

PoolCarz App - Node and Express

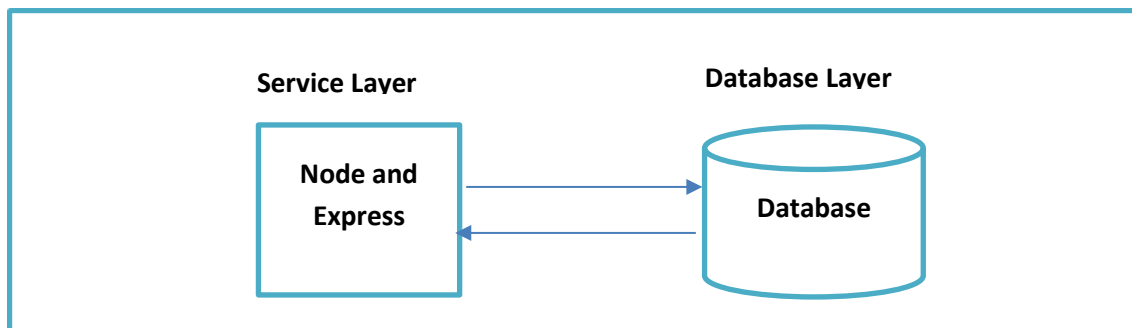
Problem Statement

PoolCarz is a web application for car-pooling. The application allows users to share ride with others. User can either book a ride or offer a ride. The application should contain the following features:

Use Case	Description
Login	Logs into the application to view ride details
Book a Ride	Allows rider to book the ride
Ride Details	It renders complete list of the rides available
Offer Ride	Allows user to register his details to offer ride to others
Logout	Logs out from the application and navigates to the login page

Application Architecture

PoolCarz application should be developed as a back end application which should have service layer and database layer.



- Node and Express are used at service layer to implement service functionality
- MongoDB is used at database layer to persist the data

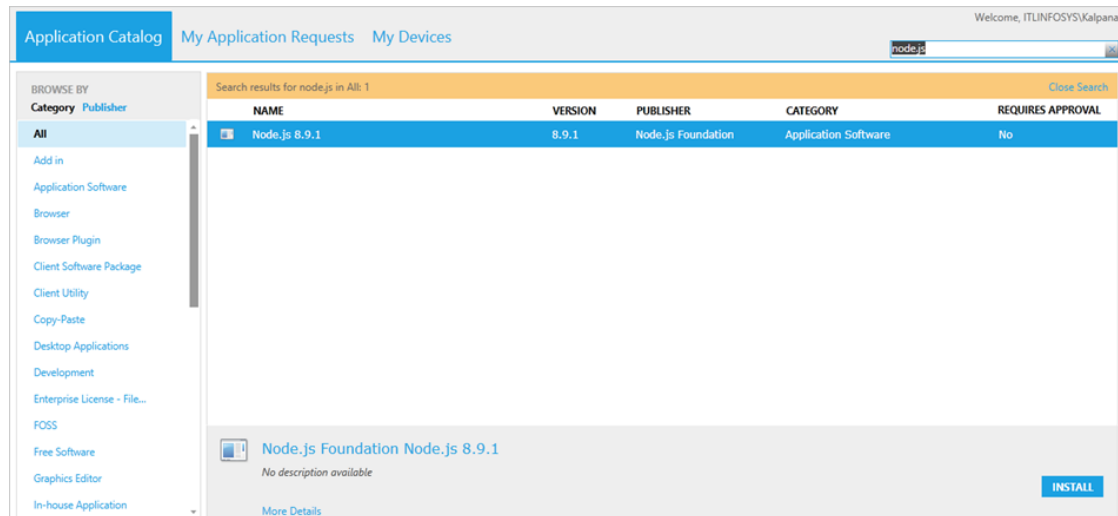
REST Client is used for interacting with HTTP APIs. It sends requests and get response from service layer.

