

# **Carry**

***Release 0.0.1***

## **Group 13**

May 24, 2020



**1**   **carry** **1**

    1.1   carry package . . . . . 1

    1.2   run module. . . . . 14

    1.3   sphinx module. . . . . 15

  

**Python Module Index** **17**

  

**Index** **19**



## 1.1 carry package

### 1.1.1 Submodules

#### 1.1.2 carry.database\_utils module

**class** `carry.database_utils.DatabaseUtils`

Bases: `object`

Database Class

---

**Note** This class only contains methods for database connection.

---

**addCalendarEventId** ( *db, booking\_id, eid* )

A function for adding a Google Calendar Event ID

**Parameters**

- **db** (*Database object, essential*) – Database
- **booking\_id** (*int, essential*) – Booking ID
- **eid** (*int, essential*) – Google Calendar Event ID

**Returns** Booking Object

**Return type** dictionary

**cancelBooking** ( *db, booking\_id* )

A function for cancelling a booking

**Parameters**

- **db** (*Database object, essential*) – Database
- **booking\_id** (*int, essential*) – Booking ID

**Returns** Booking Object

**Return type** dictionary

**createNewCar** ( *db, make, body\_type, colour, seats, cost, address, suburb, state, postcode, lat, lng* )

A function for creating a new car object in the database

**Parameters**

- **db** (*Database object, essential*) – Database
- **make** (*string, essential*) – Car Make
- **body\_type** (*string, essential*) – Car Body\_type
- **colour** (*string, essential*) – Car Colour
- **seats** (*int, essential*) – No. Car Seats
- **cost** (*float, essential*) – Car Hourly Cost
- **address** (*String, essential*) – Car Address
- **suburb** (*String, essential*) – Suburb
- **state** (*String, essential*) – State
- **postcode** (*String, essential*) – Postcode
- **lat** (*float, essential*) – Latitude
- **lng** (*float, essential*) – Longitude

**Returns** Car object if succesfully created, *None* otherwise

**Return type** dictionary

**findUser** (*db, username* )

A function for finding user by searching username or user email

**Parameters**

- **db** (*Database object, essential*) – Database
- **username** (*string, essential*) – Username / User email

**Returns** User object if succesfully found, *None* otherwise

**Return type** dictionary

**getAllCars** (*db, search\_input* )

A function for getting cars by searching features

**Parameters**

- **db** (*Database object, essential*) – Database
- **search\_input** (*string, essential*) – Search Input

**Returns** List of Car if succesfully found, an empty list otherwise

**Return type** list of dictionary

**getBookedCars** (*db, start\_datetime, end\_datetime* )

A function for getting booked cars by searching start\_datetime and end\_datetime

**Parameters**

- **db** (*Database object, essential*) – Database
- **start\_datetime** (*datetime, essential*) – Start Datetime
- **end\_datetime** (*datetime, essential*) – End Datetime

**Returns** List of Booked Car if succesfully found, an empty list otherwise

**Return type** list of dictionary

**getBooking** (*db, booking\_id* )

A function for getting booking by searching booking\_id

**Parameters**

- **db** (*Database object, essential*) – Database
- **booking\_id** (*int, essential*) – Booking ID

**Returns** Booking object if succesfully found, *None* otherwise

**Return type** dictionary

**getCar** (*db, car\_id*)

A function for getting car by searching car\_id

**Parameters**

- **db** (*Database object, essential*) – Database
- **car\_id** (*int, essential*) – Car ID

**Returns** Car object if succesfully found, *None* otherwise

**Return type** dictionary

**getCurrentBooking** (*db, user\_id, car\_id, now*)

A function for getting a current booking assigned to specific car and user

**Parameters**

- **db** (*Database object, essential*) – Database
- **user\_id** (*int, essential*) – User ID
- **car\_id** (*int, essential*) – Car ID
- **now** (*datetime, essential*) – Current Datetime

**Returns** Booking if succesfully found, *None* otherwise

**Return type** list of dictionary

**getMyBookings** (*db, user\_id*)

A function for getting all bookings made by an user

**Parameters**

- **db** (*Database object, essential*) – Database
- **user\_id** (*int, essential*) – User ID

