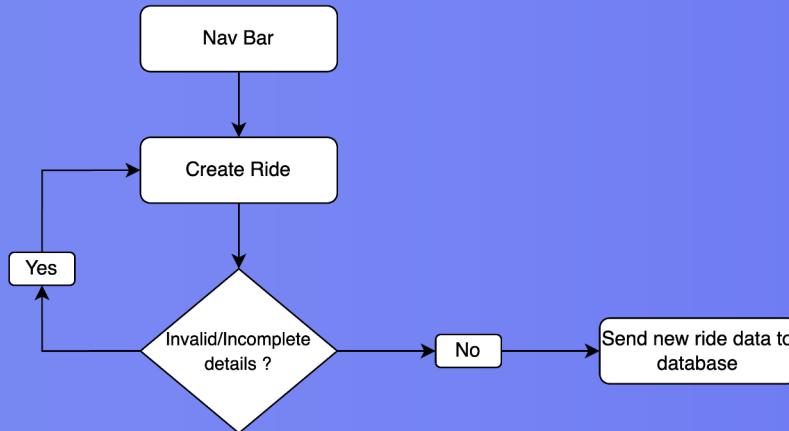
    def __str__(self):
        return f"{self.origin} to {self.destination}"
```

Use Case #1 - Create Ride (Controllers)

```
@api_view(['GET', 'POST'])
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def rides(request):
    if request.method == 'GET':
        rides = Rides.objects.exclude(creator=request.user.pk) #Returns only the open and closed rides that are not created by the current logged-in user
        rides = rides.exclude(status="Completed")
        time = timezone.make_aware(datetime.datetime.now(), timezone.get_default_timezone())
        for r in rides:
            if time>r.date_time and r.recurring==True:
                new_time = r.date_time + datetime.timedelta(days=7)
                Rides.objects.create(creator=r.creator, origin=r.origin, destination=r.destination, types=r.types, date_time=new_time, recurring=r.recurring, seats=r.seats)
                r.status = "Completed"
                r.save()
            elif time>r.date_time:
                r.status = "Completed"
                r.save()
        new_rides = rides.filter(status="Open")
        serializer = RidesSerializer(new_rides, many=True)
        return Response(serializer.data)

    if request.method == 'POST':
        data = JSONParser().parse(request)
        data["creator"] = request.user.pk #Do not need to pass creator in the api call's body, it is automatically added
        serializer = CreateRidesSerializer(data=data, many=False)
        if serializer.is_valid():
            serializer.save()
            return Response({"Success": "Ride Created Successfully"}, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Use Case #1 - Create Ride - Testing



TestID	Test Case	Expected Result	Actual Result
4.1	User creates rides with incomplete/invalid fields	User is asked to fill the fields again	User is asked to fill the fields again
4.2	User enters origin, destination, date-time, seats available and recurrence	New ride is created	New ride is created
4.3	User makes a personal car / taxi ride	Personal Car/ Taxi ride is created	Personal Car/ Taxi ride is created
4.4	User validates starting and ending location	User is shown the starting and ending point on the map with a route	User is shown the starting and ending point on the map with a route

White Box Testing

Black Box Testing

Use Case #2 - Chat

The screenshot shows the Wonderful app interface. At the top, there's a navigation bar with 'Create Ride', 'Search Ride', 'My Rides' (which is highlighted in black), 'Carparks', and 'Taxi'. On the right side of the nav bar is a user profile picture of a man and the text 'Welcome johncena'. Below the nav bar, there are three main sections: 'MyRides', 'Ride Requests', and 'FAQ'. The 'Ride Requests' section is active, displaying a card for a ride. The card includes a photo of a smiling man, the driver's name 'johncena', the pickup time '4/13/2023, 9:00:00 AM', the pickup location '80 Playfair Road', and the destination '330 Bukit Batok Street 33'. A 'Chat' button is located at the bottom right of this card. The main area below the card is a scrollable chat window. It contains messages from two users: 'jarel' and 'john cena'. The messages are as follows:

- jarel: Hi John
- jarel: Meet you at 9?
- john cena: Hi Jarel
- john cena: Yes sounds good!

