Wanderful

SWE SC2006

Group 5 Members:

Sethi Aryan

Jonathan Jiang Wenxuan

Jarel Tan Zhi Wen

Tan Wei Yuan

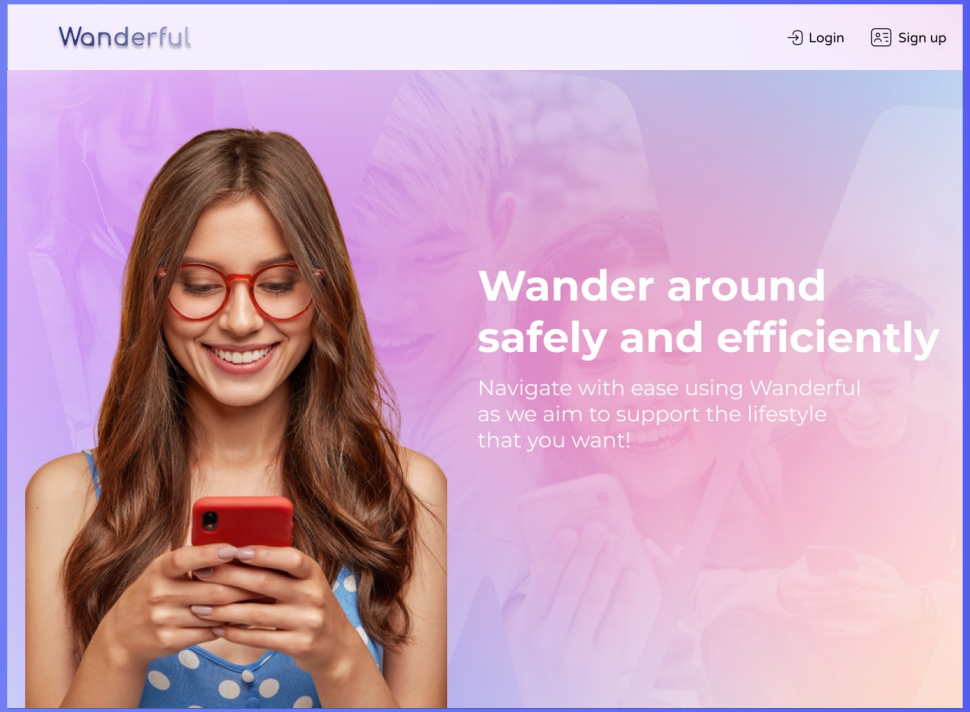
Dann, Wee Zi Jun

Table of contents

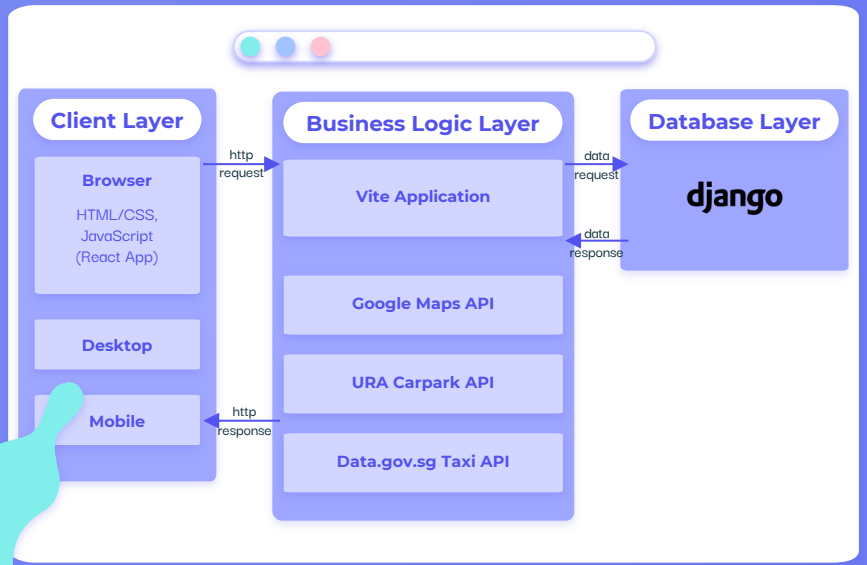
Introduction

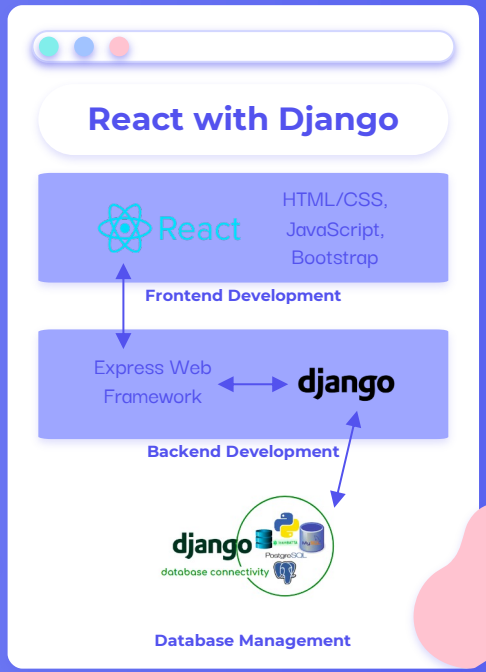
Wanderful

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



System Design





Frontend, Backend and Database

React.js

Frontend

Material UI

Frontend

Vite

Frontend

Django

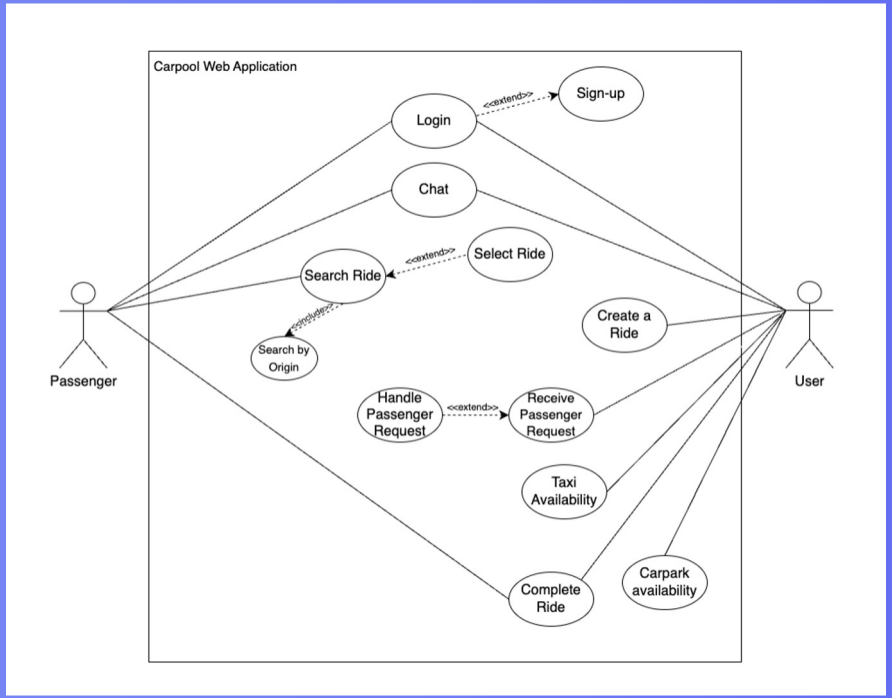
Backend

SQLite

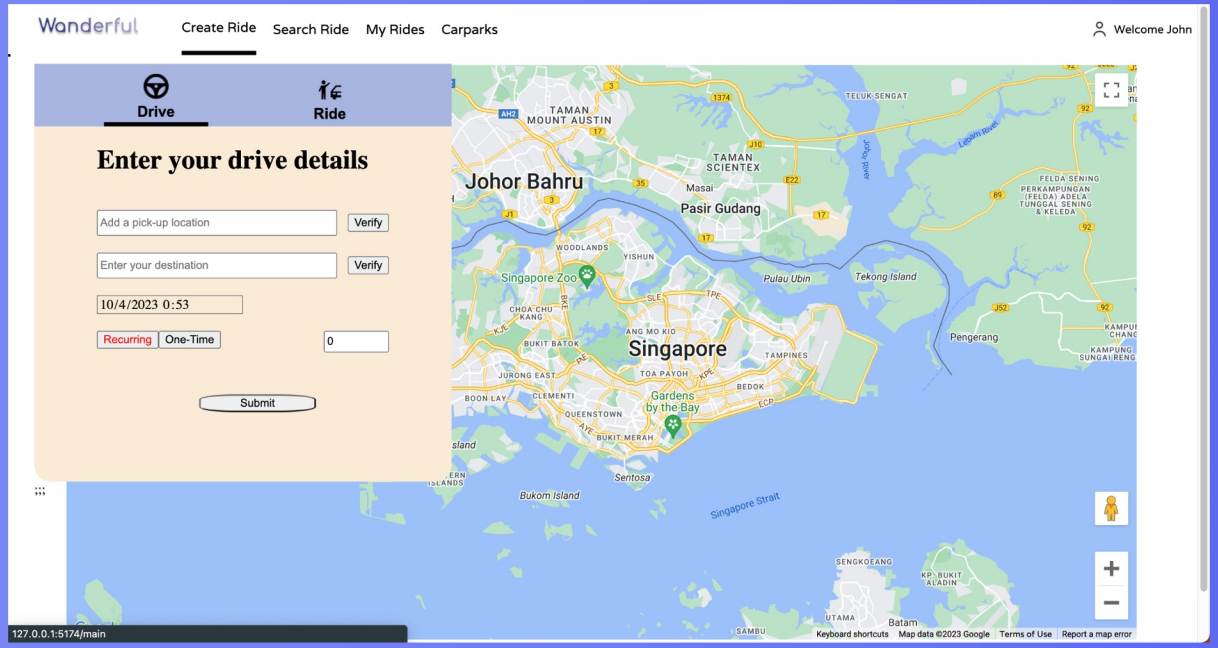