**Returns** List of Bookings if succesfully found, an empty list otherwise

**Return type** list of dictionary

**getMyPastBookings** (*db, user\_id, now*)

A function for getting all past bookings which are cancelled or finished made by an user

**Parameters**

- **db** (*Database object, essential*) – Database
- **user\_id** (*int, essential*) – User ID
- **now** (*datetime, essential*) – Current Datetime

**Returns** List of Past Bookings if succesfully found, an empty list otherwise

**Return type** list of dictionary

**getUser** (*db, user\_id*)

A function for getting user by searching user\_id

**Parameters**

- **db** (*Database object, essential*) – Database

- **user\_id**(*int*, *essential*) – User ID

**Returns** User object if successfully found, *None* otherwise

**Return type** dictionary

**logging** ( *db*, *user\_id*, *car\_id*, *booking\_id*, *lat*, *lng*, *status*, *datetime* )

A function for logging user activities

- Parameters**
- **db** (*Database object*, *essential*) – Database
  - **user\_id** (*int*, *essential*) – User ID
  - **car\_id** (*int*, *essential*) – Car ID
  - **booking\_id** (*int*, *essential*) – Booking ID
  - **lat** (*float*, *essential*) – Latitude
  - **lng** (*float*, *essential*) – Longitude
  - **status** (*string*, *essential*) – Car Status
  - **datetime** (*datetime*, *essential*) – Current Datetime

**Returns** Log object if successfully created, *None* otherwise

**Return type** dictionary

**newBooking** ( *db*, *reference*, *user\_id*, *car\_id*, *start\_datetime*, *end\_datetime*, *booking\_datetime*, *duration*, *total\_cost* )

A function for creating a new booking object in the database

- Parameters**
- **db** (*Database object*, *essential*) – Database
  - **user\_id** (*int*, *essential*) – User ID
  - **car\_id** (*int*, *essential*) – Car ID
  - **start\_datetime** (*datetime*, *essential*) – Booking Start Datetime
  - **end\_datetime** (*datetime*, *essential*) – Booking End Datetime
  - **booking\_datetime** (*datetime*, *essential*) – Booking Made Datetime
  - **duration** (*int*, *essential*) – Booking Duration
  - **total\_cost** (*float*, *essential*) – Booking Total Cost

**Returns** Booking Object

**Return type** dictionary

**register** ( *db*, *username*, *email*, *hashed\_password*, *firstname*, *lastname* )

A function for registering a new user in the database

- Parameters**
- **db** (*Database object*, *essential*) – Database
  - **username** (*string*, *essential*) – Username
  - **email** (*string*, *essential*) – User Email
  - **hashed\_password** (*string*, *essential*) – Hashed Password
  - **firstname** (*string*, *essential*) – Firstname
  - **lastname** (*string*, *essential*) – Lastname



**Returns** User object if succesfully created, *None* otherwise

**Return type** dictionary

**updateProfile** ( *db*, *profile\_url*, *firstname*, *lastname*, *location* )

A function for updating an existing user profile in the database

**Parameters**

- **db** (*Database object*, *essential*) – Database
- **profile\_url** (*string*, *optional*) – Profile URL
- **firstname** (*string*, *essential*) – Firstname
- **lastname** (*string*, *essential*) – Lastname
- **location** (*string*, *essential*) – Location

**Returns** User object if succesfully created, *None* otherwise

**Return type** dictionary

### 1.1.3 carry.fetch module

**carry.fetch.booking\_detail\_event** ( )

Add a calendar event id to the existing booking if the user adds an event to Google Calendar.

**carry.fetch.my\_bookings\_cancel** ( )

Cancel a booking by a user request.