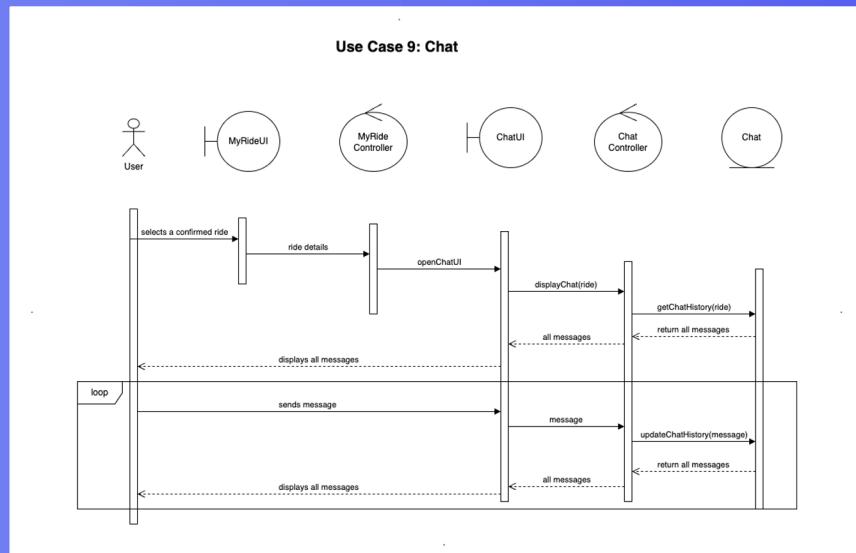
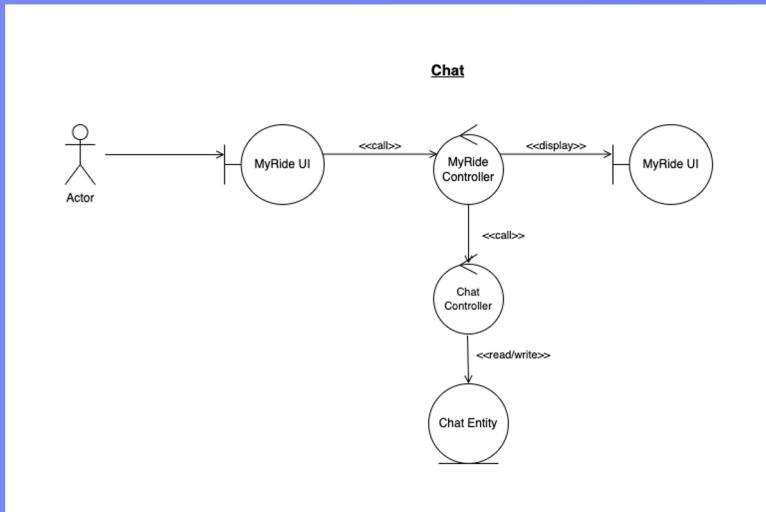
A small input field with a send icon is visible at the bottom of the chat window.

Use Case #2 - Chat

Use Case ID:	9.0		
Use Case Name:	Chat		
Created By:	Tan Wei Yuan	Last Updated By:	Tan Wei Yuan
Date Created:	07/02/2023	Date Last Updated:	07/02/2023

Actor:	<i>Driver and Passengers</i>
Description:	To allow the <i>driver</i> to communicate with the passengers
Preconditions:	<ol style="list-style-type: none">1. <i>Driver</i> must have accepted the ride request2. Ride confirmed between <i>driver</i> and <i>passenger</i>3. Conversation must be stored on a database
Postconditions:	-
Priority:	Medium
Frequency of Use:	High
Flow of Events:	<ol style="list-style-type: none">1. A ride is confirmed between <i>driver</i> and <i>passenger</i>2. <i>Driver</i> and <i>passenger</i> use the chat window to coordinate pick-up location and discuss any other details required.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Notes and Issues:	-