Database

Use Case Diagram

Use Case Diagram



Use Case #1 - Create Ride

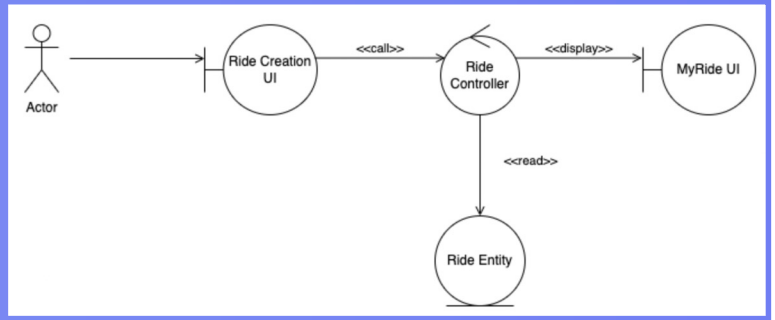


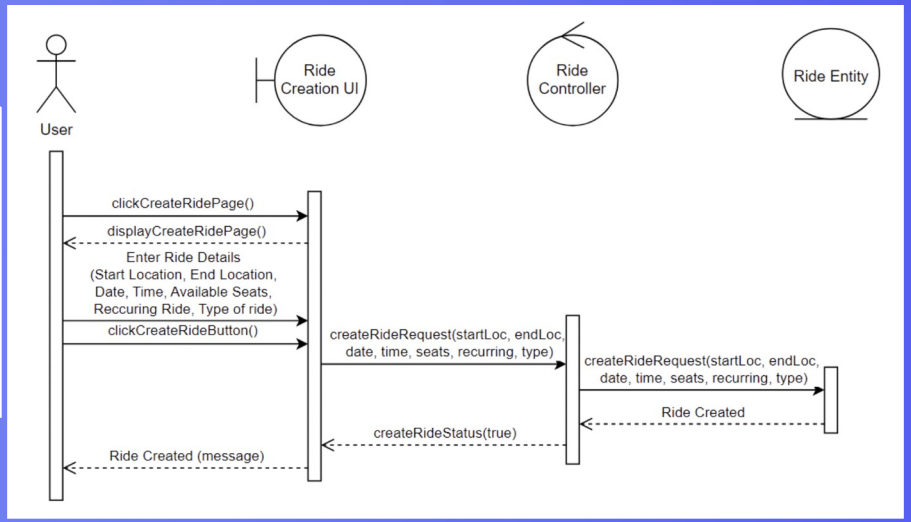
Use Case #1 - Create Ride

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 *User*
Description:	*User* creates a ride as a *Driver* or as a *Passenger*. To do so, User 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:	1. User has an account 2. User has successfully Logged into account
Postconditions:	1. The database containing all the rides has been updated with a new record
Priority:	High
Frequency of Use:	High