### 1.1.4 carry.forms module

**class** **carry.forms.BookingForm** ( *formdata*=<object object>, *\*\*kwargs* )

Bases: flask\_wtf.form.FlaskForm

A flexible form validation for Booking Detail

**Parameters** **formdata** – Used to pass data coming from the enduser, usually request.POST or equivalent. formdata should be some sort of request-data wrapper which can get multiple parameters from the form input, and values are unicode strings

**car\_id** = <UnboundField(HiddenField, ('Car ID'), {'validators': [<wtforms.validators.DataRequired object>]})>

**duration** = <UnboundField(HiddenField, ('Duration'), {'validators': [<wtforms.validators.DataRequired object>]})>

**end\_datetime** = <UnboundField(HiddenField, ('End Date'), {'validators': [<wtforms.validators.DataRequired object>]})>

**start\_datetime** = <UnboundField(HiddenField, ('Start Date'), {'validators': [<wtforms.validators.DataRequired object>]})>

**submit** = <UnboundField(SubmitField, ('Book'), {})>

**user\_id** = <UnboundField(HiddenField, ('User ID'), {'validators': [<wtforms.validators.DataRequired object>]})>

**class** **carry.forms.LoginForm** ( *formdata*=<object object>, *\*\*kwargs* )

Bases: flask\_wtf.form.FlaskForm

A flexible form validation for Login

**Parameters** **formdata** – Used to pass data coming from the enduser, usually request.POST or equivalent. formdata should be some sort of request-data wrapper which can get multiple parameters from the form input, and values are unicode strings

**password** = <UnboundField(PasswordField, ('Password'), {'validators': [<wtforms.validators.DataRequired object>, <wtforms.validators.Length object>]})>

**remember** = <UnboundField(BooleanField, ('Remember Me'), {})>

**submit** = <UnboundField(SubmitField, ('Login'), {})>

**username** = <UnboundField(StringField, ('Email / Username'), {'validators': [<wtforms.validators.DataRequired object>, <wtforms.validators.Regexp object>]})>

**class** carry.forms.**NewBookingForm** ( formdata=<object object>, \*\*kwargs )

Bases: flask\_wtf.form.FlaskForm

A flexible form validation for New Booking

**Parameters** **formdata** – Used to pass data coming from the enduser, usually request.POST or equivalent. formdata should be some sort of request-data wrapper which can get multiple parameters from the form input, and values are unicode strings

**car\_id** = <UnboundField(HiddenField, ('Car ID'), {'validators': [<wtforms.validators.DataRequired object>]})>

**duration** = <UnboundField(HiddenField, ('Duration'), {'validators': [<wtforms.validators.DataRequired object>]})>

**end\_datetime** = <UnboundField(HiddenField, ('End Date'), {'validators': [<wtforms.validators.DataRequired object>]})>

**start\_datetime** = <UnboundField(HiddenField, ('Start Date'), {'validators': [<wtforms.validators.DataRequired object>]})>

**submit** = <UnboundField(SubmitField, ('Confirm'), {'validators': [<wtforms.validators.DataRequired object>]})>

**class** carry.forms.**RegistrationForm** ( formdata=<object object>, \*\*kwargs )

Bases: flask\_wtf.form.FlaskForm

A flexible form validation for Registration

**Parameters** **formdata** – Used to pass data coming from the enduser, usually request.POST or equivalent. formdata should be some sort of request-data wrapper which can get multiple parameters from the form input, and values are unicode strings

**confirm\_password** = <UnboundField(PasswordField, ('Confirm Password'), {'validators': [<wtforms.validators.DataRequired object>, <wtforms.validators.EqualTo object>]})>

**email** = <UnboundField(StringField, ('Email'), {'validators': [<wtforms.validators.DataRequired object>, <wtforms.validators.Email object>]})>

**firstname** = <UnboundField(StringField, ('First Name'), {'validators': [<wtforms.validators.DataRequired object>, <wtforms.validators.Regexp object>]})>

**lastname** = <UnboundField(StringField, ('Last Name'), {'validators': [<wtforms.validators.DataRequired object>, <wtforms.validators.Regexp object>]})>

```
password = <UnboundField(PasswordField, ('Password'), {'validators': [<wtforms.validators.-
DataRequired object>, <wtforms.validators.Length object>]})>
```

```
submit = <UnboundField(SubmitField, ('Sign Up'), {})>
```

```
username = <UnboundField(StringField, ('Username'), {'validators': [<wtforms.validators.-
DataRequired object>, <wtforms.validators.Regexp object>]})>
```

```
validate_email ( email )
```

