# Documentation

## Backend

/server/routes

[ Functions called in /server/controller/authenticationController or /server/controller/listingController respectively. ]

**authenticationRoutes.js**

loginUser - **POST** /api/user/login

- Verify the user's information matches the email and password of a user in the database.
- Parameters:
  - email: String
  - password: String
- Example Request:

```
{
    "email": "user_test3@carns.dev",
    "password": "Password123!"
}
```

- Responses:
  - 200 - Successful user login (matches user in database)
  - 400 - Error, improper field input/user not found in database.

signupUser - **POST** /api/user/signup

- Create a user in the database given their email, password and user type.
- Parameters:
  - email: String
  - password: String
  - userType: String
  - profile: profileSchema
  -
- Example Request:

```
{
    "email": "user_test@carns.dev",
    "password": "Password123!",
    "userType": "buyer"
```

```
        "profile": {
            "name": "Foo Bar",
            "phone_number": 1234567890
        }
    }
```

- Responses:
    - 200 - Successful creation of the user in the database.
    - 400 - Error, improper field input/failed to create the user in the database.

getProfile - **GET** /api/user/profile/:id

- Get the profile details of a user given their ID.
- Parameters/Example Request:
    - None
- Reponses:
    - 200 - Successfully got the user's profile, returns profile subschema JSON.
    - 404 - Error, not a valid ID or there is no such user.

editProfile - **PATCH** /api/user/profile/:id

- Edit a user's profile details given their ID.
- Parameters/Example Request:
    - None
- Reponses:
    - 200 - Successfully edit the user's profile, returns the edited profile subschema JSON.
    - 404 - Error, not a valid ID or there is no such user.

deleteUser - **DELETE** /api/user/:id

- Deletes a user from the database given their ID.
- Parameters/Example Request:
    - None
- Reponses:
    - 200 - Successfully deleted the user, returns a JSON of the deleted user.
    - 404 - Error, not a valid ID or there is no such user.

authenticated - **GET** /api/user/authenticated

- Authenticates the user's session with a cookie.
- Parameters/Example Request:
    - None
- Reponses:

    - 200 - Successfully created a session with the user.
    - 401 - Error, user not authenticated properly.

logout - **GET** /api/user/logout

- Clears the user's session and deletes the cookie.
- Parameters/Example Request:
    - None
- Reponses:
    - 200 - Successfully deleted the user session.

**listingRoutes.js**

postBuyListing - **POST** /api/listing/post-buy

- Create a listing for sale in the database.
- Parameters:
    - listingName: String
    - isBuy: Boolean
    - buyListingDetails: listingDetails
- Example Request:

```
{
        "listingName": "Test Listing 3",
        "isBuy": true
        "listingDetails": {
                "listingDescription": "Test description.",
                "vehicleType": "Car",
                "salePrice": 30000
        }
}
```

- Responses:
    - 200 - Successful creation of the listing.
    - 400 - Error creating the listing, improper field input/failed to create a listing in the database.

postRentListing - **POST** /api/listing/post-rent

- Create a listing for rent in the database.
- Parameters:
    - vendorID: String

    - listingName: String
    - isBuy: Boolean
    - rentListingDetails: listingDetails
- Example Request:

```
{
        "listingName": "Test Listing 3",
        "isBuy": false
        "rentListingDetails": {
                "listingDescription": "Test description.",
                "vehicleType": "Car",
                "rentPrice": 30000,
                "availabilityStart": "10/11/2022",
                "availabilityEnd": "10/28/2022",
        }
}
```

- Responses:
    - 200 - Successful creation of the listing.
    - 400 - Error creating the listing, improper field input/failed to create a listing in the database.

viewBuyListings - **GET** /api/listing/view-buy

- Get all the listings for sale in the database (in descending creation order).
- Parameters/Example Request:
    - None
- Reponses:
    - 200 - All the listing JSON objects are returned in descending creation order.
    - 400 - Error when returning all of the listings in the database.

viewRentListings - **GET** /api/listing/view-rent

- Get all the listings for rent in the database (in descending creation order).
- Parameters/Example Request:
    - None
- Reponses:
    - 200 - All the listing JSON objects are returned in descending creation order.
    - 400 - Error when returning all of the listings in the database.

getDetailBuy - **GET** /api/listing/view-detail-buy/:id

- Get a specific buy listing from the database.
- Parameters/Example Request:
    - None
- Reponses:
    - 200 - Get the buy listing details (returns the subschema JSON)
    - 404 - Error, not a valid listing ID, no such listing or missing the buy listing details.

updateBuyListing - **PUT** /api/listing/update-buy/:id

- Edit a buy listing from the database given the listing ID.
- Parameters:
    - id
    - newListingDescription
    - newSalePrice
    - newLocation
- Example Request:

```
{
        "newSalePrice": 213,
        “newLocation”: “Toronto”
}
```

- Reponses:
    - 200 - Successful listing update, returns the updated listing JSON.
    - 404 - Error when editing, couldn’t find the listing or invalid ID.

updateRentListing - **PUT** /api/listing/update-rent/:id

- Edit a rent listing from the database given the listing ID.
- Parameters:
    - id
    - newListingDescription
    - newRentPrice
    - newLocation
- Example Request:

```
{
        "newRentPrice": 7987,
        "newListingDescription": "This is a car."
}
```

- Reponses:
    - 200 - Successful listing update, returns the updated listing JSON.
    - 404 - Error when editing, couldn't find the listing or invalid ID.

deleteListing - **DELETE** /api/listing/:id

- Delete a listing from the database given its ID.
- Parameters/Example Request:
    - None
- Reponses:
    - 200 - Successful deletion, returns the deleted listing JSON.
    - 404 - Error when deleting, couldn't find the listing or invalid ID.