Flow of Events:	1. *User* clicks on the “Create a Ride” Button 2. *User* is prompted to enter ride details: 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. *User* clicks on the Create Ride Button 4. The System will add the details of the ride into the database 5. UI displays the message “Ride Created Successfully”
Alternative Flows:	-
Exceptions:	-
Includes:	-
Notes and Issues:	-

Use Case #1 - Create Ride





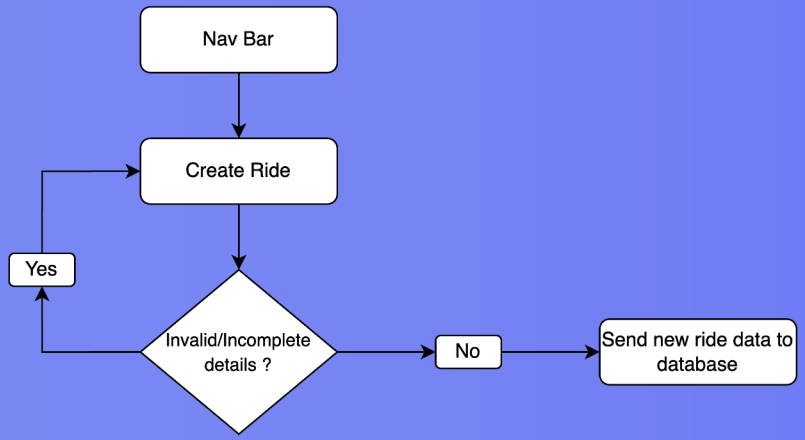
Use Case #1 - Create Ride (Entity)

```python
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)