Software & Project Setup

Software's required

- Node.js (min version required 8.x)
- Express
- Visual Studio Code
- MongoDB
- postman

1. **Node.js:** Install node.js from software house/software center as shown below or take help from CCD to get it installed



To check whether node.js is installed or not in your machine, go to **node command prompt** and check the node.js version by typing the following command. It will display the version of node.js installed.

```
D:\>node -v
v8.11.2
```

2. **Express:** Install Express using the below command:

```
D:\>npm install -g express
```

Once the installation process is complete, execute the **express** command with a version check.

```
D:\>express --version
4.14.0
```

Express provides a scaffolding tool called **express generator** which helps to quickly generate an Express application with typical support for routes. The express-generator tool helps in generating the application skeleton which can be later populated with site-specific routes, templates and database calls.

For installing the express-generator tool globally, use the following command.

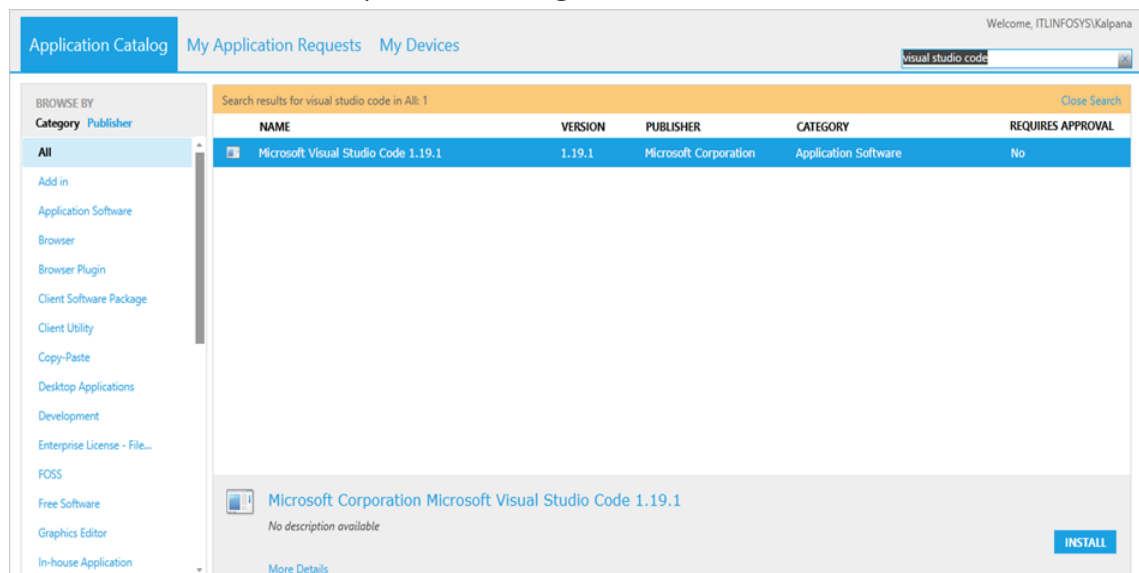
```
D:\Demos>npm install express-generator -g
```

The express-generator package installs the **express** command-line tool. Once the generator is installed, it is extremely simple to create the application using 'express' command.

Invoke the generator on the command line with a new application name.

```
D:\Demos>express <application_name>
```

3. **Visual Studio Code:** Install Visual Studio code from software house/software center as shown below or take help from CCD to get it installed.



4. **MongoDB:** Install MongoDB from Sparsh - > downloads using the below link
<http://sparshv2/portals/CCD/Downloads/Pages/Downloads.aspx>

Downloads

mongodb

Operating System

All Operating Systems

1 - 1 of 1 Software

MongoDB 3.6.4

186,292 KB - Published 68 day ago - For Windows 7 & 8.1, Linux & Mac

MongoDB is a cross-platform document-oriented database. Classified as a NoSQL database. It helps in making the integration of data in certain types of applications easier and faster.