## /server/models

**authenticationModel.js**

Schema for user login, authentication and user profiles.

- email: String, required
    - Email of the user for login.
- password: String, required
    - Password for user login.
- userType: String, required
    - Type of user: "buyer" or "vendor"
- profile: profileSchema, required
    - name: String
        - Name of user.
    - phone_number: String
        - Phone number of user.

**listingModel.js**

Schema for the vendor listing.

- vendorID: String, required
- listingName: String, required
- isBuy: Boolean
    - If true, the listing is for Buyers (single sale) and if false, the listing is for renters.
- buyListingDetails: buyListingDetails
    - listingDescription:  String

        - Optional description of the listing.
    - vehicleType: String
        - Type of vehicle, eg. “Car”, “Bike” etc.
    - salePrice: Number
        - Sale price
    - location: String
- rentListingDetails: rentListingDetails
    - listingDescription:  String
        - Optional description of the listing.
    - vehicleType: String
        - Type of vehicle, eg. “Car”, “Bike” etc.
    - rentPrice: Number
        - rent price per day
    - location: String
    - availabilityStart: Date
        - Start date for the period the vehicle is available for rent
    - availabilityEnd: Date
        - End date for the period the vehicle is available for rent
    - allUnavailableDates: [Date]
        - All dates for which the rent listing is unavailable.
    - booking:
        - customerID: String
        - dates: [Date]
            - All the dates the specific customer has rented out

## Frontend

/client/src

**App.js**
This file contains the mainline logic of the frontend. It displays the navigation bar and landing page and contains routes to other pages.

Methods: None

/client/src/components

**Hero.js**
This file contains the main hero component displayed on the landing page. It displays a graphic and contains button that directs the user to the sign up page

Methods:

- useAuth: this method authenticates the user and contains the current user's data. It also contains api calls from authenticationRoutes.js

**Navbar.js**
This file contains the navigation bar component which is displayed in every screen and allows users to easily switch between screens. It has a different view when the user is logged in or logged out.

Methods:

- useAuth: this method authenticates the user and contains the current user's data. It also contains api calls from authenticationRoutes.js

**Profile.js**
This file contains the user profile component. The component displays the user's profile information in a neatly formatted way, and contains a button which links to the edit profile page.

Methods:

- useAuth: this method authenticates the user and contains the current user's data. It also contains api calls from authenticationRoutes.js

**ProfileSideBar.js**
This file contains the user profile sidebar component. The component displays the navigation options in the user profile, it displays different options depending on whether the logged in user is a vendor or a buyer.

Methods:

- useAuth: this method authenticates the user and contains the current user's data. It also contains api calls from authenticationRoutes.js

**SignIn.js**
This file contains the sign in component to allow users with accounts to access the features of the site. It contains an email and password field. login will be called to authenticate the user credentials.

Methods:

- login:  this method will authenticate the fields provided by the user to sign them in

**SignUp.js**

This file contains the signup component to allow users to create an account. It contains a name field, email field, password/confirm password fields, and a checkbox to set the account type. After fields have been entered, signup will be called to create the user object in the database.

Methods:

- signup:  this method will create a user object in the database with the provided fields
-

**EditProfileForm.js**

This file contains the edit profile form component to allow users to make changes to some of their profile information. Once users fill out the new fields and click save changes, editprofile is called and their information is updated in the database.

Methods:

- useAuth: this method authenticates the user and contains the current user's data. It also contains api calls from authenticationRoutes.js
- edit: this method updates the name, email, and phone number fields of the current user

**ListingForm.js**

This file contains the form for creating a vehicle listing. Vendors can fill out fields for vehicles and after all fields have been entered, a listing object will be created in the database.

Methods:

- none

/client/src/components/pages

**BuyDetail.js**

This file contains the details of a specific vehicle listing. It displays a description, dates, vendor id, etc.

Methods:

- useAuth: this method authenticates the user and contains the current user's data. It also contains api calls from authenticationRoutes.js

**ContactInfo.js**

This file contains the contact details of a vendor in a card format

Methods:

- useAuth: this method authenticates the user and contains the current user's data. It also contains api calls from authenticationRoutes.js

**CreateListingPage.js**

This file represents the create listing page. It contains the listing form component

Methods: none

**EditProfilePage.js**

This file represents the edit profile page. It contains the edit profile form component

Methods: none

**LandingPage.js**

This file contains the combined components for the landing page. It currently contains the Hero component.

Methods: none

**BuyListing.js**

This file represents the buy listing page where users buy cars. The page will call viewListing with isBuy=true and display all listings in the database in descending order or creation.

Methods:

- fetchBuyListings - retrieves all buy listing objects from the database in descending order of creation

**RentListing.js**
This file represents the rent listing page where users rent cars. The page will call viewListing with isBuy=false and display all listings in the database in descending order or creation.

Methods:

- fetchRentListings - retrieves all rent listing objects from the database in descending order of creation

**SignUpPage.js**
This represents the page for user Sign In, it contains the sign up component

Methods: none

**SignInPage.js**
This represents the page for user Sign In, it contains the sign in component

Methods: none