```python
@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.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

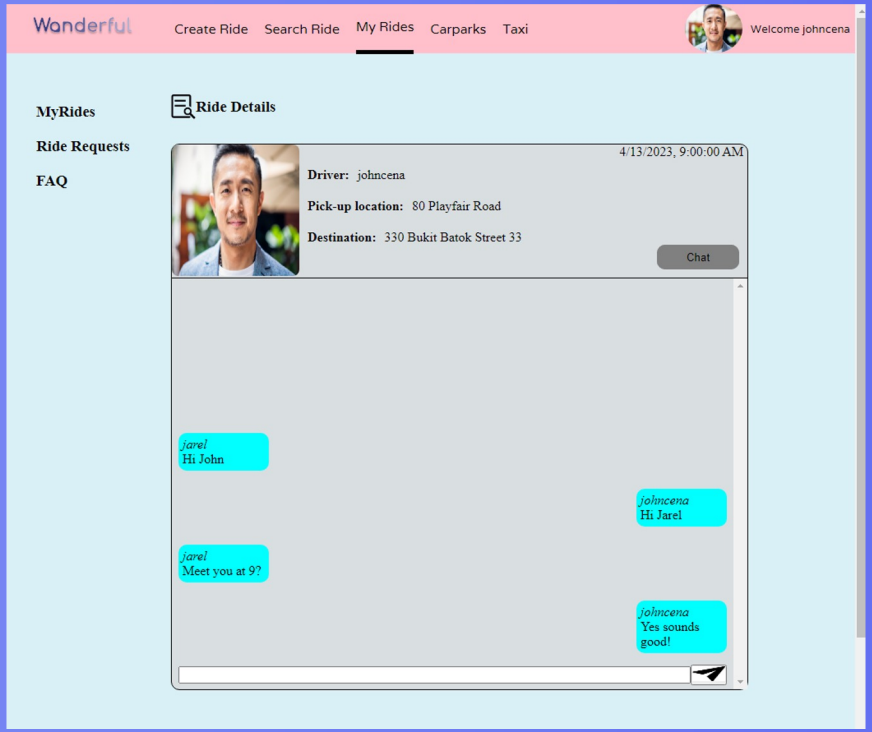


TestID	Test Case	Expected Result	Actual Result
4.1	User creates rides with incomplete/invalid fields	User is asked to fill the fields again	User is asked to fill the fields again
4.2	User enters origin, destination, date-time, seats available and recurrence	New ride is created	New ride is created
4.3	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