A function for checking if email already exists in the database

Parameters **email** (*string*, *essential*) – Email

Returns ValidationError()

Return type function

```
validate_username ( username )
```

A function for checking if username already exists in the database

Parameters **username** (*string*, *essential*) – Username

Returns ValidationError()

Return type function

```
class carry.forms.UpdateAccountForm ( formdata=<object object>, **kwargs )
```

Bases: flask\_wtf.form.FlaskForm

A flexible form validation for Profile Update

Parameters **formdata** – Used to pass data coming from the enduser, usually request.POST or equivalent. formdata should be some sort of request-data wrapper which can get multiple parameters from the form input, and values are unicode strings

```
email = <UnboundField(StringField, ('Email'), {'validators': [<wtforms.validators.DataRequired
object>, <wtforms.validators.Email object>]})>
```

```
firstname = <UnboundField(StringField, ('First Name'), {'validators': [<wtforms.validators.-
DataRequired object>, <wtforms.validators.Regexp object>]})>
```

```
lastname = <UnboundField(StringField, ('Last Name'), {'validators': [<wtforms.validators.-
DataRequired object>, <wtforms.validators.Regexp object>]})>
```

```
location = <UnboundField(SelectField, ('Location'), {'choices': [('melbourne', 'Melbourne')],
'validators': [<wtforms.validators.DataRequired object>]})>
```

```
profile_url = <UnboundField(FileField, ('Update Profile Picture'), {'validators': [<flask_wtf.-
file.FileAllowed object>]})>
```

```
submit = <UnboundField(SubmitField, ('Update'), {})>
```

```
username = <UnboundField(StringField, ('Username'), {'validators': [<wtforms.validators.-
DataRequired object>]})>
```

### 1.1.5 carry.models module

```
class carry.models.Booking ( **kwargs )
```

Bases: sqlalchemy.ext.declarative.api.Model

### Booking Database Model

Each booking in the database will be assigned a unique id value, which is a primary key.

---

**Note** Reference should be unique. Booking Model has `user_id` and `car_id` fields which are foreign keys.

---

`booking_datetime`

`calendar_eid`

`cancelled`

`car_id`

`duration`

`end_datetime`

`finished`

`id`

`logs`

`reference`

`rendBy`

`rented`

`start_datetime`

`started`

`total_cost`

`user_id`

```
class carry.models.Car ( **kwargs )
```

```
    Bases: sqlalchemy.ext.declarative.api.Model
```

Car Database Model

Each car in the database will be assigned a unique id value, which is a primary key.

`address`

`body_type`

`bookings`

`colour`

`cost`

`id`

`img_url`

lat  
lng  
logs  
make  
postcode  
seats  
state  
suburb

```
class carry.models.CarSchema ( *args, **kwargs )  
    Bases: marshmallow_sqlalchemy.schema.sqlalchemy_schema.SQLAlchemyAutoSchema  
  
    class Meta  
        Bases: object  
  
        model  
            alias of Car  
  
    opts = <marshmallow_sqlalchemy.schema.sqlalchemy_schema.SQLAlchemyAutoSchemaOpts  
    object>
```

```
class carry.models.Log ( **kwargs )  
    Bases: sqlalchemy.ext.declarative.api.Model  
    Log Database Model  
    Each Log in the database will be assigned a unique id value, which is a primary key.
```

---

**Note** Log Model has user\_id, car\_id and booking\_id fields which are foreign keys.

---

booking\_id  
car\_id  
datetime  
has  
id  
lat  
lng  
logged  
logs  
status

**user\_id**

**class** carry.models.**User** ( *\*\*kwargs* )

Bases: sqlalchemy.ext.declarative.api.Model, flask\_login.mixins.UserMixin

User Database Model

Each user in the database will be assigned a unique id value, which is a primary key.