Windows 32 bit

Windows 64 bit

Linux 32 bit

Linux 64 bit

Mac

Steps to configure MongoDB:

1. Create data\db folder in D drive, as mongodb requires a data folder to store its files
2. Open windows command prompt and go to mongodb\bin (C drive) path and give the command **mongod --dbpath "d:\data\db"** to start the mongo process
3. Open another windows command prompt and go to mongodb\bin path and give the command **mongo** to start the mongo shell

5. Postman:

Download the Postman app from <https://www.getpostman.com/apps>

← → ↻ Secure | <https://www.getpostman.com/apps>


POSTMAN


5 million developers use Postman. You should too!
Download the free app today


PRODUCT SOLUTIONS ENTERPRISE PLANS & PRICING DOCS API NETWORK SIGN IN

NATIVE APPS

Choose your platform:



Postman for Mac
For OS X Yosemite or higher
Download



Postman for Windows
For Windows 7 or later
x64 Download


Postman for Linux
x64 Download

If you want to be first in line to experience new features, download our latest [Canary build](#).

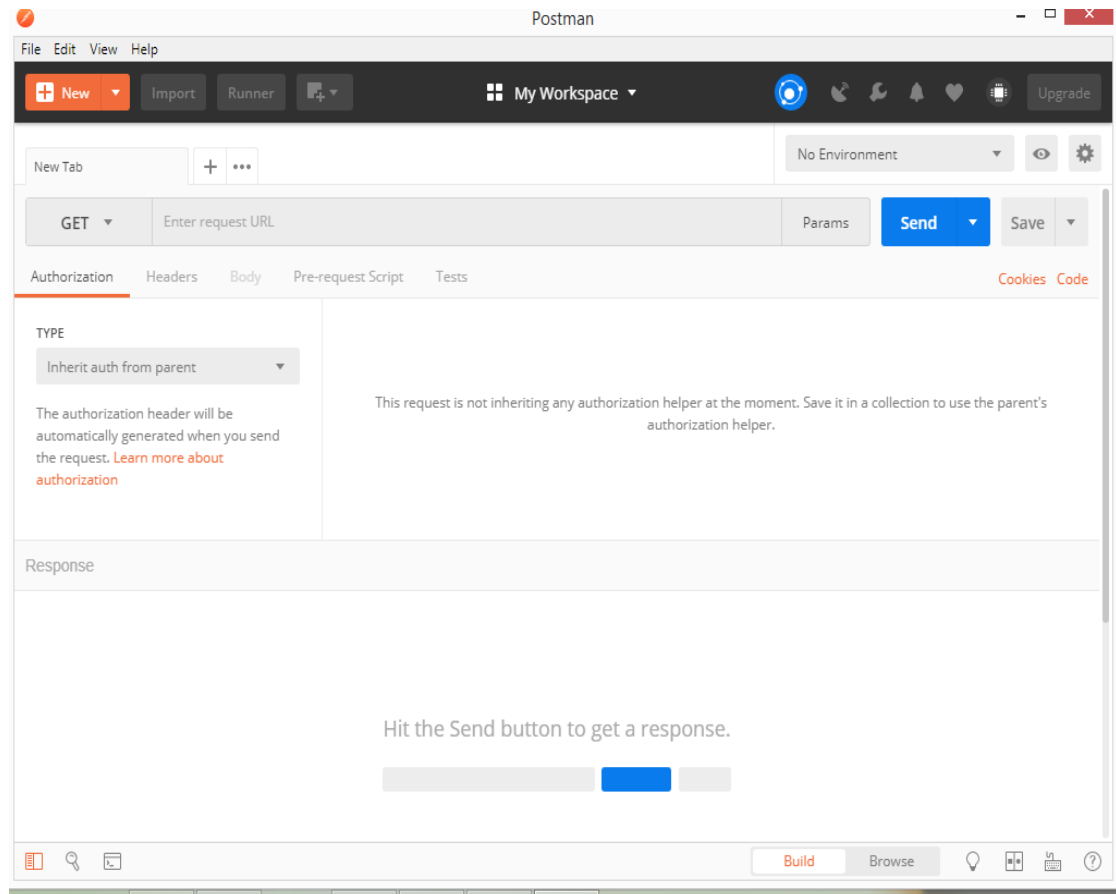
DOWNLOAD ADD-ONS


Automated testing with Newman
Postman's command-line companion lets you do amazing things! With Newman, you can integrate Postman collections with your build system. Or you can run automated tests for your API through a cron job.
Get Newman (Free)

