Restaurant Reservation

Group : My Little Pony

Constraint

1. The system shall allow a user to register by specifying the name, telephone number, email, and password.

2.After registration, the user becomes a registered user, and the system shall allow the user to log in to use the system by specifying the email and password. The system shall allow a registered user to log out.

3. After login, the system shall allow the registered user to reserve up to 3 tables by specifying the date and the preferred restaurant. The restaurant list is also provided to the user. A restaurant information includes the name, address, telephone number, and open-close time.

4. The system shall allow the registered user to view his/her restaurant reservation.

5. The system shall allow the registered user to edit his/her restaurant reservation.

6. The system shall allow the registered user to delete his/her restaurant reservation.

7. The system shall allow the admin to view any restaurant reservation.

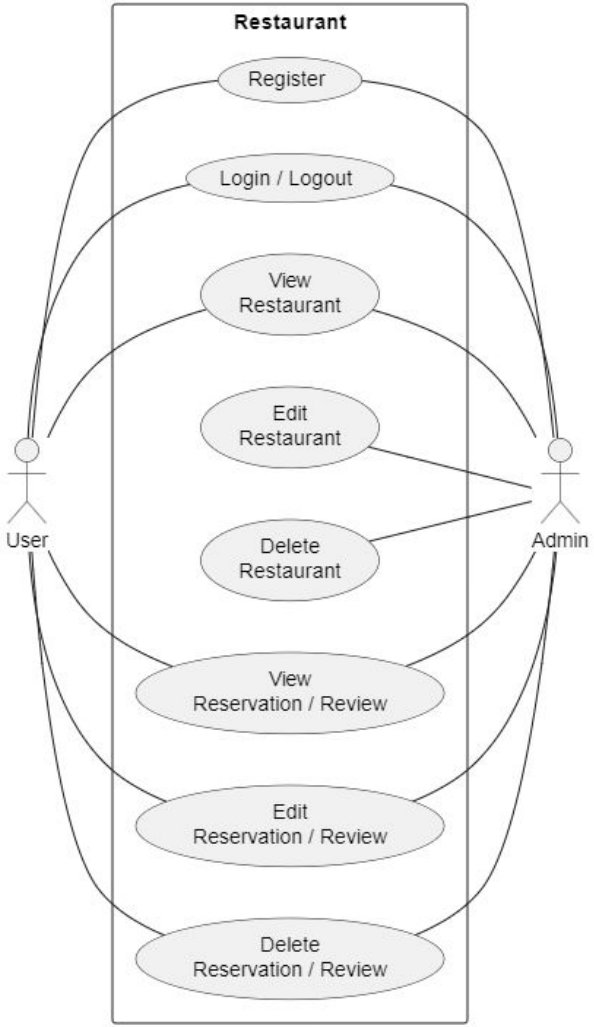
8. The system shall allow the admin to edit any restaurant reservation.

9. The system shall allow the admin to delete any restaurant reservation.

Access Rights

User Level	Operation	Data
User	CR__	Own User
	_R__	All Restaurants
	CRUD	Own Reservation / Review
Admin	CR__	All Users
	CRUD	All Restaurants
	CRUD	All Reservations / Reviews
Non-Register	_R__	All Restaurants
	____	User / Reservation / Review

Case Diagram



server.js (1/2)

```
const hospitals = require('./routes/hospitals');
const appointments = require('./routes/appointments');
```

```
const restaurants = require('./routes/restaurants');
const reservations = require('./routes/reservations');
```

server.js (2/2)

```
app.use('/api/v1/hospitals', hospitals);
app.use('/api/v1/appointments', appointments);
```

```
app.use('/api/v1/restaurants', restaurants);
app.use('/api/v1/reservations', reservations);
```

models/User.js

```
const UserSchema=new mongoose.Schema({
    name:{
        type:String,
        required:[true,'Please add a name']
    },
    email:{
        type: String,
        required:[true,'Please add an email'],
        unique: true,
        match: [
        /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)
        'Please add a valid email'
        ]
    },
    role: {
        type:String,
        enum: ['user','admin'],
        default: 'user'
    },
    password: {
        type:String,
        required:[true,'Please add a password'],
        minlength: 6,
        select: false
    },
    resetPasswordToken: String,
    resetPasswordExpire: Date,
    createdAt:{
        type: Date,
        default:Date.now
    }

});
```

```
const UserSchema=new mongoose.Schema({
    name:{
        type: String,
        required: [true,'Please add a name']
    },
    tel: {
        type: String,
        required: [true, 'Please add a telephone number']
    },
    email:{
        type: String,
        required:[true,'Please add an email'],
        unique: true,
        match: [
        /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*
        'Please add a valid email'
        ]
    },
    role: {
        type:String,
        enum: ['user','admin'],
        default: 'user'
    },
    password: {
        type:String,
        required:[true,'Please add a password'],
        minlength: 6,
        select: false
    },
    resetPasswordToken: String,
    resetPasswordExpire: Date,
    createdAt:{
        type: Date,
        default:Date.now
    }
});
```

