```
from flask import Flask, make_response, jsonify, request
from pymongo import MongoClient, collection
from bson import ObjectId
import jwt
import datetime
import json
from functools import wraps
import bcrypt
from flask_cors import CORS

class MyEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, ObjectId):
            return str(obj)
        return super(MyEncoder, self).default(obj)

app = Flask(__name__)
CORS(app)
app.json_encoder = MyEncoder


app.config['SECRET_KEY'] = 'mysecret'

client = MongoClient( "mongodb://127.0.0.1:27017" )
db = client.carDB
cars = db.cars
users = db.user
blacklist = db.blacklist


def jwt_required(func):
    @wraps(func)
    def jwt_required_wrapper(*args, **kwargs):
        token = None
        if 'x-access-token' in request.headers:
            token = request.headers['x-access-token']
        if not token:
            return jsonify( { 'message' : 'Token is missing' }), 401
        try:
            date = jwt.decode(token, app.config['SECRET_KEY'])
        except:
            return jsonify( { 'message' : 'Token is invalid'} ), 401

        bl_token = blacklist.find_one( { "token" : token } )
        if bl_token is not None:
            return make_response( jsonify( { 'message' : 'Token has been
cancelled'} ), 401 )
        return func(*args, **kwargs)

    return jwt_required_wrapper
```

```
def admin_required(func):
    @wraps(func)
    def admin_required_wrapper(*args, **kwargs):
        token = request.headers['x-access-token']
        data = jwt.decode( token, app.config['SECRET_KEY'] )
        if data["admin"] :
            return func(*args, **kwargs)
        else:
            return make_response( jsonify ( { 'message' : 'Admin access required' }
), 401)

    return admin_required_wrapper

@app.route("/api/v1.0/cars", methods=["GET"])
def show_all_cars():
    page_num, page_size = 1, 10
    if request.args.get("pn"):
        page_num = int(request.args.get('pn'))
    if request.args.get("ps"):
        page_size = int(request.args.get('ps'))
    page_start = (page_size * (page_num - 1))

    data_to_return = []
    for car in cars.find().skip(page_start).limit(page_size):
        car["_id"] = str(car["_id"])
        data_to_return.append(car)

    return make_response( jsonify( data_to_return ), 200 )

@app.route("/api/v1.0/cars/<string:id>", methods=["GET"])
def show_one_car(id):
    car = cars.find_one( {"_id" : ObjectId(id)})
    if car is not None:
        car["_id"] = str(car["_id"])
        return make_response( jsonify( [car] ), 200 )
    else:
        return make_response( jsonify( {"error" : "Invalid Car ID"} ), 404)

@app.route("/api/v1.0/cars", methods=["POST"])
@jwt_required
def add_car():
    if "brand" in request.form and "model" in request.form and "year_" in
request.form and "fuel" in request.form and "shifttype" in request.form and
"motorvolume" in request.form and "casetype" in request.form and "variant" in
request.form:
        new_car = {
            "brand" : request.form["brand"],
            "model" : request.form["model"],
            "year_" : request.form["year_"],
            "fuel" : request.form["fuel"],
            "shifttype" : request.form["shifttype"],
```

```
            "motorvolume" : request.form["shifttype"],
            "casetype" : request.form["casetype"],
            "variant" : request.form["variant"]
        }
        new_car_id = cars.insert_one(new_car)
        new_car_link = "http://127.0.0.1:5000/api/v1.0/cars/" +
str(new_car_id.inserted_id)
        return make_response( jsonify( { "url" : new_car_link } ), 201 )
    else:
        return make_response( jsonify( { "error" : "Missing form data" } ), 404 )

@app.route("/api/v1.0/cars/<string:id>", methods=["PUT"])
@jwt_required
def edit_car(id):
    if "brand" in request.form and "model" in request.form and "year_" in
request.form and "fuel" in request.form and "shifttype" in request.form and
"motorvolume" in request.form and "casetype" in request.form and "variant" in
request.form:
        result = cars.update_one(
            { "_id" : ObjectId(id) },
            {
                "$set" : {
                    "brand" : request.form["brand"],
                    "model" : request.form["model"],
                    "year_" : request.form["year_"],
                    "fuel" : request.form["fuel"],
                    "shifttype" : request.form["shifttype"],
                    "motorvolume" : request.form["shifttype"],
                    "casetype" : request.form["casetype"],