---

**Note** Username and Email should be unique.

---

**auth**

**bookings**

**email**

**firstname**

**id**

**lastname**

**location**

**logs**

**password**

**profile\_url**

**username**

carry.models.**load\_user** ( *user\_id* )

Decorator for reloading the user from the user\_id in the session :param user\_id: Primary key of User  
:type user\_id: int, essential :return User Object where id = user\_id :rtype: dictionary

### 1.1.6 carry.routes module

carry.routes.**booking\_detail** ( )

New Booking Detail Page.

carry.routes.**getDetailedMap** ( *car* )

A Method for creating a detailed Google map

carry.routes.**getMap** ( )

A Method for creating a Google map

carry.routes.**home** ( )

Find Car Page.

carry.routes.**index** ( )

Landing Page.

carry.routes.**login** ( )

User Login Page.

`carry.routes.logout ( )`

User Logout Page.

`carry.routes.my_bookings ( )`

My Bookings Page.

`carry.routes.new_booking ( )`

New Booking Page.

`carry.routes.profile ( )`

User Profile Page.

`carry.routes.register ( )`

User Registration Page.

`carry.routes.save_picture ( picture )`

A Method for saving a picture from user input

### 1.1.7 carry.socket module

`carry.socket.background_thread ( )`

Background Thread which runs to receive messages from Agent Pis

`carry.socket.decrypt ( key, ciphertext )`

A function for decryption with shared key between Agent Pis and Master Pi

`carry.socket.sendBookingID ( UDPServerSocket, address, request, user_id, car_id )`

A function for sending a booking id if there is current booking for specific user and car

`carry.socket.sendJson ( UDPServerSocket, address, msgFromServer )`

A function for sending json data to Agent pi

`carry.socket.statusUpdate ( UDPServerSocket, address, request, user_id, car_id, booking_id, lat, lng, datetime )`

A function for storing user activities in the database

`carry.socket.user_credential ( UDPServerSocket, address, username, password, car_id )`

A function for validating user inputs from Agent Pis

### 1.1.8 carry.test\_database\_utils module

`class carry.test_database_utils.TestDatabaseUtils ( methodName='runTest' )`

Bases: `unittest.case.TestCase`

This testcase is created by subclassing `unittest.TestCase`

---

**Note** To run this test class, enter this command in the project directory.

\$ `python3 -m unittest carry/test.py`

---

`classmethod setUpClass ( )`

A class method called before tests in an individual class are run.

`setUpClass` is called with the class as the only argument and must be decorated as a class-method()

**classmethod tearDownClass ( )**

A class method called after tests in an individual class have run.

tearDownClass is called with the class as the only argument and must be decorated as a class-method():

**test\_a ( )**

A test method if database configuration is set to the test database

**test\_b\_registerUser ( )**

A test method for user registration

**test\_d\_findUser ( )**

A test method for finding user

**test\_e\_getUser ( )**

A test method for getting user

**test\_f\_dupliUser ( )**

A test method for validating duplicate user

**test\_g\_createCar ( )**

A test method for creating a new car

**test\_h\_getCar ( )**

A test method for getting car

**test\_i\_searchCar ( )**

A test method for searching car details

**test\_j\_newBooking ( )**

A test method for creating a new booking

**test\_k\_addCalendarBooking ( )**

A test method for adding calendar event id to the existing booking

**test\_l\_getMyBookings ( )**

A test method for getting all bookings by the user

**test\_m\_getBookedCar ( )**

A test method for getting booked car during the period

**test\_n\_cancelBooking ( )**

A test method for cancelling the booking

**test\_o\_getMyPastBookings ( )**

A test method for getting past bookings

**test\_p\_getCurrentBooking ( )**

A test method for getting current booking allocated to specific car and user

**test\_q\_logging ( )**

A test method for getting current booking allocated to specific car and user



### 1.1.9 carry.test\_routes module

**class** carry.test\_routes.**TestRoutes** ( *methodName='runTest'* )

Bases: unittest.case.TestCase

This class provides some basic methods for testing.

---

**Note** To run this test class, enter this command in the project directory.

```
$ python3 -m unittest carry/test.py
```

---

**classmethod** setUp ( )

The testing framework will automatically call for every single test.

**test\_booking** ( )

a method which checks if booking page works properly.

**Returns:**

Boolean

```
>>> follow_redirects=True
return True if it redirects successfully
return False if not.
```

**test\_booking\_detail** ( )

a method which checks if booking detail page works properly.