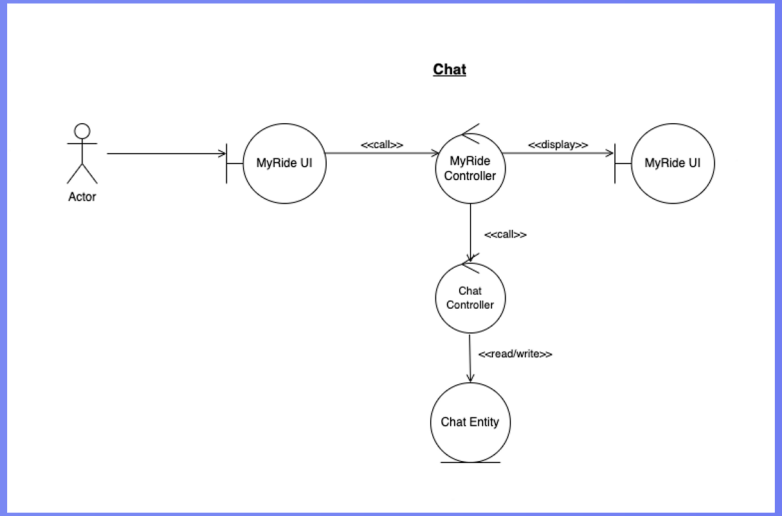


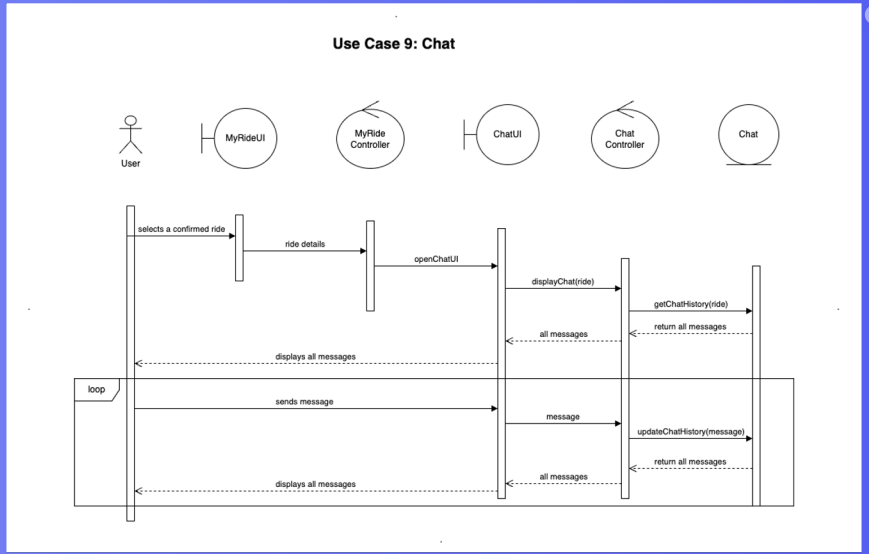
Use Case #2 - Chat

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:	***Driver*** and ***Passengers***
Description:	To allow the *driver* to communicate with the passengers
Preconditions:	1. *Driver* must have accepted the ride request 2. Ride confirmed between *driver* and *passenger* 3. Conversation must be stored on a database
Postconditions:	-
Priority:	Medium
Frequency of Use:	High
Flow of Events:	1. A ride is confirmed between *driver* and *passenger* 2. *Driver* and *passenger* 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)

```python
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',)
```

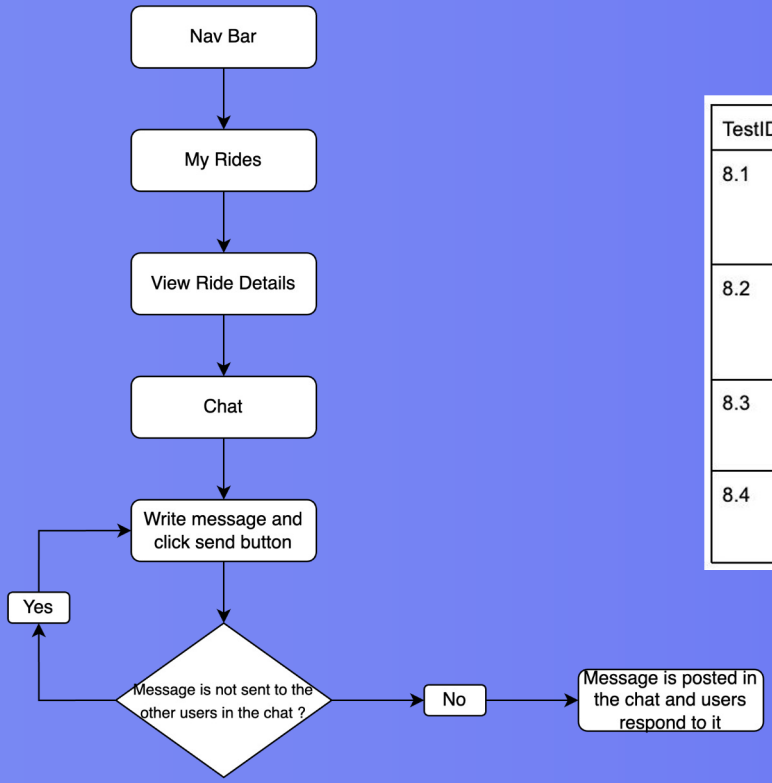
```python
class ConversationMessage(models.Model):
    conversation = models.ForeignKey(Conversation, related_name='messages', on_delete=models.CASCADE)
    content = models.CharField(max_length=1000)
    created_by = models.ForeignKey(User, related_name='created_messages', on_delete=models.CASCADE)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