models/Restaurant.js (1/2)

```
const HospitalSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true,'Please add a name'],
        unique: true,
        trim: true,
        maxlength: [50,'Name can not be more than 50 characters']
    },
    address:{
        type: String,
        required: [true,'Please add a address']
    },
    district:{
        type: String,
        required: [true,'Please add a district']
    },
    province:{
        type: String,
        required: [true,'Please add a province']
    },
    postalcode:{
        type: String,
        required: [true,'Please add a postalcode'],
        maxlength: [5,'Postal Code can not be more than 5 digits']
    },
    tel:{
        type: String,
    },
    region:{
        type: String,
        required: [true,'Please add a region']
    }
},{
    toJSON: {virtuals: true},
    toObject: {virtuals: true}
});
```

```
const RestaurantSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true,'Please add a name'],
        unique: true,
        trim: true,
        maxlength: [50,'Name can not be more than 50 characters']
    },
    address:{
        type: String,
        required: [true,'Please add a address']
    },
    tel:{
        type: String,
    },
    opentime:{
        type: String,
        required: [true,'Please add a restaurant opening time']
    },
    closetime:{
        type: String,
        required: [true,'Please add a restaurant closing time']
    }
},{
    toJSON: {virtuals: true},
    toObject: {virtuals: true}
});
```

models/Restaurant.js (2/2)

```
HospitalSchema.pre('deleteOne', {document: true, query: false}, async function(next){
    console.log(`Appointments being removed from hospital ${this._id}`);

    await this.model('Appointment').deleteMany({hospital: this._id});

    next();

});

HospitalSchema.virtual('appointments',{
    ref: 'Appointment',
    localField: '_id',
    foreignField: 'hospital',
    justOne:false
});

module.exports = mongoose.model('Hospital', HospitalSchema);
```

```
RestaurantSchema.pre('deleteOne', {document: true, query: false}, async function(next){
    console.log(`Reservations being removed from restaurant ${this._id}`);

    await this.model('Reservation').deleteMany({restaurant: this._id});

    next();

});

RestaurantSchema.virtual('reservations',{
    ref: 'Reservation',
    localField: '_id',
    foreignField: 'restaurant',
    justOne:false
});

module.exports = mongoose.model('Restaurant', RestaurantSchema);
```

models/Reservation.js

```
const AppointmentSchema = new mongoose.Schema({
    apptDate: {
        type: Date,
        required: true
    },
    user: {
        type: mongoose.Schema.ObjectId,
        ref: 'User',
        required: true
    },
    hospital:{
        type: mongoose.Schema.ObjectId,
        ref: 'Hospital',
        required: true
    },
    createdAt:{
        type:Date,
        default: Date.now
    }
});

module.exports = mongoose.model('Appointment', AppointmentSchema);
```

```
const ReservationSchema = new mongoose.Schema({
    reservationDate: {
        type: Date,
        required: true
    },
    user: {
        type: mongoose.Schema.ObjectId,
        ref: 'User',
        required: true
    },
    restaurant:{
        type: mongoose.Schema.ObjectId,
        ref: 'Restaurant',
        required: true
    },
    createdAt:{
        type:Date,
        default: Date.now
    }
});

module.exports = mongoose.model('Reservation', ReservationSchema);
```