Use Case #2 - Chat



Use Case #2 - Chat (Entity Classes)

```
class Conversation(models.Model):
    rides = models.ForeignKey(Rides, related_name='conversations', on_delete=models.CASCADE)
    members = models.ManyToManyField(User, related_name='users')
    created_at = models.DateTimeField(auto_now_add=True)
    modified_at = models.DateTimeField(auto_now=True)

    class Meta:
        ordering = ('-modified_at',)
```

```
class ConversationMessage(models.Model):
    conversation = models.ForeignKey(Conversation, related_name='messages', on_delete=models.CASCADE)
    content = models.CharField(max_length=1000)
    created_by = models.ForeignKey(User, related_name='created_messages', on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

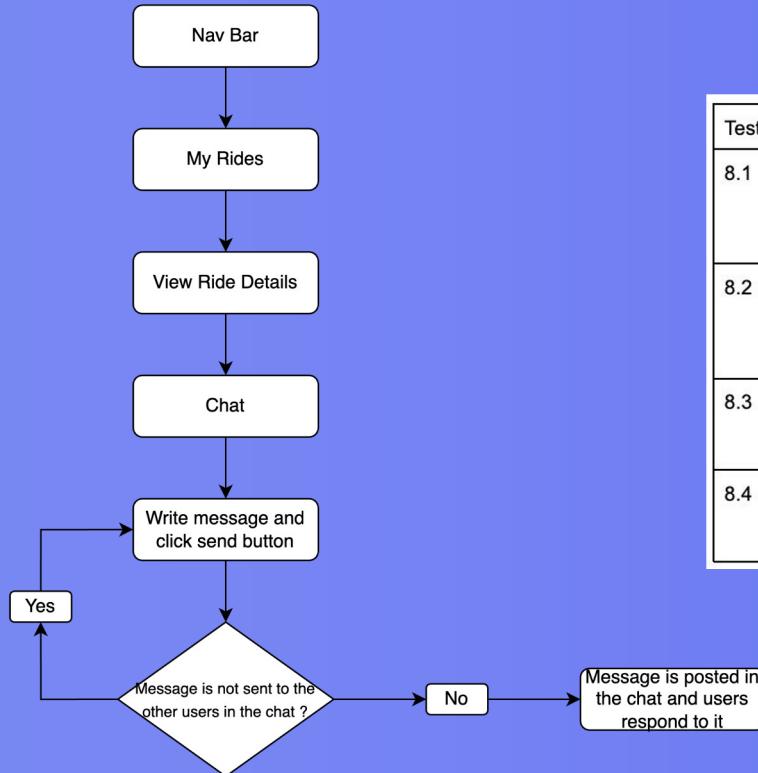
    class Meta:
        ordering = ('created_at',)

    def clean(self):
        if self.created_by not in self.conversation.members.all():
            raise ValidationError('The creator of the message must be a member of the conversation.'
```