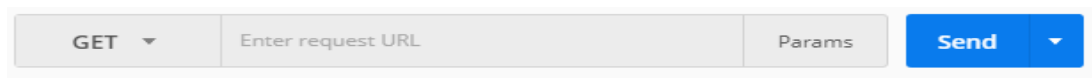

Capture and inspect with Interceptor
Postman Interceptor brings the power of your Chrome window to Postman! You can set custom headers (including cookies) from within Postman, and view cookies already set on the domain. You can also capture requests being sent from Chrome and import them into Postman. This makes building APIs a breeze!
Get Interceptor (Free)

Working with the Postman REST Client

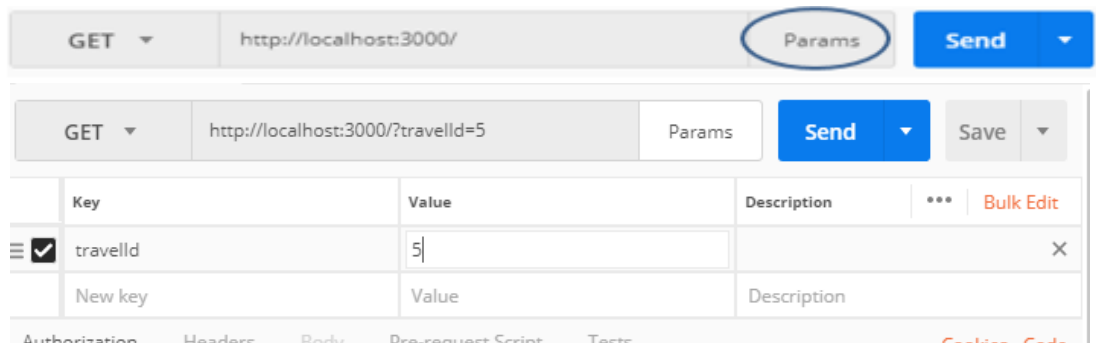
1. Screen looks as below on start of application



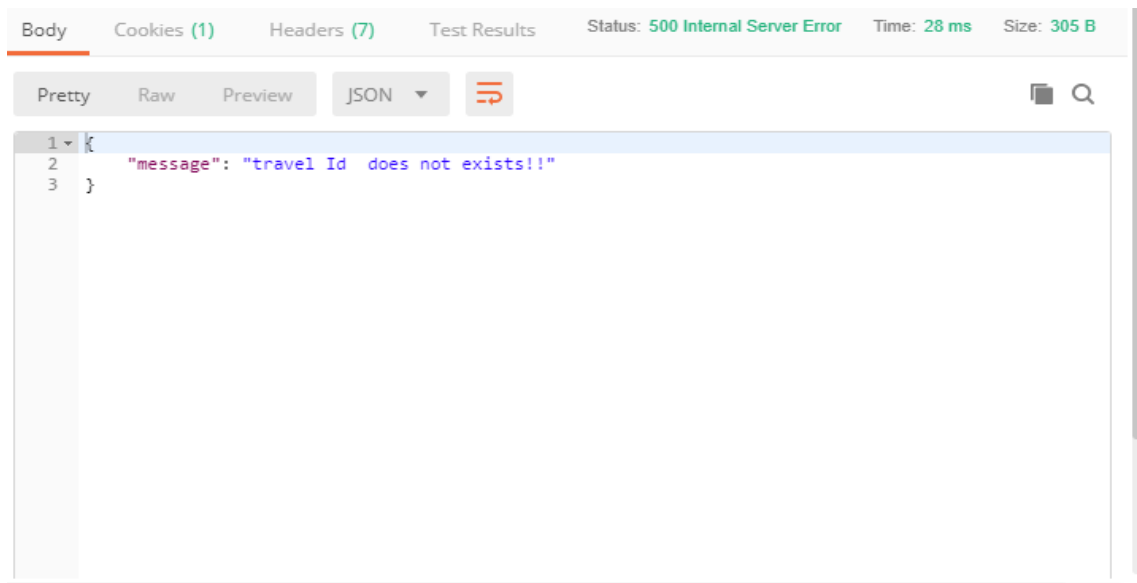
2. Enter the URL where it says 'Enter request URL' and select the method (the action type) on the left of that field. The default method is GET.