routes/restaurant.js

```
const {getHospitals, getHospital, createHospital, updateHospital, deleteHospital, getVacCenters} = require('../controllers/hospitals');

const appointmentRouter = require('./appointments');

const router = express.Router();

const {protect, authorize} = require('../middleware/auth');

router.use('/:hospitalId/appointments/',appointmentRouter);

router.route('/vacCenters').get(getVacCenters);
router.route('/').get(getHospitals).post(protect,authorize('admin'), createHospital);
router.route('/:id').get(getHospital).put(protect,authorize('admin'), updateHospital).delete(protect,authorize('admin'), deleteHospital);
```

```
const {getRestaurants, getRestaurant, createRestaurant, updateRestaurant, deleteRestaurant } = require('../controllers/restaurants');

const reservationRouter = require('./reservations');

const router = express.Router();

const {protect, authorize} = require('../middleware/auth');

router.use('/:restaurantId/reservations/', reservationRouter);

router.route('/').get(getRestaurants).post(protect,authorize('admin'), createRestaurant);
router.route('/:id').get(getRestaurant).put(protect,authorize('admin'), updateRestaurant).delete(protect,authorize('admin'), deleteRestaurant);
```

routes/reservation.js

```
const {getAppointments, getAppointment, addAppointment, updateAppointment, deleteAppointment} = require('../controllers/appointments');

const router = express.Router({mergeParams: true});

const {protect, authorize} = require('../middleware/auth');

router.route('/').get(protect, getAppointments).post(protect, authorize('admin','user'), addAppointment);

router.route('/:id').get(protect, getAppointment).put(protect, authorize('admin','user'),updateAppointment).delete(protect, authorize('admin','user'),deleteAppointment);
```

```
const {getReservations, getReservation, addReservation, updateReservation, deleteReservation} = require('../controllers/reservations');

const router = express.Router({mergeParams: true});

const {protect, authorize} = require('../middleware/auth');

router.route('/').get(protect, getReservations).post(protect, authorize('admin','user'), addReservation);

router.route('/:id').get(protect, getReservation).put(protect, authorize('admin','user'),updateReservation).delete(protect, authorize('admin','user'),deleteReservation);
```

controllers/restaurant.js (GET ALL , GET , DELETE)

```
const Hospital = require('../models/Hospital');

const vacCenter = require('../models/VacCenter');

exports.getHospitals=async(req,res,next)=>{

    let query;

    const reqQuery = {...req.query};

    const removeFields = ['select', 'sort', 'page', 'limit'];

    removeFields.forEach(param=>delete reqQuery[param]);
    console.log(reqQuery);

    let queryStr=JSON.stringify(reqQuery);
    queryStr=queryStr.replace(/\b(gt|gte|lt|lte|in)\b/g, match => `$${match}`);

    query = Hospital.find(JSON.parse(queryStr)).populate('appointments');

    if(req.query.select){
        const fields = req.query.select.split(',').join(' ');
        query = query.select(fields);
    }

    if(req.query.sort){
        const sortBy = req.query.sort.split(',').join(' ');
        query = query.sort(sortBy);
    }else{
        query = query.sort('-createdAt');
    }
```

```
const Restaurant = require('../models/Restaurant');

exports.getRestaurants=async(req,res,next)=>{

    let query;

    const reqQuery = {...req.query};

    const removeFields = ['select', 'sort', 'page', 'limit'];

    removeFields.forEach(param=>delete reqQuery[param]);
    console.log(reqQuery);

    let queryStr=JSON.stringify(reqQuery);
    queryStr=queryStr.replace(/\b(gt|gte|lt|lte|in)\b/g, match => `$${match}`);

    query = Restaurant.find(JSON.parse(queryStr)).populate('reservations');

    if(req.query.select){
        const fields = req.query.select.split(',').join(' ');
        query = query.select(fields);
    }

    if(req.query.sort){
        const sortBy = req.query.sort.split(',').join(' ');
        query = query.sort(sortBy);
    }else{
        query = query.sort('-createdAt');
    }
```

controllers/restaurant.js (CREATE RESTAURANT)