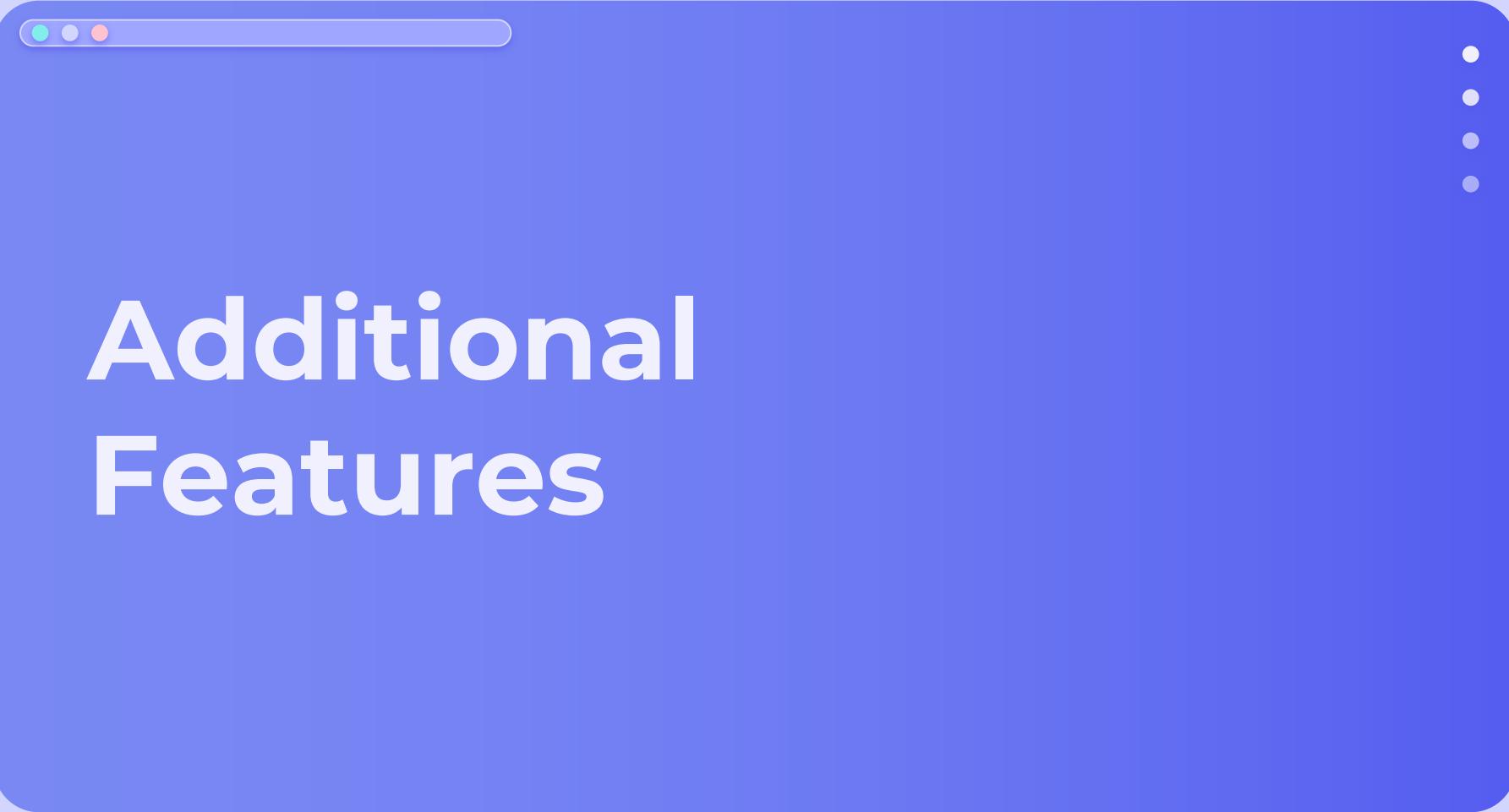
Use Case #2 - Chat (Controllers)

```
@api_view(['GET'])
@authentication_classes([TokenAuthentication,])
@permission_classes([IsAuthenticated,])
def chat(request,index):
    user=request.user
    try:
        conversationdata = Conversation.objects.get(rides=index , members=user)
        messagedata = ConversationMessage.objects.filter(conversation = conversationdata)
        print(messagedata)
        serializer = ConversationMessageSerializer(messagedata, many=True)
        return JsonResponse(serializer.data, safe=False)
    except ObjectDoesNotExist:
        return HttpResponse("ObjectDoesNotExist")
    return Response(f"Success : Message sent successfully ", status=status.HTTP_201_CREATED)
except ValidationError as e:
    return Response(f"Failed! {str(e)}", status=status.HTTP_400_BAD_REQUEST)
except Exception as e:
    return Response(f"Failed! {str(e)}", status=status.HTTP_400_BAD_REQUEST)
@api_view(['GET'])
@authentication_classes([TokenAuthentication,])
@permission_classes([IsAuthenticated,])
def members(request,index):
    user=request.user
    try:
        conversationdata = Conversation.objects.filter(rides=index)
        serializer = ConversationSerializer(conversationdata, many=True)
        return JsonResponse(serializer.data, safe=False)
    except ObjectDoesNotExist:
        return HttpResponse("ObjectDoesNotExist")
```

Use Case #2 - Chat - Testing



TestID	Test Case	Expected Result	Actual Result
8.1	Multiple users in one conversation	Conversation is created with multiple users according to passengers of ride	Conversation is created with multiple users according to passengers of ride
8.2	User starts chat with no previous history	No messages will be displayed	No messages will be displayed upon the creation of a conversation
8.3	User opens chat with existing messages	Previous sent messages will be displayed	Previous sent messages will be displayed
8.4	Users can be added to the conversation after some time	Users are added to the conversation	Users are added to the conversation



Additional Features

Carpark Availability

Car Park Availability is accessible via the Home Page for Drivers to pre-plan their drives in the event where they are looking for nearby car park lots to park at.