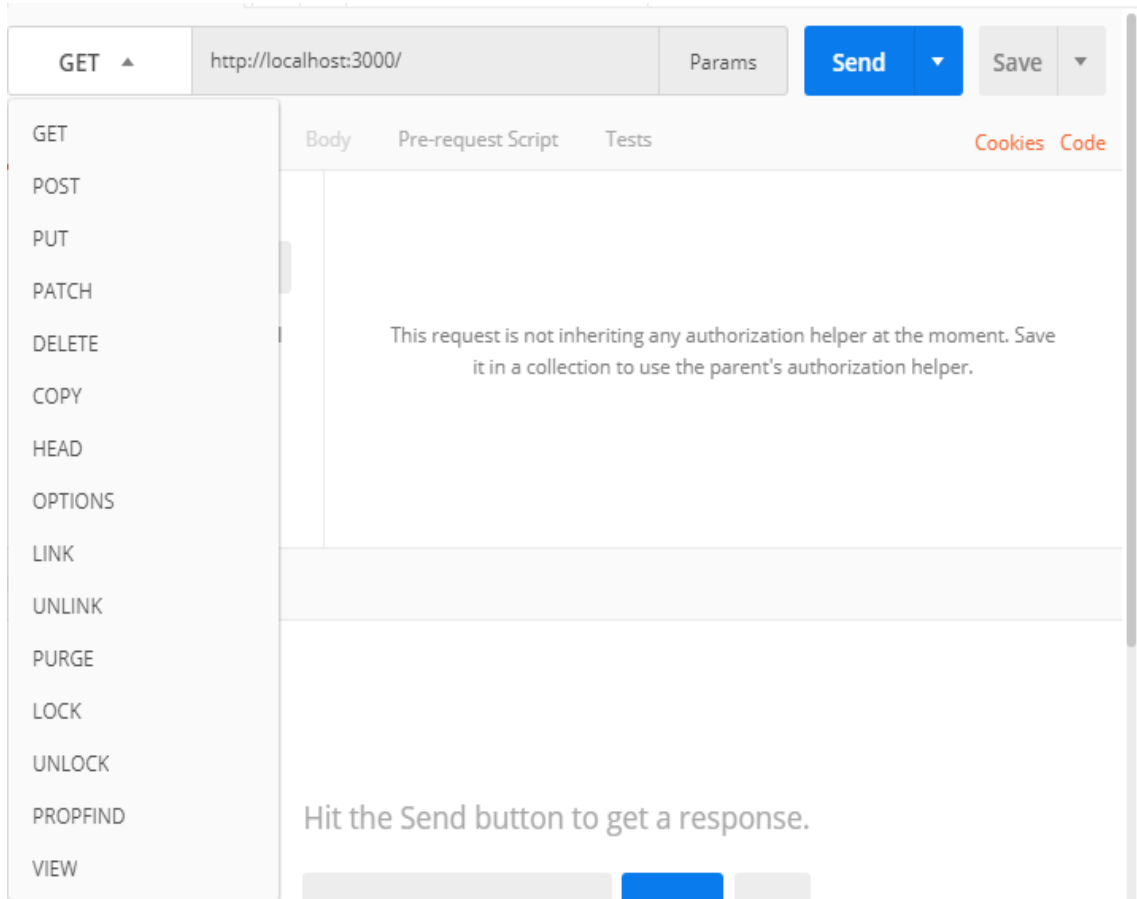
a. Click on Params button to pass parameters to the URL



- b. To execute, click Send button, which is located to the right of the API request field and observe the output



3. To change to POST request, click on GET icon and select POST from the list



a. Now select 'Body' and then 'x-www-form-urlencoded' and pass the data

POST http://localhost:3000/ Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Cookies Code

form-data x-www-form-urlencoded raw binary

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	username	admin			
<input checked="" type="checkbox"/>	password	admin			X
	New key	Value	Description		

