Modules

routes/reservations

Express router providing reservation related routes

routes/users

Express router providing reservation related routes

Classes

Database

Represents the Database.

ParkingLot

Represents a Parking Lot.

Reservation

Represents a Reservation.

User

Represents a User.

routes/reservations

Express router providing reservation related routes

Requires: `module:express`

- routes/reservations
    - ~reservationRouter : `object`
        - ~get/reservations(path, middleware)
        - ~post/reservations(path, middleware)
        - ~delete/reservations(path, middleware)

routes/reservations~reservationRouter : `object`

Express router to mount reservation related functions on.

Kind: inner namespace of `routes/reservations`

- ~reservationRouter : `object`
    - ~get/reservations(path, middleware)
    - ~post/reservations(path, middleware)
    - ~delete/reservations(path, middleware)

reservationRouter~get/reservations(path, middleware)

Route to get reservations.

Kind: inner method of `reservationRouter`

Param	Type	Description
path	`string`	Express path
middleware	`callback`	Express middleware.

reservationRouter~post/reservations(path, middleware)

Route to post reservations.

Kind: inner method of `reservationRouter`

Param	Type	Description
path	`string`	Express path
middleware	`callback`	Express middleware.

reservationRouter~delete/reservations(path, middleware)

Route to delete reservations.

Kind: inner method of `reservationRouter`

Param	Type	Description
path	`string`	Express path
middleware	`callback`	Express middleware.

routes/users

Express router providing reservation related routes

Requires: `module:express`

- routes/users
    - ~usersRouter : `object`
        - ~get/loggedIn(path, middleware)
        - ~get/users(pathWithID, middleware)
        - ~get/users(path, middleware)

routes/users~usersRouter : `object`

Express router to mount users related functions on.

Kind: inner namespace of `routes/users`

- ~usersRouter : `object`
    - ~get/loggedIn(path, middleware)
    - ~get/users(pathWithID, middleware)
    - ~get/users(path, middleware)

usersRouter~get/loggedIn(path, middleware)

Used to pass data to dashboard html file, since GET /dashboard sends just html file without user data

Kind: inner method of `usersRouter`

Param	Type	Description
path	`string`	Express path
middleware	`callback`	Express middleware.

usersRouter~get/users(pathWithID, middleware)

Used to get user with specified ID

Kind: inner method of `usersRouter`

Param	Type	Description
pathWithID	`string`	Express path
middleware	`callback`	Express middleware.

usersRouter~get/users(path, middleware)

Used to get all users

Kind: inner method of `usersRouter`

Param	Type	Description
path	`string`	Express path

Param	Type	Description
middleware	`callback`	Express middleware.

Database

Represents the Database.

Kind: global class

- Database
  - new Database()
  - .addReservation
  - .getReservation ⇒ `object`
  - .getReservations ⇒ `Array`
  - .deleteReservation
  - .getUserById ⇒ `object`
  - .getUserByEmail ⇒ `object`
  - .getUsers ⇒ `Array`
  - .addUser
  - .updateUser ⇒ `object`
  - .authenticateUser ⇒ `object`
  - .addParkingLot

new Database()

Constructor for Database handles the connection to MongoDB.

database.addReservation

Adds a reservation to the database

Kind: instance property of `Database`

Param	Type	Description
reservation	`object`	the reservation to add to the database

database.getReservation ⇒ `object`

Get the reservation associated with email address from the database

Kind: instance property of `Database`
Returns: `object` - - Reservation

Param	Type	Description
userEmail	`string`	the email address of the user who created the reservation

database.getReservations ⇒ `Array`

Gets all reservations from the database

Kind: instance property of `Database`
Returns: `Array` - - All Reservations

database.deleteReservation

Deletes reservation for the specified email address from the database

Kind: instance property of `Database`

Param	Type	Description
email	`string`	the email address of the user who created the reservation

database.getUserById ⇒ `object`

Gets user by ID from the database

Kind: instance property of `Database`
Returns: `object` - - A user.

Param	Type	Description
id	`string`	The ID of the user.

database.getUserByEmail ⇒ `object`

Gets user by email from the database

Kind: instance property of `Database`
Returns: `object` - - A user.