**Returns:**

Boolean

```
>>> follow_redirects=True
return True if it redirects successfully
return False if not.
```

**test\_home** ( )

a method which checks if find car page works properly.

**Returns:**

Boolean

```
>>> follow_redirects=True
return True if it redirects successfully
return False if not.
```

**test\_index** ( )

a method which checks if index page works properly.

**Returns:**

Boolean

```
>>> follow_redirects=True
return True if it redirects successfully
return False if not.
```

**test\_login** ( )

a method which checks if logout works properly.

**Returns:**

Boolean

```
>>> follow_redirects=True
return True if it redirects successfully
return False if not.
```

**test\_logout ( )**

a method which checks if logout works properly.

**Returns:**

Boolean

```
>>> follow_redirects=True
return True if it redirects successfully
return False if not.
```

**test\_my\_bookings ( )**

a method which checks if mybookings page works properly.

**Returns:**

Boolean

```
>>> follow_redirects=True
return True if it redirects successfully
return False if not.
```

**test\_profile ( )**

a method which checks if profile page works properly.

**Returns:**

Boolean

```
>>> follow_redirects=True
return True if it redirects successfully
return False if not.
```

**test\_register ( )**

a method which checks if registration page works properly.

**Returns:**

Boolean

```
>>> follow_redirects=True
return True if it redirects successfully
return False if not.
```

## 1.1.10 Module contents

## 1.2 run module

## 1.3 sphinx module

### 1.3.1 Sphinx

The Sphinx documentation toolchain.

**copyright** Copyright 2007-2020 by the Sphinx team, see AUTHORS.

**license** BSD, see LICENSE for details.

`sphinx.version_info = (3, 0, 3, 'final', 0)`

Version info for better programmatic use.

A tuple of five elements; for Sphinx version 1.2.1 beta 3 this would be (1, 2, 1, 'beta', 3).

The fourth element can be one of: `alpha`, `beta`, `rc`, `final`. `final` always has 0 as the last element.

New in version 1.2: Before version 1.2, check the string `sphinx.__version__`.

- [Index](#)
- [Module Index](#)
- [Search Page](#)



## C

`carry`, [14](#)  
    `carry.database_utils`, [1](#)  
    `carry.fetch`, [5](#)  
    `carry.forms`, [5](#)  
    `carry.models`, [7](#)  
    `carry.routes`, [10](#)  
    `carry.socket`, [11](#)  
    `carry.test_database_utils`, [11](#)  
    `carry.test_routes`, [13](#)

## r

`run`, [14](#)

## S

`sphinx`, [15](#)



**A**

addCalendarEventId() (carry.database\_utils.  
DatabaseUtils method), 1  
address (carry.models.Car attribute), 8  
auth (carry.models.User attribute), 10

**B**

background\_thread() (in module carry.socket), 11  
body\_type (carry.models.Car attribute), 8  
Booking (class in carry.models), 7  
booking\_datetime (carry.models.Booking  
attribute), 8  
booking\_detail() (in module carry.routes), 10  
booking\_detail\_event() (in module carry.fetch), 5  
booking\_id (carry.models.Log attribute), 9  
BookingForm (class in carry.forms), 5  
bookings (carry.models.Car attribute), 8  
bookings (carry.models.User attribute), 10

**C**

calendar\_eid (carry.models.Booking attribute), 8  
cancelBooking() (carry.database\_utils.DatabaseU-  
tills method), 1  
cancelled (carry.models.Booking attribute), 8  
Car (class in carry.models), 8  
car\_id (carry.forms.BookingForm attribute), 5  
car\_id (carry.forms.NewBookingForm attribute),  
6  
car\_id (carry.models.Booking attribute), 8  
car\_id (carry.models.Log attribute), 9  
carry  
    module, 14  
carry.database\_utils  
    module, 1  
carry.fetch  
    module, 5  
carry.forms

    module, 5  
carry.models  
    module, 7  
carry.routes  
    module, 10  
carry.socket  
    module, 11  
carry.test\_database\_utils  
    module, 11  
carry.test\_routes  
    module, 13  
CarSchema (class in carry.models), 9  
CarSchema.Meta (class in carry.models), 9  
colour (carry.models.Car attribute), 8  
confirm\_password (carry.forms.RegistrationForm  
attribute), 6  
cost (carry.models.Car attribute), 8  
createNewCar() (carry.database\_utils.DatabaseU-  
tills method), 1