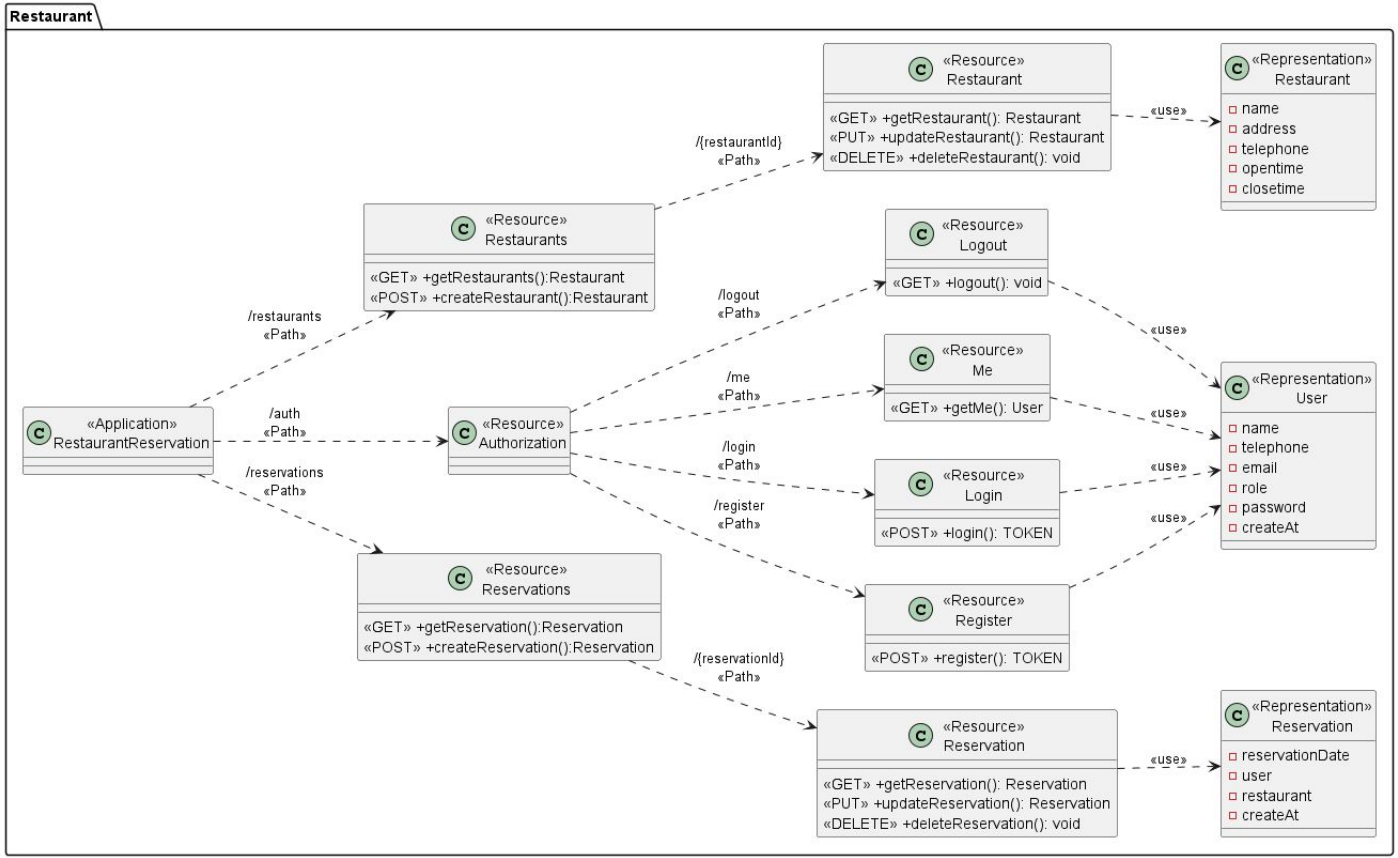
```
exports.createHospital=async (req,res,next)=>{
    const hospital = await Hospital.create(req.body);
    res.status(201).json({success:true, data:hospital});
};
```

```
exports.createRestaurant=async (req,res,next)=>{
    try{
        const restaurant = await Restaurant.create(req.body);
        res.status(201).json({success:true, data:restaurant});
    }catch(err){
        res.status(400).json({success:false, msg: "Please provide an alternative name for the restaurant, as the previous one has
    }
};
```