Response

b. To execute this click Send button, which is located to the right of the API request field and observe the response in the response window

Project Reference

[Database Design](#)

Following is the list of collections to be created in MongoDB

- Users
- Offers
- Rides

a) **Collection Name: Users** This collection contains the details of users

Properties:

Property Name	Data Type	Description
userName	String	Name of the user
password	String	Password of the user

Sample Data:

{userName:" admin", password:" admin"}

b) **Collection Name: Offers** This collection contains the details of the riders

Properties:

Property Name	Data Type	Description
offerId	Number	Id for the rider
name	String	Name of the rider
car	String	Name of the car offering the ride

seatsLeft	Number	Number of seats available in the car
pickUp	String	Starting point for pickup
destination	string	Destination point for dropping

Sample Data:

{offerId: 1000, name: "admin", car: "Innova", seatsLeft:4, pickUp:"Uppal", destination: "Infosys"}

- c) **Collection Name: Rides** This collection contains the details of the booked or cancelled rides

Properties:

Property Name	Data Type	Description
rideId	Number	Id for the ride
riderName	String	Name of the rider
rideeName	String	Name of the ridee who booked the car
pickUp	String	Starting point for pickup
destination	String	Destination point for dropping
status	string	Status of the ride which should be either 'Booked' or 'Cancelled'

Sample Data:

{rideId: 1001, riderName: "admin", rideeName:"user1", pickUp:"Uppal", destination: "Infosys", status: "Booked"}

[Data Access Layer](#)

Following table lists all the requests handled in RestService along with their description

Service Url	Request Type	Request Data	Response Data	Description
/	get	{}	{ "message": "Welcome to Carpoolz application"}	Service Class should be able to display the message
/login	post	<u>Valid Credentials:</u> { userName: 'admin', password: 'admin' }	{ message: "Login successful", status: 200 }	Service class should contain functionality to fetch user's data from Users collection and should validate the credentials sent along with the request. It should return status as success or failure based on the validity check
		<u>Invalid Credentials:</u> { userName: 'ram', password: 'ram' }	{ message: "Login Unsuccessful", status: 401 }	
/show_rides	get	{}	returns array of all the available rides	Service class should fetch all rides details from Offers collection and should return it
/book_ride	post	{ "rider": { "name": "Preethi", "car": "Huidai i10", "seatsLeft": 2, "pickUp": "Hampan katta", "destination": "MN G SEZ", "offerId": 1002, "ridee": "admin" } }	{ "rideId": 1001, "seatsLeft": 1, "message": "Ride booked successfully", "status": 200 }	Service class should update seatsLeft property in Offers collection and should insert the ride details in Rides collection.
/cancel_ride	post	{rideId:1001}	{message: "Ride cancelled successfully", status: 200}	Service class should update seatsLeft property in Offers collection and change the status property to 'Cancelled' in rides collection
/offer_ride	post	{ name: 'divi', pickUp: 'Hampankatta', destination: 'MNG SEZ', car: 'Swift', seatsLeft: 3 }	{ message: "Offer added successfully", status: 200 }	Service class should insert the new ride details in Offers collection and should return the status
/logout	get	{}	{ "message": "Welcome to Carpoolz application"}	Service class should destroy the session and should be redirected to the base URL '/'
*	get, post, update, delete	{}	{ "message": "Requested URL is not available!", "status": 404 }	Service class should return custom message by using error handling middleware for URL's that are not handled.

Project Guidelines

The coding standards to be followed in PoolCarz application are as follows:

Object Type	Guide Lines	Examples
Variables	Camel casing	userName
Methods	Pascal casing	ShowAllRides
Properties	Camel casing	status