Future Expansion:

- Reservation System to reserve car park parking spaces for events or popular destinations
- Payment Integration using Payment Gateways API (Paypal, Stripe etc)
- Augmented Reality to see the real time live view of the car park (remotely observe their parked cars)

The screenshot shows the 'Carparks' tab of the Wanderful app. A search bar at the top displays 'Bukit Merah'. Below it, a list titled 'Carparks availability' shows four entries:

- T0017 Carpark lots available:19
- D0028 Carpark lots available:209
- O0028 Carpark lots available:0
- O0028 Carpark lots available:21

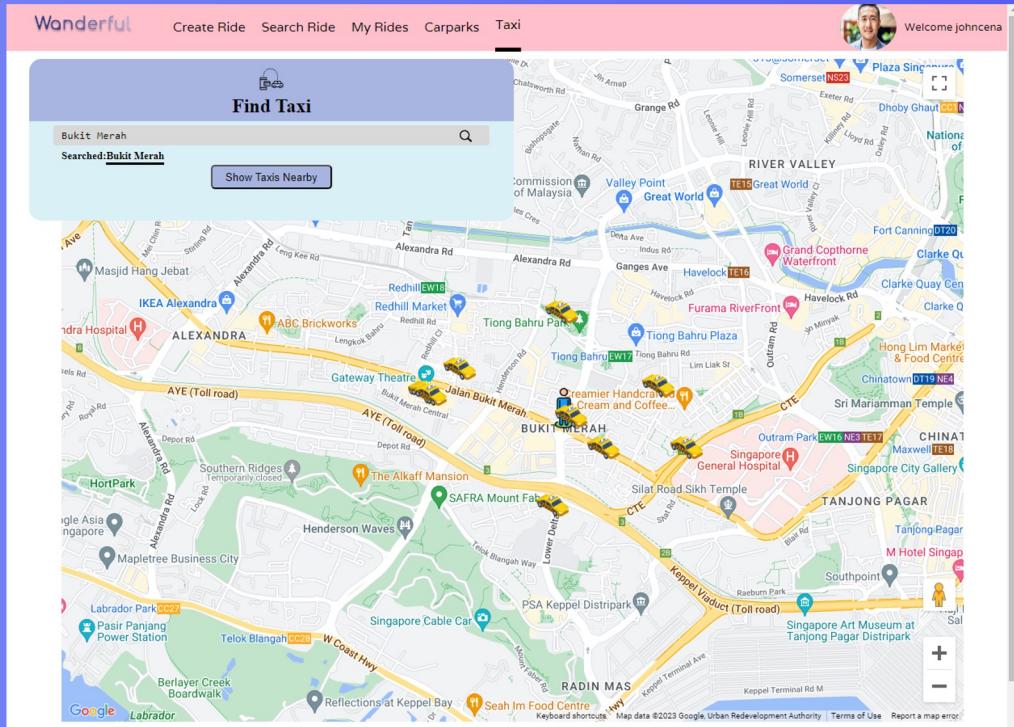
Below the list is a map of Singapore with several red 'P' icons indicating car park locations. The map includes labels for various neighborhoods like Bukit Merah, Chinatown, Little India, and Marina Bay. A legend in the bottom right corner of the map area shows icons for a person, a plus sign, and a minus sign.

Taxi Availability

Taxi Availability is accessible via the Home Page for Drivers for users to find alternative taxi rides to carpool in.

Future Expansion:

- Artificial Intelligence to provide more personalised recommendations and better route optimisation for users
- Voice-activated assistants to allow users to book a ride, check the status of their ride or get voice notifications.



Chat

Drivers and Passengers are able to chat with one another through our integrated chat.

Future Expansion:

- Artificial Intelligence Chatbot to provide personalised hat recommendations for better communication.
- Voice-activated Chat to allow drivers and users to communicate better. Drivers can activate this feature when driving to avoid phone usage.

The screenshot shows the Wonderful app's interface. At the top, there is a navigation bar with links: Create Ride, Search Ride, My Rides (which is underlined), Carparks, and Taxi. On the right side of the top bar is a user profile picture and the text "Welcome john cena".