controllers/auth.js

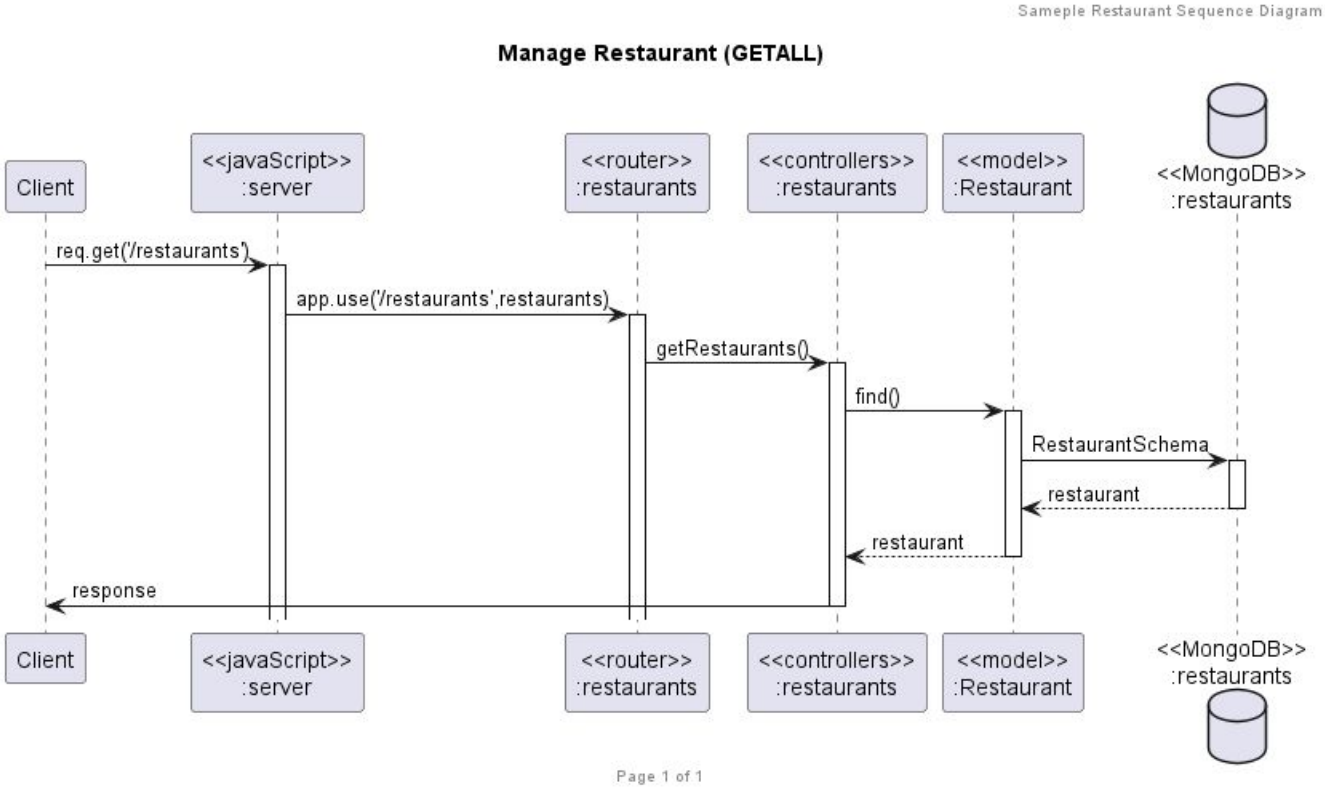
```
const sendTokenResponse = (user, statusCode, res) => {
    const token = user.getSignedJwtToken();

    const options = {
        expires: new Date(Date.now() + process.env.JWT_COOKIE_EXPIRE*24*60*60*1000),
        httpOnly: true
    };

    if(process.env.NODE_ENV === 'production'){
        options.secure = true;
    }

    res.status(statusCode).cookie('token',token,options).json({
        success:true,
        token,
        data: user
    })
}
```