```python
@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")
```

```python
@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

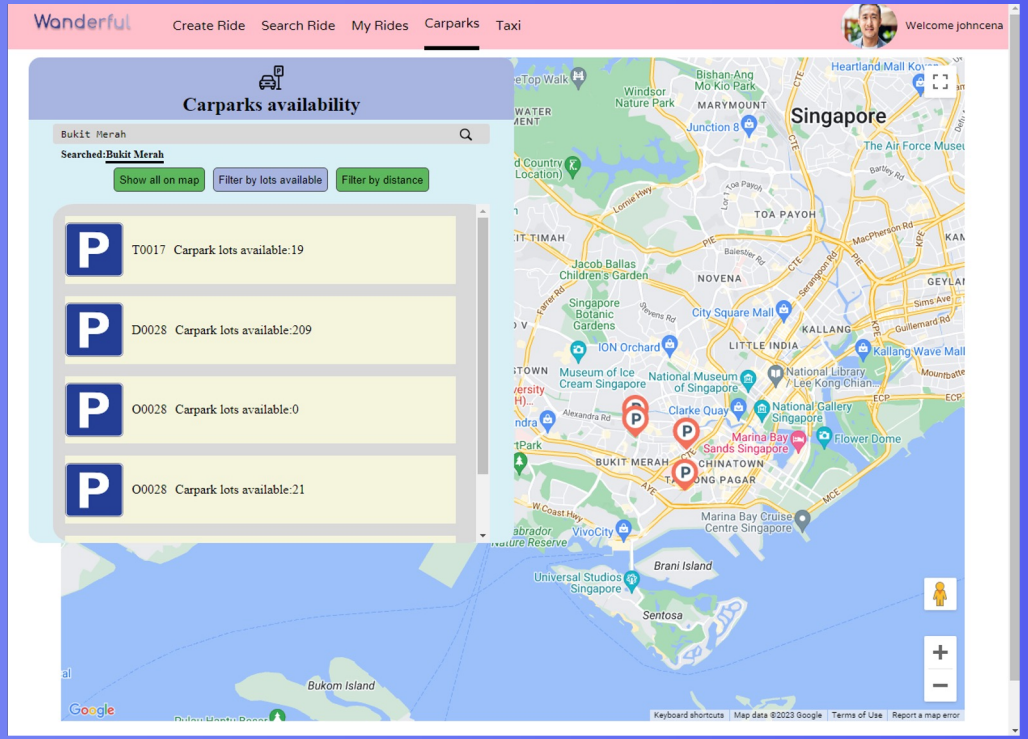
Additional Features

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)

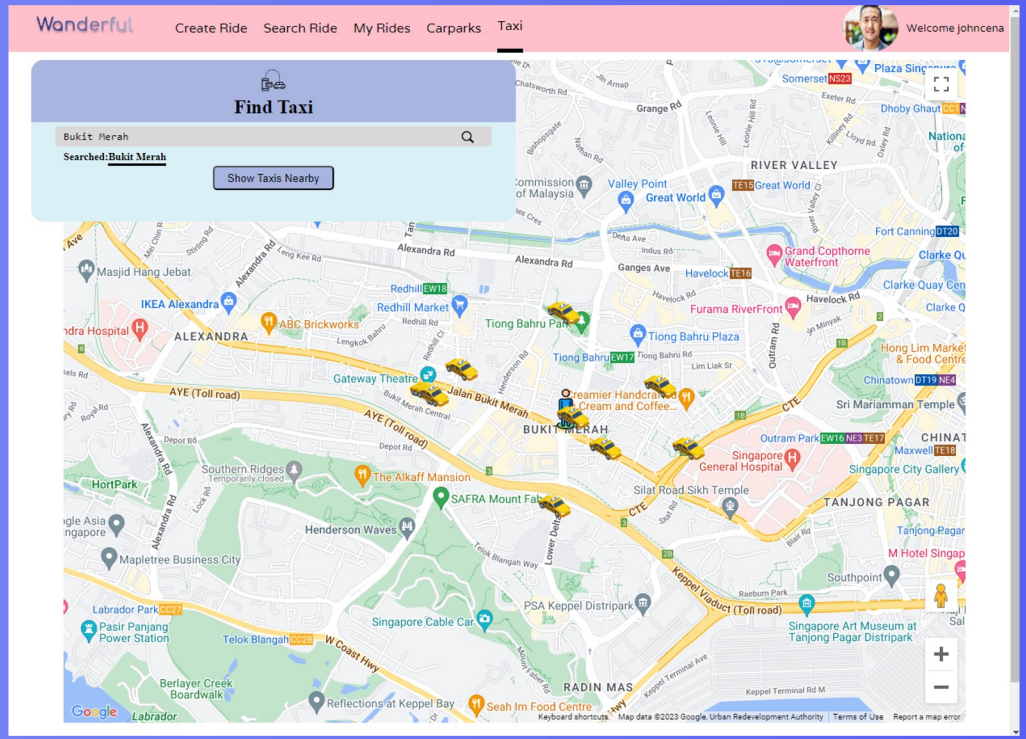


Taxi Availability

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.