Param	Type	Description
id	`string`	The email of the user.

database.getUsers ⇒ `Array`

Gets all users from the database

Kind: instance property of `Database`
Returns: `Array` - - All users.

database.addUser

Adds a user to the database

Kind: instance property of `Database`

Param	Type	Description
user	`object`	the user that is being added to the database

database.updateUser ⇒ `object`

Updates a user in the database

Kind: instance property of `Database`
Returns: `object` - - The updated user

Param	Type	Description
user	`object`	The current User

database.authenticateUser ⇒ `object`

Authenticates the a user with the database

Kind: instance property of `Database`
Returns: `object` - - If the user password and username are correct

Param	Type	Description
email	`string`	The current User
password	`string`	The users password
done	`function`	The callback function when the operation is complete

database.addParkingLot

Add a parking lot to the database

Kind: instance property of `Database`

Param	Type	Description
parkingLot	`object`	the parking lot object

ParkingLot

Represents a Parking Lot.

Kind: global class

- ParkingLot
    - new ParkingLot(db)
    - .addParkingLot()
    - .reserveSpot(spotID) ⇒ `string`
    - .isParkingSpotAvailable(spotID) ⇒ `boolean`
    - .findSpotCoordinatesX(spotID) ⇒ `number`
    - .findSpotCoordinatesY(spotID) ⇒ `number`
    - .openSpot(spotID)
    - .reservedSpot(spotID)
    - .occupiedSpot(spotID)
    - .closeSpot(spotID)
    - .storeSpot(spotID)
    - .isParkingLotFull() ⇒ `boolean`
    - .updateSpotStatus(spotID, newSpotStatus)
    - .findClosestSpot(storeID) ⇒ `number`
    - .findRandomSpot() ⇒ `number`

new ParkingLot(db)

Constructor for Parking Lot.

Param	Type	Description
db	`DataBase`	A reference to the database instance.

parkingLot.addParkingLot()

Calls database function to add the parking spots to database.

Kind: instance method of `ParkingLot`

parkingLot.reserveSpot(spotID) ⇒ `string`

Attempts to reserve a spot based on spotID

Kind: instance method of `ParkingLot`
Returns: `string` - "space reserved", "spot unavailable" or "spot does not exist"

Param	Type	Description
spotID	`number`	The id of the parking spot

parkingLot.isParkingSpotAvailable(spotID) ⇒ `boolean`

Takes ID for the spot and checks if it is "open"

Kind: instance method of `ParkingLot`
Returns: `boolean` - true if spot is available

Param	Type	Description
spotID	`number`	The id of the parking spot

parkingLot.findSpotCoordinatesX(spotID) ⇒ `number`

Takes ID for spot and returns x coordinate

Kind: instance method of `ParkingLot`
Returns: `number` - - The column index of the parking spot in the array

Param	Type	Description
spotID	`number`	The id of the parking spot

parkingLot.findSpotCoordinatesY(spotID) ⇒ `number`

Takes ID for spot and returns y coordinate

Kind: instance method of `ParkingLot`
Returns: `number` - - The row index of the parking spot in the array

Param	Type	Description
spotID	`number`	The id of the parking spot

parkingLot.openSpot(spotID)

Takes spot and updates its status to "open"

Kind: instance method of `ParkingLot`

Param	Type	Description
spotID	`number`	The id of the parking spot

parkingLot.reservedSpot(spotID)

Takes spot and updates its status to "reserved"

Kind: instance method of `ParkingLot`

Param	Type	Description
spotID	`number`	The id of the parking spot

parkingLot.occupiedSpot(spotID)

Takes spot and updates its status to "occupied" when camera detects car is present

Kind: instance method of `ParkingLot`

Param	Type	Description
spotID	`number`	The id of the parking spot

parkingLot.closeSpot(spotID)

Takes spot and updates its status to "closed"

Kind: instance method of `ParkingLot`

Param	Type	Description
spotID	`number`	The id of the parking spot

parkingLot.storeSpot(spotID)

Takes spot and updates its status to "store"

Kind: instance method of `ParkingLot`

Param	Type	Description
spotID	`number`	The id of the parking spot

parkingLot.isParkingLotFull() ⇒ `boolean`

Checks if the parking lot is full.

Kind: instance method of `ParkingLot`
Returns: `boolean` - - true if parking lot is full

parkingLot.updateSpotStatus(spotID, newSpotStatus)

Takes the parking spot's ID and the string status to be updated to

Kind: instance method of `ParkingLot`

Param	Type	Description
spotID	`number`	The id of the parking spot
newSpotStatus	`string`	The new status of the spot

parkingLot.findClosestSpot(storeID) ⇒ `number`

Takes ID for store and returns the ID for the closest "open" parking spot

Kind: instance method of `ParkingLot`
Returns: `number` - - the closest open parking spot to the store

Param	Type	Description
storeID	`number`	The id of the store

parkingLot.findRandomSpot() ⇒ `number`

Returns random "open" spot

Kind: instance method of `ParkingLot`
Returns: `number` - - random open parking spot

Reservation

Represents a Reservation.