Class Diagram

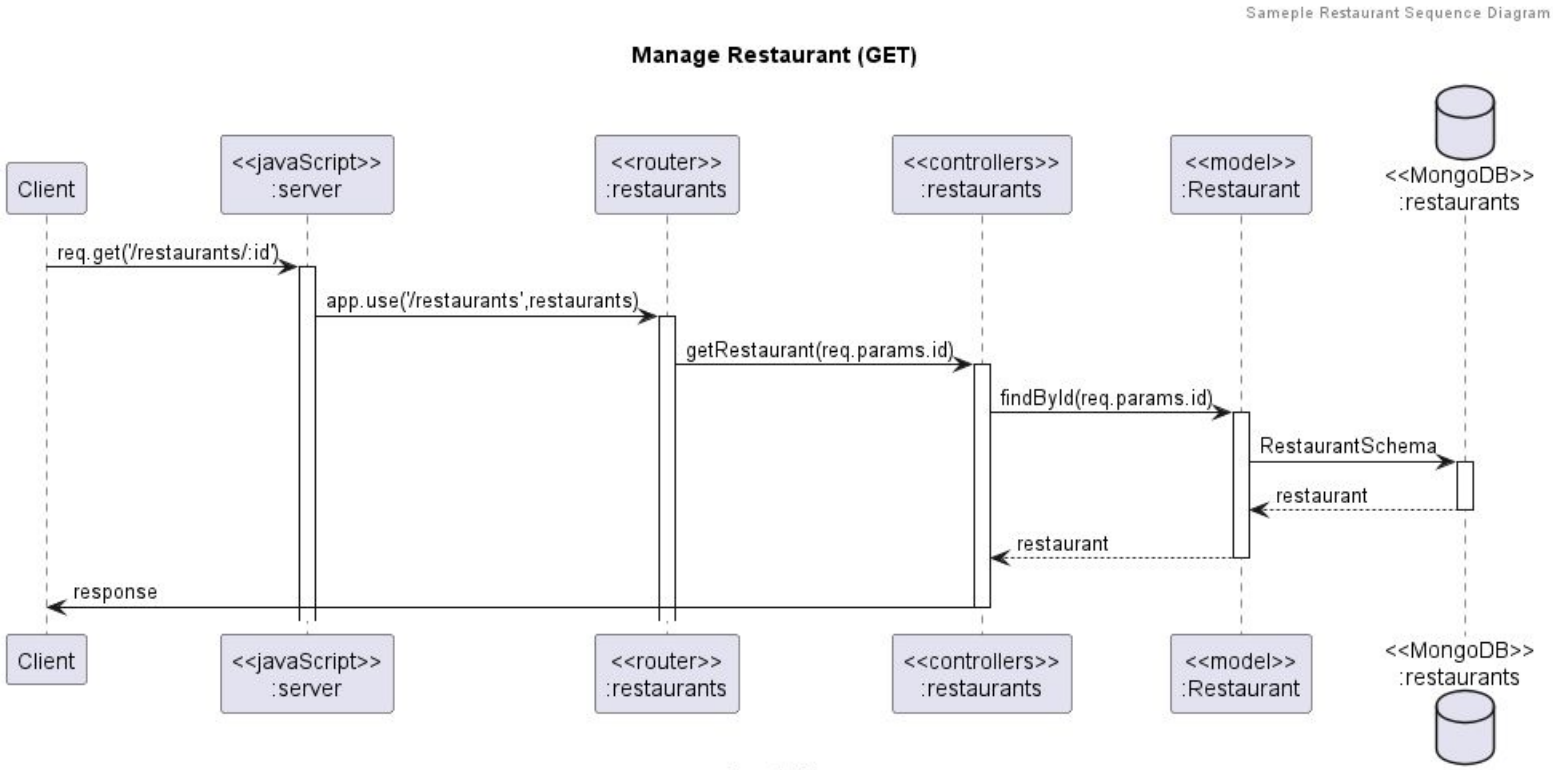


Sequence Diagram