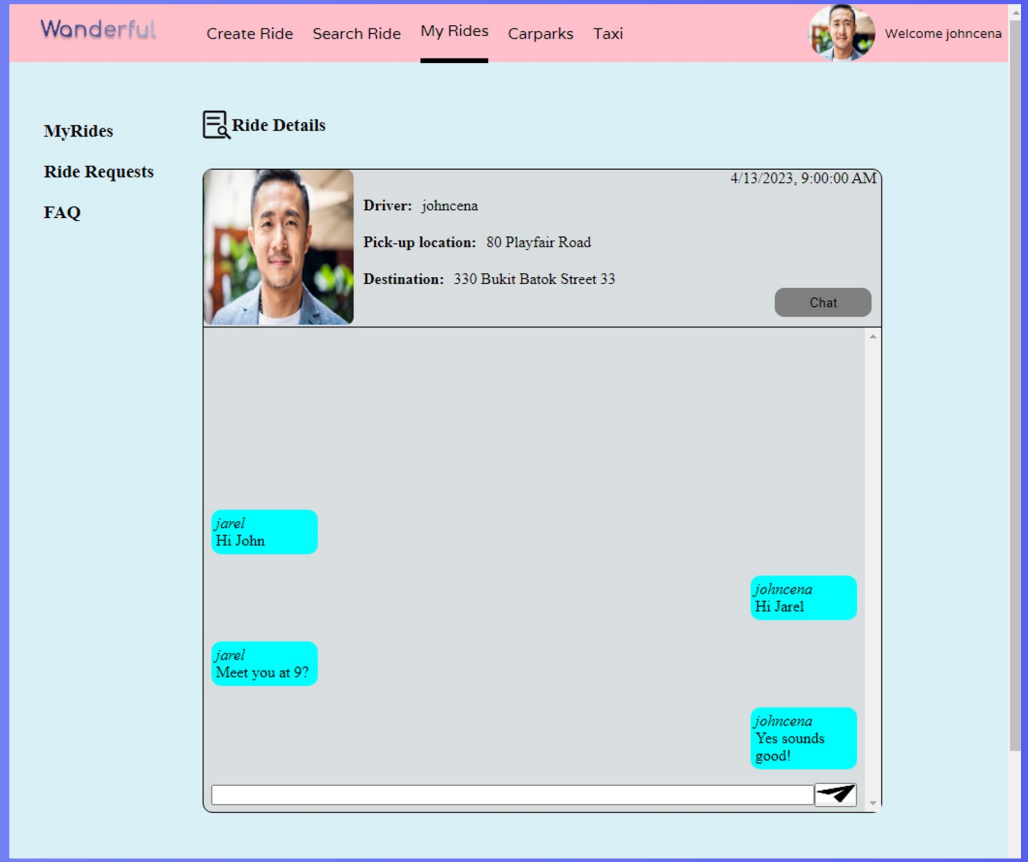


Chat

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.



Demonstration

Good Software Engineering (SE) Practices

Security

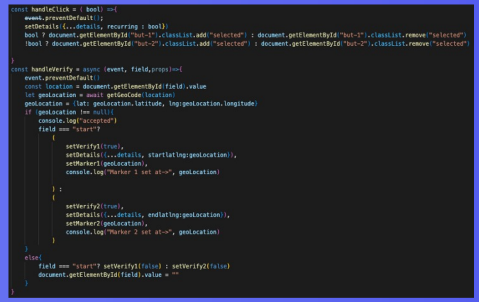
- The highlighted portion creates a token every time a new user is created
- Used with every REST API on our backend to make sure that only the authenticated users are allowed to access it

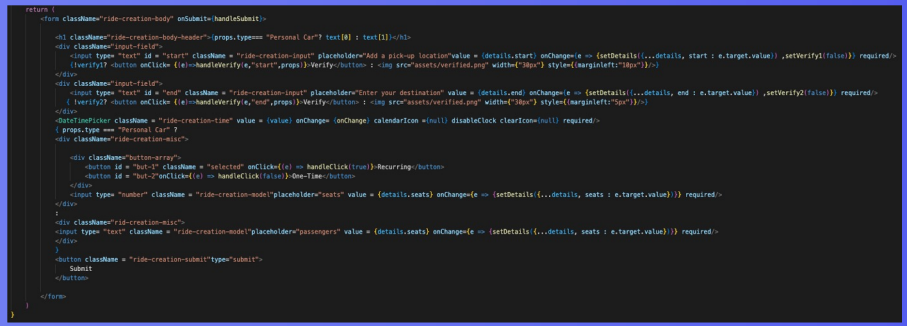
```python
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.contrib.auth.models import User
from rest_framework.authtoken.models import Token
from .models import UserProfiles

@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)
```

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

Encapsulation, Abstraction.