On the left side of the main content area, there are three menu items: MyRides, Ride Requests, and FAQ. The main content area is titled "Ride Details" and displays the following information:

- Driver: john cena
- Pick-up location: 80 Playfair Road
- Destination: 330 Bukit Batok Street 33

Below this information is a "Chat" button. The main part of the screen is a chat interface between two users, "jarel" and "john cena". The messages are as follows:

- jarel: Hi John
- jarel: Meet you at 9?
- john cena: Hi Jarel
- john cena: Yes sounds good!

At the bottom of the chat interface is a text input field and a send icon.

Demonstratio n



Good Software Engineering (SE) Practices

Security

- The highlighted portion creates a token every time a new user is created
- Used with every REST API on our backend to make sure that only the authenticated users are allowed to access it

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.contrib.auth.models import User
from rest_framework.authtoken.models import Token
from .models import UserProfiles

@receiver(post_save, sender=User, weak=False)
def report_uploaded(sender, instance, created, **kwargs):
    if created:
        Token.objects.create(user=instance)
        UserProfiles.objects.create(user=instance)
```

```
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
```

Encapsulation, Abstraction.

- We follow the **Entity Boundary**
Class structure by creating a class that acts as an interface between the software system's internal entities and the external world
- This structure promotes clear separation (Encapsulation) and exposes simpler and more intuitive interface (Abstraction)

```
/*
 * displays a form requiring the user to input the necessary details based on props.type
 * @param {type} props
 * @return {node} details input form
 */
const CreateRide = (props) =>{
  const [marker1, setMarker1] = useContext(Marker1Context)
  const [marker2, setMarker2] = useContext(Marker2Context)

  const [verify1, setVerify1] = useState(false)
  const [verify2, setVerify2] = useState(false)
  const [value, onChange] = useState(new Date())
  const [seats, setSeats] = useState({
    start : '',
    end : '',
    time : new Date(),
    seats : 1,
    recurring : false,
    type : '',
    startingAt : {},
    endingAt : {}
  })
  useEffect(() => {
    setDetails({...details, type : props.type});
  }, [props.type]);

  const text = ["Enter your drive details","Enter your ride details"]

  const handleSubmit = async (event)=>
  event.preventDefault();
  if (!verify1 && !verify2){
    setDetails({...details, time : value})
    console.log(details.time.toDateString())
    console.log(typeof details.seats)
    props.parentCallBack(details,1)
  }
  else if (!verify1){
    alert("Please Verify Start location")
  }
  else if (!verify2){
    alert("Please Verify End location")
  }
}

return (
  <form className="ride-creation-body" onSubmit={handleSubmit}>
    <div className="ride-creation-body-header">{props.type === "Personal Car"? text[0]: text[1]}</div>
    <div className="input-field">
      <input type="text" id="start" className="input-field" placeholder="Add a pick-up location" value = {details.start} onChange={e=>{setDetails({...details, start : e.target.value}), setVerify1(!value)}} required>
      <input type="button" value="Verify" onClick={()=>handleVerify(e,"start",props)}>Verify</button> 
    </div>
    <div className="input-field">
      <input type="text" id="end" className="input-field" placeholder="Enter your destination" value = {details.end} onChange={e=>{setDetails({...details, end : e.target.value}), setVerify2(!value)}} required>
      <input type="button" value="Verify" onClick={()=>handleVerify(e,"end",props)}>Verify</button> 
    </div>
    <div className="button-area">
      <button type="button" className="button" onClick={e=> handleClick(e)}>Recurring</button>
      <button type="button" className="button" onClick={e=> handleClick(e)}>One-Time</button>
    </div>
    <div>
      <input type="number" className="ride-creation-node" placeholder="seats" value = {details.seats} onChange={e=>{setDetails({...details, seats : e.target.value})}} required>
    </div>
    <div className="ride-creation-node">
      <input type="text" className="ride-creation-node" placeholder="passenger" value = {details.passenger} onChange={e=>{setDetails({...details, passenger : e.target.value})}} required>
    </div>
    <button className="ride-creation-submit" type="submit">
      Submit
    </button>
  </form>
)
```

Low Coupling

- Each page has a single, specific and well-defined responsibility, making it easier to modify or replace individual parts without affecting other parts of the web application