Kind: global class

- Reservation
  - new Reservation(db, parkingLot)
  - .makeReservation(email, storeID, dateTime)
  - .getReservation(email) ⇒ `object`
  - .getReservations() ⇒ `Array`
  - .cancelReservation(email)

new Reservation(db, parkingLot)

Constructor for Reservation.

Param	Type	Description
db	`DataBase`	A reference to the database instance.

Param	Type	Description
parkingLot	`ParkingLot`	A reference to the Parking Lot instance.

reservation.makeReservation(email, storeID, dateTime)

Makes a reservation

Kind: instance method of `Reservation`

Param	Type	Description
email	`string`	the email address of the user creating the reservation
storeID	`number`	the id of the store that the quest is visiting
dateTime	`DateTime`	the date and time of the reservation

reservation.getReservation(email) ⇒ `object`

Get the reservation associated with email address

Kind: instance method of `Reservation`
Returns: `object` - - Reservation

Param	Type	Description
email	`string`	the email address of the user who created the reservation

reservation.getReservations() ⇒ `Array`

Gets all reservations

Kind: instance method of `Reservation`
Returns: `Array` - - All Reservations

reservation.cancelReservation(email)

Deletes reservation for the specified email address

Kind: instance method of `Reservation`

Param	Type	Description
email	`string`	the email address of the user who created the reservation

User

Represents a User.

Kind: global class

- User
    - new User(db)
    - .getUserById(id) ⇒ `object`
    - .getUserByEmail(id) ⇒ `object`
    - .getUsers() ⇒ `Array`
    - .addUser(firstName, lastName, phone, email, password, permissions, modified_date, modified_by, paymentMethod, vehicles)
    - .updateUser(user) ⇒ `object`
    - .authenticateUser(user, password) ⇒ `object`
    - .serializeUser(id, done) ⇒ `function`
    - .deserializeUser(id, done) ⇒ `function`

new User(db)

Constructor for User.

Param	Type	Description
db	`DataBase`	A reference to the database instance.

user.getUserById(id) ⇒ `object`

Gets user by ID

Kind: instance method of `User`
Returns: `object` - - A user.

Param	Type	Description
id	`string`	The ID of the user.

user.getUserByEmail(id) ⇒ `object`

Gets user by email

Kind: instance method of `User`
Returns: `object` - - A user.

Param	Type	Description
id	`string`	The email of the user.

user.getUsers() ⇒ `Array`

Gets all users

Kind: instance method of `User`
Returns: `Array` - - All users.

user.addUser(firstName, lastName, phone, email, password, permissions, modified_date, modified_by, paymentMethod, vehicles)

Adds a user

Kind: instance method of `User`

Param	Type	Description
firstName	`string`	the first name of the user
lastName	`string`	the last name of the user
phone	`string`	the phone number of the user
email	`string`	the email of the user
password	`string`	the password of the user
permissions	`string`	the permissions the user has
modified_date	`DateTime`	the date the user was last modified
modified_by	`string`	the identifer of the user who modified this user
paymentMethod	`string`	the payment method of the user
vehicles	`Array`	the vehicles that the user plans to make reservations for

user.updateUser(user) ⇒ `object`

Updates the current user

Kind: instance method of `User`
Returns: `object` - - The updated user

Param	Type	Description
user	`object`	The current User

user.authenticateUser(user, password) ⇒ `object`

Authenticates the current user

Kind: instance method of `User`
Returns: `object` - - If the user password and username are correct

Param	Type	Description
user	`object`	The current User
password	`string`	The users password

user.serializeUser(id, done) ⇒ `function`

Result is attached to the session as req.session.passport.user

Kind: instance method of `User`
Returns: `function` - Callback function:

Param	Type	Description
id	`string`	_id stored in mongodb
done	`function`	called internally by strategy implementation

user.deserializeUser(id, done) ⇒ `function`

In ID is used to find the user, which is then restored in req.user

Kind: instance method of `User`
Returns: `function` - Async callback function:

Param	Type	Description
id	`string`	Corresponds to _id in mongo database
done	`function`	called internally by strategy implementation