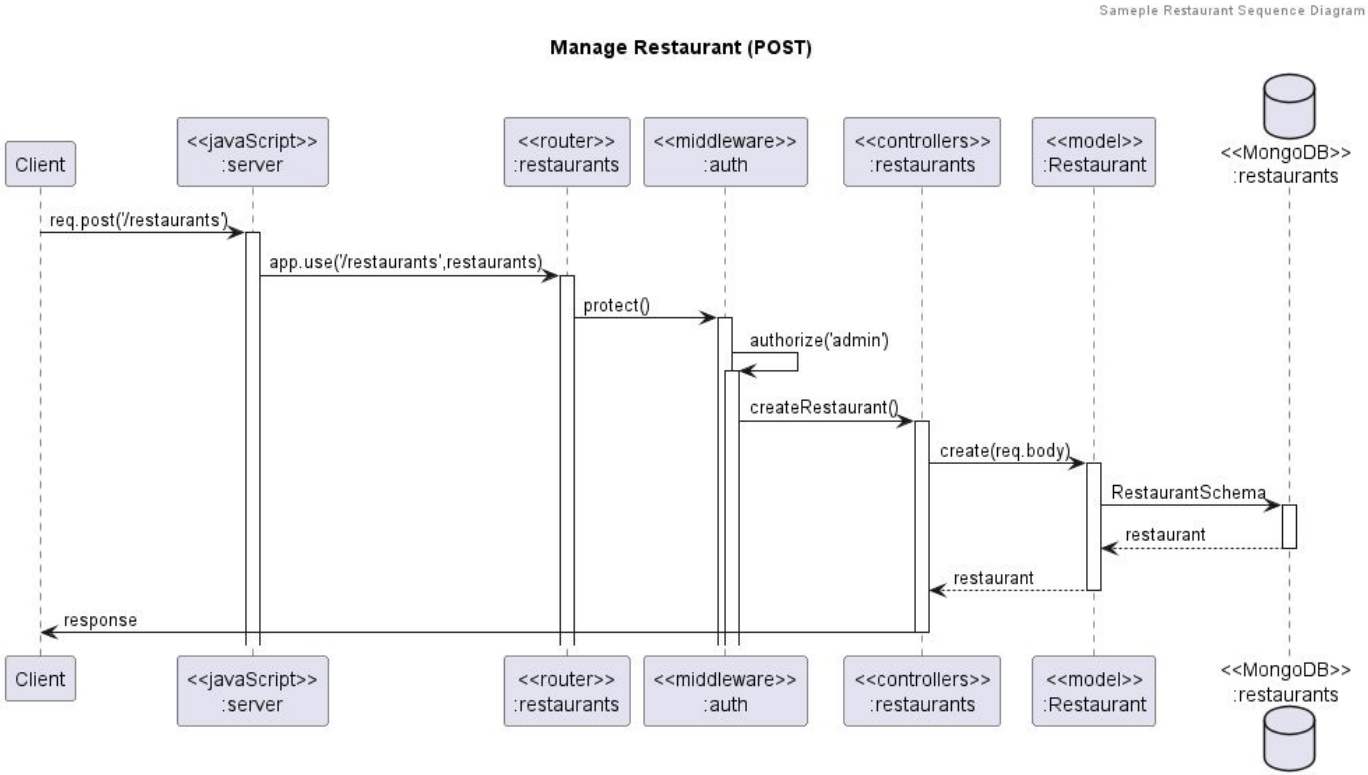
GET ALL Restaurants



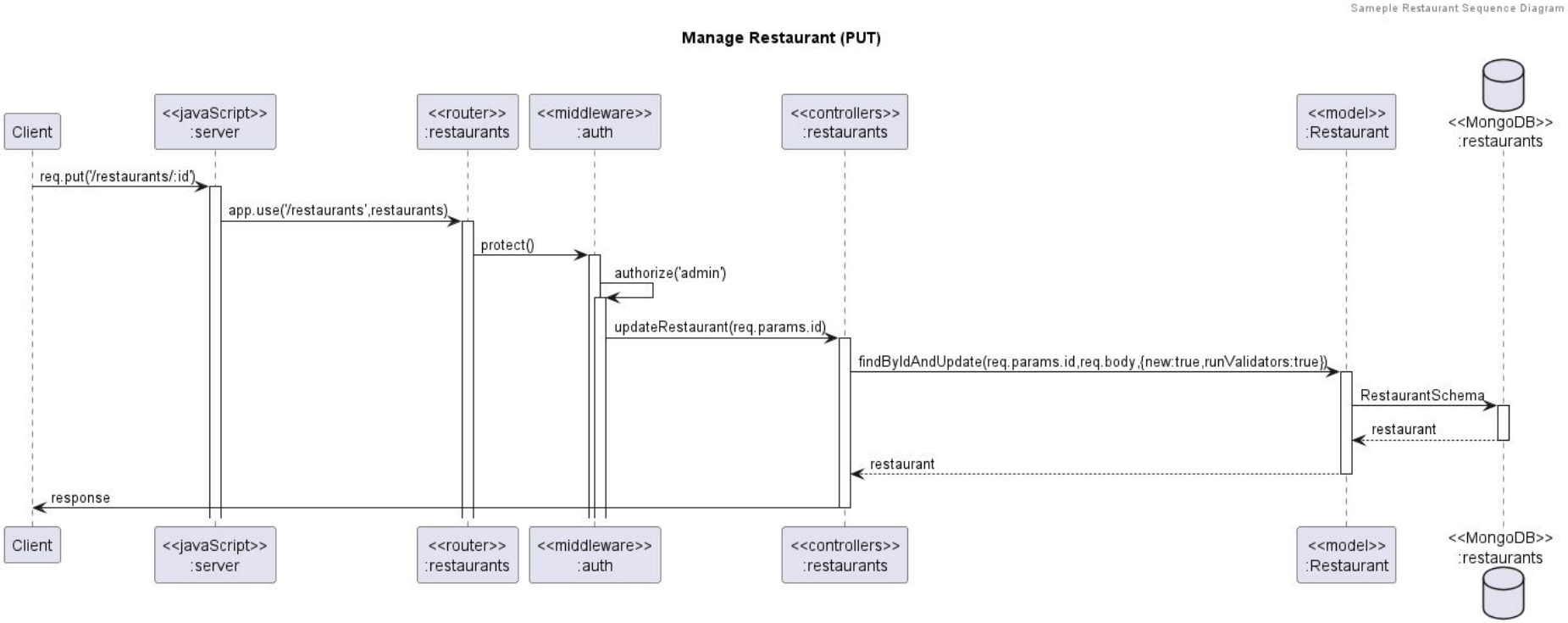