                    "variant" : request.form["variant"]
                }
            }
        )
        if result.matched_count == 1:
            edited_car_link = "http://127.0.0.1:50000/api/v1.0/cars/" + id
            return make_response( jsonify( { "url": edited_car_link } ), 200 )
        else:
            return make_response( jsonify( {"error" : "Invalid Car ID"} ), 404),
    else:
          return make_response( jsonify( {"error" : "Missing form data"} ), 404)


@app.route("/api/v1.0/cars/<string:id>", methods=["DELETE"])
@jwt_required
@admin_required
def delete_car(id):
    result = cars.delete_one( { "_id" : ObjectId(id) } )
    if result.deleted_count == 1:
        return make_response( jsonify( {} ), 204 )
    else:
        return make_response( jsonify( {"error" : "Invalid Car ID"} ), 404)
```

```
@app.route("/api/v1.0/cars/<string:id>/reviews", methods=["POST"])
def add_new_review(id):
    new_review = {
        "_id" : ObjectId(),
        "user_id" : "618e043eaabbeab59eaea466",
        "username" : request.form["username"],
        "comment" : request.form["comment"],
        "stars" : request.form["stars"],
        "date" : request.form["date"]
    }
    cars.update_one( { "_id" : ObjectId(id) },
         {
              "$push": { "reviews" : new_review }
         }
    )
    new_review_link ="http://127.0.0.1:5000/api/v1.0/cars/" + id +"/reviews/" +
str(new_review['_id'])

    return make_response( jsonify( { "url" : new_review_link } ), 201 )

@app.route("/api/v1.0/cars/<string:id>/reviews", methods=["GET"])
def fetch_all_reviews(id):
    data_to_return = []
    car = cars.find_one(
        { "_id" : ObjectId(id) },
        { "reviews" : 1, "_id" : 0 }
    )
    for review in car["reviews"]:
        review["_id"] = str(review["_id"])
        data_to_return.append(review)
    return make_response( jsonify( data_to_return ), 200 )

@app.route("/api/v1.0/cars/<string:id>/reviews/<string:review_id>",
methods=["GET"])
def fetch_one_review(id, review_id):
    car = cars.find_one(
        { "reviews._id" : ObjectId(review_id) },
        { "_id" : 0, "reviews.$" : 1 }
    )
    if car is None:
        return make_response( jsonify( {"error":"Invalid Car ID or review
ID"}),404)
    else:
        car['reviews'][0]['_id'] = str(car['reviews'][0]['_id'])
        return make_response( jsonify( car['reviews'][0]), 200)

@app.route("/api/v1.0/cars/<string:id>/reviews/<string:review_id>",
methods=["PUT"])
@jwt_required
def edit_review(id, review_id):
```

```
    edited_review = {
        "reviews.$.username" : request.form["username"],
        "reviews.$.comment" : request.form["comment"],
        "reviews.$.stars" : request.form['stars']
    }
    cars.update_one(
        { "reviews._id" : ObjectId(review_id) },
        { "$set" : edited_review }
    )
    edit_review_url = "http://127.0.0.1:5000/api/v1.0/cars/" + id + "/reviews/" +
review_id
    return make_response( jsonify( {"url" : edit_review_url} ), 200)

@app.route("/api/v1.0/cars/<string:id>/reviews/<string:review_id>",
methods=["DELETE"])
@jwt_required
@admin_required
def delete_review(id, review_id):
    cars.update_one(
        { "_id" : ObjectId(id) },
        { "$pull" : { "reviews" : { "_id" : ObjectId(review_id) } } }
    )
    return make_response( jsonify( {} ), 204)

@app.route('/api/v1.0/login', methods=['GET'])
def login():
    auth = request.authorization
    if auth:
        user = users.find_one( { "username" : auth.username } )
        if user is not None:
            if bcrypt.checkpw( bytes( auth.password, 'UTF-8' ), user["password"]):
                token = jwt.encode( {
                        'user' : auth.username,
                        'admin' : user["admin"],
                        'exp' : datetime.datetime.utcnow() +
datetime.timedelta(minutes=30)
                    }, app.config['SECRET_KEY'])
                return make_response( jsonify({'token' : token.decode('UTF-8') } ),
200 )
            else:
                return make_response( jsonify( {'message' : 'Bad password'} ), 401)

    return make_response( jsonify( { 'message' : 'Authentication required'} ),401 )

@app.route('/api/v1.0/logout', methods=['GET'])
@jwt_required
def logout():
    token = request.headers['x-access-token']
    blacklist.insert_one( {"token" : token } )
    return make_response( jsonify( { 'message' : 'Logout successful'} ), 200 )
```

```python
if __name__ == "__main__":
    app.run(debug=True)
```