**D**

DatabaseUtils (class in carry.database\_utils), 1  
datetime (carry.models.Log attribute), 9  
decrypt() (in module carry.socket), 11  
duration (carry.forms.BookingForm attribute), 5  
duration (carry.forms.NewBookingForm  
attribute), 6  
duration (carry.models.Booking attribute), 8

**E**

email (carry.forms.RegistrationForm attribute), 6  
email (carry.forms.UpdateAccountForm  
attribute), 7  
email (carry.models.User attribute), 10  
end\_datetime (carry.forms.BookingForm  
attribute), 5  
end\_datetime (carry.forms.NewBookingForm  
attribute), 6

end\_datetime (carry.models.Booking attribute), 8

## F

findUser() (carry.database\_utils.DatabaseUtils method), 2

finished (carry.models.Booking attribute), 8

firstname (carry.forms.RegistrationForm attribute), 6

firstname (carry.forms.UpdateAccountForm attribute), 7

firstname (carry.models.User attribute), 10

## G

getAllCars() (carry.database\_utils.DatabaseUtils method), 2

getBookedCars() (carry.database\_utils.DatabaseUtils method), 2

getBooking() (carry.database\_utils.DatabaseUtils method), 2

getCar() (carry.database\_utils.DatabaseUtils method), 3

getCurrentBooking() (carry.database\_utils.DatabaseUtils method), 3

getDetailedMap() (in module carry.routes), 10

getMap() (in module carry.routes), 10

getMyBookings() (carry.database\_utils.DatabaseUtils method), 3

getMyPastBookings() (carry.database\_utils.DatabaseUtils method), 3

getUser() (carry.database\_utils.DatabaseUtils method), 3

## H

has (carry.models.Log attribute), 9

home() (in module carry.routes), 10

## I

id (carry.models.Booking attribute), 8

id (carry.models.Car attribute), 8

id (carry.models.Log attribute), 9

id (carry.models.User attribute), 10

img\_url (carry.models.Car attribute), 8

index() (in module carry.routes), 10

## L

lastname (carry.forms.RegistrationForm attribute), 6

lastname (carry.forms.UpdateAccountForm attribute), 7

lastname (carry.models.User attribute), 10

lat (carry.models.Car attribute), 9

lat (carry.models.Log attribute), 9

lng (carry.models.Car attribute), 9

lng (carry.models.Log attribute), 9

load\_user() (in module carry.models), 10

location (carry.forms.UpdateAccountForm attribute), 7

location (carry.models.User attribute), 10

Log (class in carry.models), 9

logged (carry.models.Log attribute), 9

logging() (carry.database\_utils.DatabaseUtils method), 4

login() (in module carry.routes), 10

LoginForm (class in carry.forms), 5

logout() (in module carry.routes), 11

logs (carry.models.Booking attribute), 8

logs (carry.models.Car attribute), 9

logs (carry.models.Log attribute), 9

logs (carry.models.User attribute), 10

## M

make (carry.models.Car attribute), 9

model (carry.models.CarSchema.Meta attribute), 9

module

    carry, 14

    carry.database\_utils, 1

    carry.fetch, 5

    carry.forms, 5

    carry.models, 7

    carry.routes, 10

    carry.socket, 11

    carry.test\_database\_utils, 11

    carry.test\_routes, 13

    run, 14

    sphinx, 15

my\_bookings() (in module carry.routes), 11

my\_bookings\_cancel() (in module carry.fetch), 5

## N

new\_booking() (in module carry.routes), 11

newBooking() (carry.database\_utils.DatabaseUtils method), 4

NewBookingForm (class in carry.forms), 6

## O

opts (carry.models.CarSchema attribute), 9

## P

password (carry.forms.LoginForm attribute), 6

password (carry.forms.RegistrationForm attribute), 7

password (carry.models.User attribute), 10

postcode (carry.models.Car attribute), 9



profile() (in module carry.routes), 11  
 profile\_url (carry.forms.UpdateAccountForm attribute), 7  
 profile\_url (carry.models.User attribute), 10