Project Implementation

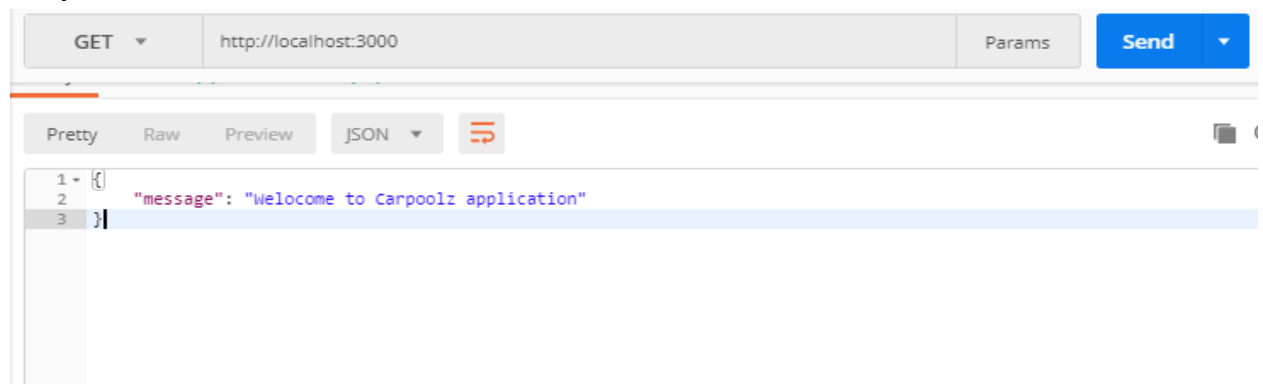
From Postman, use the below URLs for the corresponding functionalities

1. Home: <http://localhost:3000>
2. Login: <http://localhost:3000/login>
3. Show All Rides: http://localhost:3000/show_rides
4. Book Ride: http://localhost:3000/book_ride
5. Cancel Ride: http://localhost:3000/cancel_ride
6. Offer Ride: http://localhost:3000/offer_ride
7. Logout: <http://localhost:3000/logout>
8. Error handling: http://localhost:3000/invalid_URL

1. Home: <http://localhost:3000>

Response Data: {"message": "Welcome to Carpoolz application"}

Output:



2. Login: <http://localhost:3000/login>

a) Valid Credentials:

Request Data: {userName: 'admin', password: 'admin'}

Response Data: {message: "Login successful", status: 200}

Output:

The screenshot shows a REST client interface with a POST request to `http://localhost:3000/login`. The request body is `x-www-form-urlencoded` with parameters `userName=admin` and `password=admin`. The response is a JSON object: `{ "message": "Login successful", "status": 200 }`. The status is 200 OK and the time taken is 28 ms.

Key	Value	Description
userName	admin	
password	admin	
New key	Value	Description

```
1 {  
2   "message": "Login successful",  
3   "status": 200  
4 }
```

b) Invalid Credentials:

Request Data: {userName: 'ram', password: 'ram' }

Response Data: {message: "Login Unsuccessful", status: 401}

Output:

The screenshot shows a REST client interface with a POST request to `http://localhost:3000/login`. The request body is `x-www-form-urlencoded` with parameters `userName=ram` and `password=ram`. The response is a JSON object: `{ "message": "Login Unsuccessful", "status": 401 }`. The status is 200 OK and the time taken is 52 ms.

Key	Value	Description
userName	ram	
password	ram	
New key	Value	Description

```
1 {  
2   "message": "Login Unsuccessful",  
3   "status": 401  
4 }
```

3. Show All Rides: http://localhost:3000/show_rides

Response Data: Returns array of all the available rides

Output:

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:3000/show_rides
- Status:** 200 OK
- Time:** 17 ms
- Size:** 956 B

The response body is displayed in JSON format, showing an array of three ride objects:

```
1 [
2   {
3     "_id": "5b7cf3f19e1f950c8ce8046c",
4     "name": "Krishna",
5     "car": "Swift",
6     "seatsLeft": 2,
7     "pickUp": "MNG SEZ",
8     "destination": "Pumpwell",
9     "__v": 0
10  },
11  {
12    "_id": "5b7cf3f19e1f950c8ce8046d",
13    "name": "Shiva",
14    "car": "Audi",
15    "seatsLeft": 3,
16    "pickUp": "MNG SEZ",
17    "destination": "Kottara",
18    "__v": 0
19  },
20  {
21    "_id": "5b7cf3f19e1f950c8ce8046e",
22    "name": "Sneha",
23    "car": "Honda",
24    "seatsLeft": 1,
25    "pickUp": "MNG SEZ",
26    "destination": "Kottara",
27    "__v": 0
28  }
29 ]
```