```
└─ styles
    # Carpark.css
    # DefaultStyles.css
    # EntranceScreen.css
    # Form.css
    # LoginScreen.css
    # main.css
    # MyRides.css
    # Nav.css
    # RideCreation.css
    # SearchRide.css
    # Settings.css
    # SignupScreen.css
    ⚡ App.jsx
    ⚡ main.jsx
    ⚡ .gitignore
    ⚡ index.html
    { } package-lock.json
    { } package.json
    📄 README.md
    JS vite.config.js
```

```
└─ src
    > assets
    └─ components
        ⚡ Carpark.jsx
        ⚡ Dashboard.jsx
        ⚡ EntranceScreen.jsx
        ⚡ Helper.jsx
        ⚡ LoginScreen.jsx
        ⚡ Main.jsx
        ⚡ Map.jsx
        ⚡ MyRides.jsx
        ⚡ Navblank.jsx
        ⚡ NavGuest.jsx
        ⚡ NavMain.jsx
        ⚡ NavUser.jsx
        ⚡ RideCreation.jsx
        ⚡ Rides.jsx
        ⚡ SearchRide.jsx
        ⚡ Settings.jsx
        ⚡ SignupScreen.jsx
        ⚡ Usercontext.jsx
```

Maintainability & Readability

- Organised our Software Engineering Project with clear and consistent directory structure
- Appropriate .jsx names
- Document and Commented the code using JSDoc

```
✓ SOFTWARE-ENG
> node_modules
> public
✓ src
> assets
✓ components
❶ Carpark.jsx
❶ Dashboard.jsx
❶ EntranceScreen.jsx
❶ Helper.jsx
❶ LoginScreen.jsx
❶ Main.jsx
❶ Map.jsx
❶ MyRides.jsx
❶ Navblank.jsx
❶ NavGuest.jsx
❶ NavMain.jsx
❶ NavUser.jsx
❶ RideCreation.jsx
❶ Rides.jsx
❶ SearchRide.jsx
❶ Settings.jsx
❶ SignupScreen.jsx
❶ Usercontext.jsx
> styles
❶ App.jsx
❶ main.jsx
❷ .gitignore
❸ index.html
{} package-lock.json
{} package.json
❶ README.md
JS vite.config.js
```

```
1 import React, { useEffect, useState, useContext } from "react";
2 import './styles/RideCreation.css'
3 import DateTimePicker from "react-datetime-picker";
4 import { getGeoCode } from "./Helper";
5 import { MarkerContext } from "./Usercontext";
6 import { Marker } from "google-maps-react";
7 import { Marker2Context } from "./Usercontext";
8
9
10 /**
11  * Returns the CreateRideUI that allows Users to select "Drive" or "Taxi"
12  * @param {parentCallBack} props complete
13  * @returns the CreateRideUI
14 */
15
16
17 export default function RideCreation(props) {
18
19   const handleSubmit = (event) =>{
20     event.preventDefault();
21   }
22 }
```

Resource Optimisation

- UseEffect hooks to make function calls only when a certain state has been changed
- This cuts down on the total number of API calls

```
useEffect(()=>{
  console.log("In useEffect")
  let arr = []
  allCarparks.map((a) => {
    const geom = a.geometries[0].coordinates.toString()
    const geomArr = geom.split(",")
    const Xdiff = Math.abs(+geomArr[0] - searchCoords[0])
    const Ydiff = Math.abs(+geomArr[1] - searchCoords[1])
    const totalDiff = Xdiff + Ydiff
    if(Xdiff<5000 && Ydiff<5000){
      // console.log(a.totalDiff)
      arr.push(a, totalDiff)
    }
  })
  console.log("carpark arr",arr)
  console.log(filter)
  filter === "Distance" ?
  arr.sort((function(index){
    return function(a, b){
      return (a[index] === b[index] ? 0 : (a[index] < b[index] ? -1 : 1));
    }
  })(1)):
  arr.sort((function(index){
    return function(a, b){
      return (+a[index].lotsAvailable === +b[index].lotsAvailable ? 0 : (+a[index].lotsAvailable > +b[index].lotsAvailable ? -1 : 1));
    }
  })(0));
  arr.length > 57 setFilteredCarparks(arr.slice(0,57)) : setFilteredCarparks(arr)
  console.log("Set CarparkMarker", filteredCarparks)
  setCarparkMarker([ ])
}
, [searchCoords, filter])
```



Thank
you!