## R

reference (carry.models.Booking attribute), 8  
 register() (carry.database\_utils.DatabaseUtils method), 4  
 register() (in module carry.routes), 11  
 RegistrationForm (class in carry.forms), 6  
 remember (carry.forms.LoginForm attribute), 6  
 rendBy (carry.models.Booking attribute), 8  
 rented (carry.models.Booking attribute), 8  
 run  
   module, 14

## S

save\_picture() (in module carry.routes), 11  
 seats (carry.models.Car attribute), 9  
 sendBookingID() (in module carry.socket), 11  
 sendJson() (in module carry.socket), 11  
 setUp() (carry.test\_routes.TestRoutes class method), 13  
 setUpClass() (carry.test\_database\_utils.TestDatabaseUtils class method), 11  
 sphinx  
   module, 15  
 start\_datetime (carry.forms.BookingForm attribute), 5  
 start\_datetime (carry.forms.NewBookingForm attribute), 6  
 start\_datetime (carry.models.Booking attribute), 8  
 started (carry.models.Booking attribute), 8  
 state (carry.models.Car attribute), 9  
 status (carry.models.Log attribute), 9  
 statusUpdate() (in module carry.socket), 11  
 submit (carry.forms.BookingForm attribute), 5  
 submit (carry.forms.LoginForm attribute), 6  
 submit (carry.forms.NewBookingForm attribute), 6  
 submit (carry.forms.RegistrationForm attribute), 7  
 submit (carry.forms.UpdateAccountForm attribute), 7  
 suburb (carry.models.Car attribute), 9

## T

tearDownClass() (carry.test\_database\_utils.TestDatabaseUtils class method), 12  
 test\_a() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_b\_registerUser() (carry.test\_database\_util-

s.TestDatabaseUtils method), 12  
 test\_booking() (carry.test\_routes.TestRoutes method), 13  
 test\_booking\_detail() (carry.test\_routes.TestRoutes method), 13  
 test\_d\_findUser() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_e\_getUser() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_f\_dupliUser() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_g\_createCar() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_h\_getCar() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_home() (carry.test\_routes.TestRoutes method), 13  
 test\_i\_searchCar() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_index() (carry.test\_routes.TestRoutes method), 13  
 test\_j\_newBooking() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_k\_addCalendarBooking() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_l\_getMyBookings() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_login() (carry.test\_routes.TestRoutes method), 13  
 test\_logout() (carry.test\_routes.TestRoutes method), 14  
 test\_m\_getBookedCar() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_my\_bookings() (carry.test\_routes.TestRoutes method), 14  
 test\_n\_cancelBooking() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_o\_getMyPastBookings() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_p\_getCurrentBooking() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_profile() (carry.test\_routes.TestRoutes method), 14  
 test\_q\_logging() (carry.test\_database\_utils.TestDatabaseUtils method), 12  
 test\_register() (carry.test\_routes.TestRoutes method), 14  
 TestDatabaseUtils (class in carry.test\_database\_utils), 11  
 TestRoutes (class in carry.test\_routes), 13  
 total\_cost (carry.models.Booking attribute), 8

## U

UpdateAccountForm (class in `carry.forms`), 7  
updateProfile() (`carry.database_utils.DatabaseU-`  
    `utils` method), 5  
User (class in `carry.models`), 10  
user\_credential() (in module `carry.socket`), 11  
user\_id (`carry.forms.BookingForm` attribute), 5  
user\_id (`carry.models.Booking` attribute), 8  
user\_id (`carry.models.Log` attribute), 10  
username (`carry.forms.LoginForm` attribute), 6  
username (`carry.forms.RegistrationForm`  
    attribute), 7  
username (`carry.forms.UpdateAccountForm`  
    attribute), 7  
username (`carry.models.User` attribute), 10

## V

validate\_email() (`carry.forms.RegistrationForm`  
    method), 7  
validate\_username() (`carry.forms.Registra-`  
    `tionForm` method), 7  
version\_info (in module `sphinx`), 15