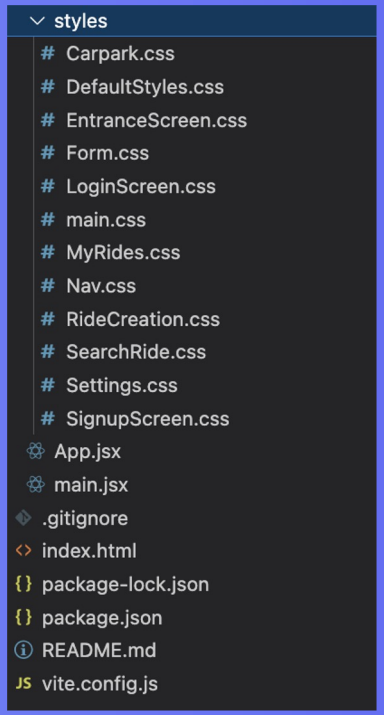
- We follow the **Entity Boundary Class** structure by creating a class that acts as an interface between the software system's internal entities and the external world
- This structure promotes clear separation (Encapsulation) and exposes simpler and more intuitive interface (Abstraction)

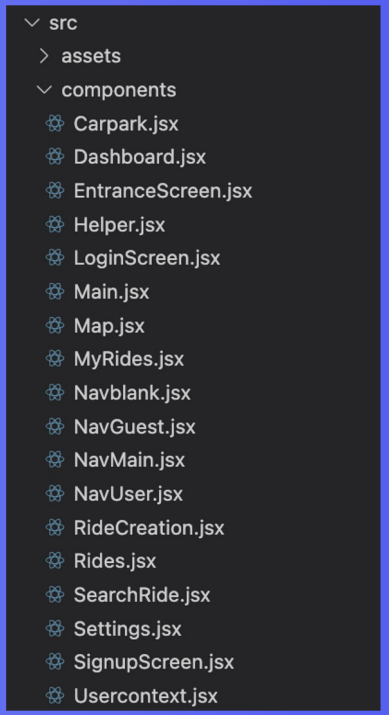




Low Coupling

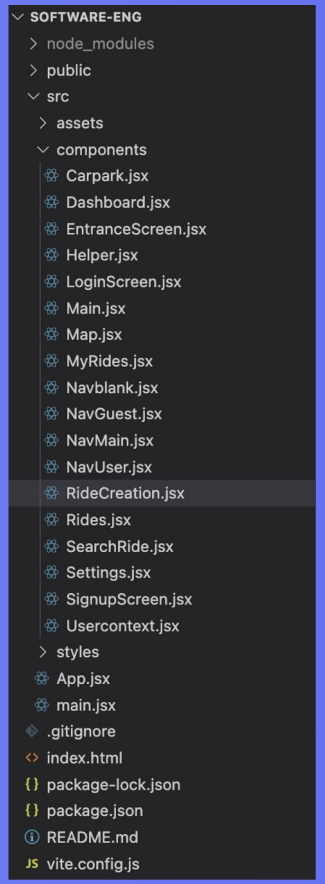
- Each page has a single, specific and well-defined responsibility, making it easier to modify or replace individual parts without affecting other parts of the web application





Maintainability & Readability

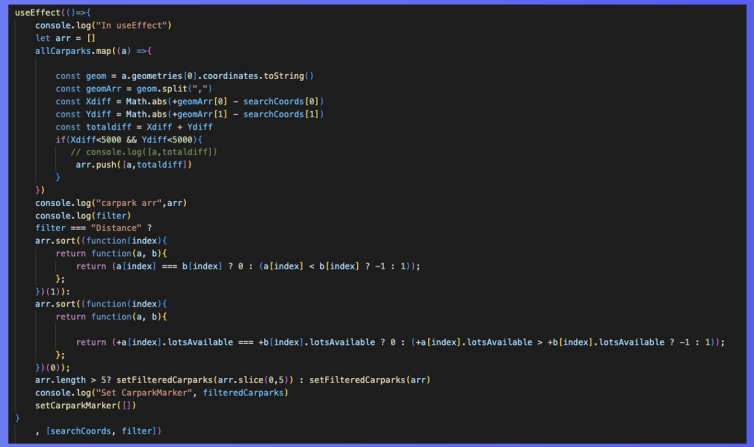
- Organised our Software Engineering Project with clear and consistent directory structure
- Appropriate .jsx names
- Document and Commented the code using JSDoc




```jsx
import React, { useEffect, useState, useContext } from "react";
import "../styles/RideCreation.css"
import DateTimePicker from "react-datetime-picker";
import { getGeoCode } from "./Helper";
import { Marker1Context } from "./Usercontext";
import { Marker } from "google-maps-react";
import { Marker2Context } from "./Usercontext";


/**
 * Rerturns the CreateRideUI that allows Users to select "Drive" or "Taxi"
 * @param {parentCallBack} props complete
 * @returns the CreateRideUI
 */


export default function RideCreation(props) {

    const handleSubmit = (event) =>{
        event.preventDefault();

    }
```

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 > 5? setFilteredCarparks(arr.slice(0,5)) : setFilteredCarparks(arr)
    console.log("Set CarparkMarker", filteredCarparks)
    setCarparkMarker([])
}
    , [searchCoords, filter])
```

Thank you!