4. Book Ride: http://localhost:3000/book_ride

Request Data:

```
{ "rider": { "name": "Preethi",
"car": "Huidai i10",
"seatsLeft": 2,
"pickUp": "Hampankatta",
"destination": "MNG SEZ",
"offerId": 1002},
"ridee": "admin" }
```

Response Data:

```
{
  "rideId": 1001,
  "seatsLeft": 1,
  "message": "Ride booked successfully",
  "status": 200
}
```

Output:

POST http://localhost:3000/book_ride

Authorization Headers (1) Body Pre-request Script Tests Cookies Code

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {
2   "rider": {"name": "Preethi", "car": "Huidai i10", "seatsLeft": 2, "pickup": "Hampankatta", "destination": "MNG SEZ",
3   "offerId": 1002, "ridee": "admin"}
4 }
```

Body Cookies (1) Headers (11) Test Results Status: 200 OK Time: 1300 ms Size: 620 B

Pretty Raw Preview JSON Save Response

```
1 {
2   "rideId": 1001,
3   "seatsLeft": 1,
4   "message": "Ride booked successfully",
5   "status": 200
6 }
```

5. Cancel Ride: http://localhost:3000/cancel_ride

Request Data: {rideId:1001}

Response Data: {message: "Ride cancelled successfully", status: 200}

Output:

POST http://localhost:3000/cancel_ride

Params Send

Key	Value	Description	**
New key	Value	Description	

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

Key	Value	Description	**
<input checked="" type="checkbox"/> rideId	1001		
New key	Value	Description	

Body Cookies (1) Headers (10) Test Results Status: 200 OK Time: 2054 ms

Pretty Raw Preview JSON

```
1 {
2   "message": "Ride cancelled successfully",
3   "status": 200
4 }
```

6. Offer Ride: http://localhost:3000/offer_ride

Request Data:

```
{ name: 'divi',  
  pickUp: 'Hampankatta',  
  destination: 'MNG SEZ',  
  car: 'Swift',  
  seatsLeft: 3 }
```

Response Data:

```
{ message: "Offer added successfully", status: 200 }
```

POST http://localhost:3000/offer_ride Params Send

Authorization Headers (1) Body Pre-request Script Tests

☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary

	Key	Value	Description
<input checked="" type="checkbox"/>	name	divi	
<input checked="" type="checkbox"/>	car	Swift	
<input checked="" type="checkbox"/>	seatsLeft	3	
<input checked="" type="checkbox"/>	pickUp	Hampankatta	
<input checked="" type="checkbox"/>	destination	MNG SEZ	
	New key	Value	Description

Body Cookies (1) Headers (10) Test Results Status: 200 OK Time: 30 m

Pretty Raw Preview JSON

```
1 {  
2   "message": "Offer added successfully",  
3   "status": 200  
4 }
```

7. Logout: <http://localhost:3000/logout>

Output:

GET <http://localhost:3000/logout> Params Send

Pretty Raw Preview JSON

```
1 {  
2   "message": "Welcome to Carpoolz application"  
3 }
```

8. Error handling: http://localhost:3000/invalid_URL

Response Data:

```
{  
  "message": "Requested URL is not available!",  
  "status": 404  
}
```

Output:

The screenshot shows a REST client interface with the following components:

- Request Bar:** Method: GET, URL: http://localhost:3000/invalid_URL, Params: empty.
- Buttons:** Send (blue), Save (grey).
- Tabs:** Authorization, Headers (selected), Body, Pre-request Script, Tests.
- Headers Table:**

Key	Value	Description
New key	Value	Description
- Body Tab:** Shows the response in JSON format:

```
{  
  "message": "Requested URL is not available!",  
  "status": 404  
}
```
- Summary Bar:** Status: 200 OK, Time: 46 ms, Size: 500 B.
- Response Format:** Pretty, Raw, Preview (selected), JSON (selected).