GET ONE Restaurant



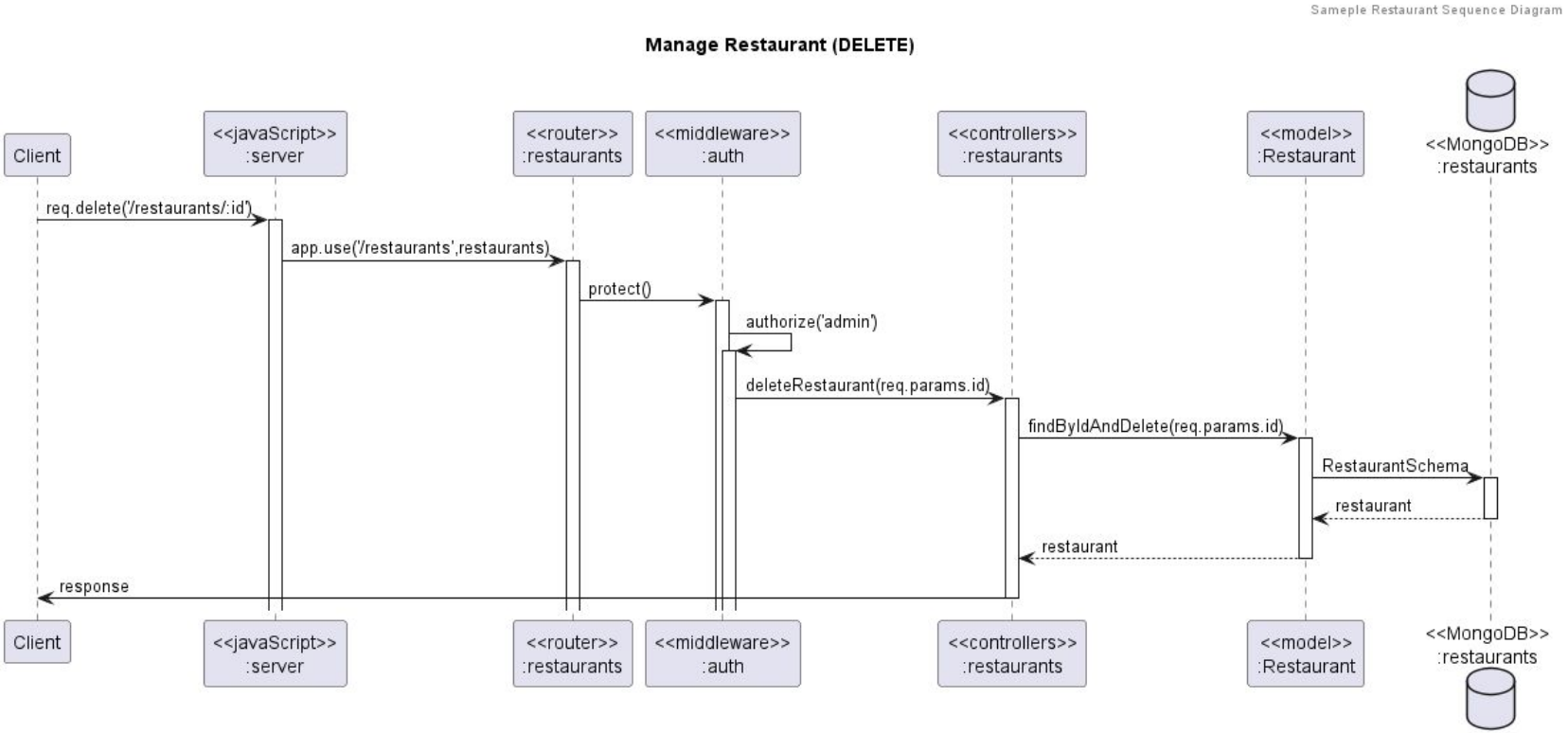
POST Restaurant



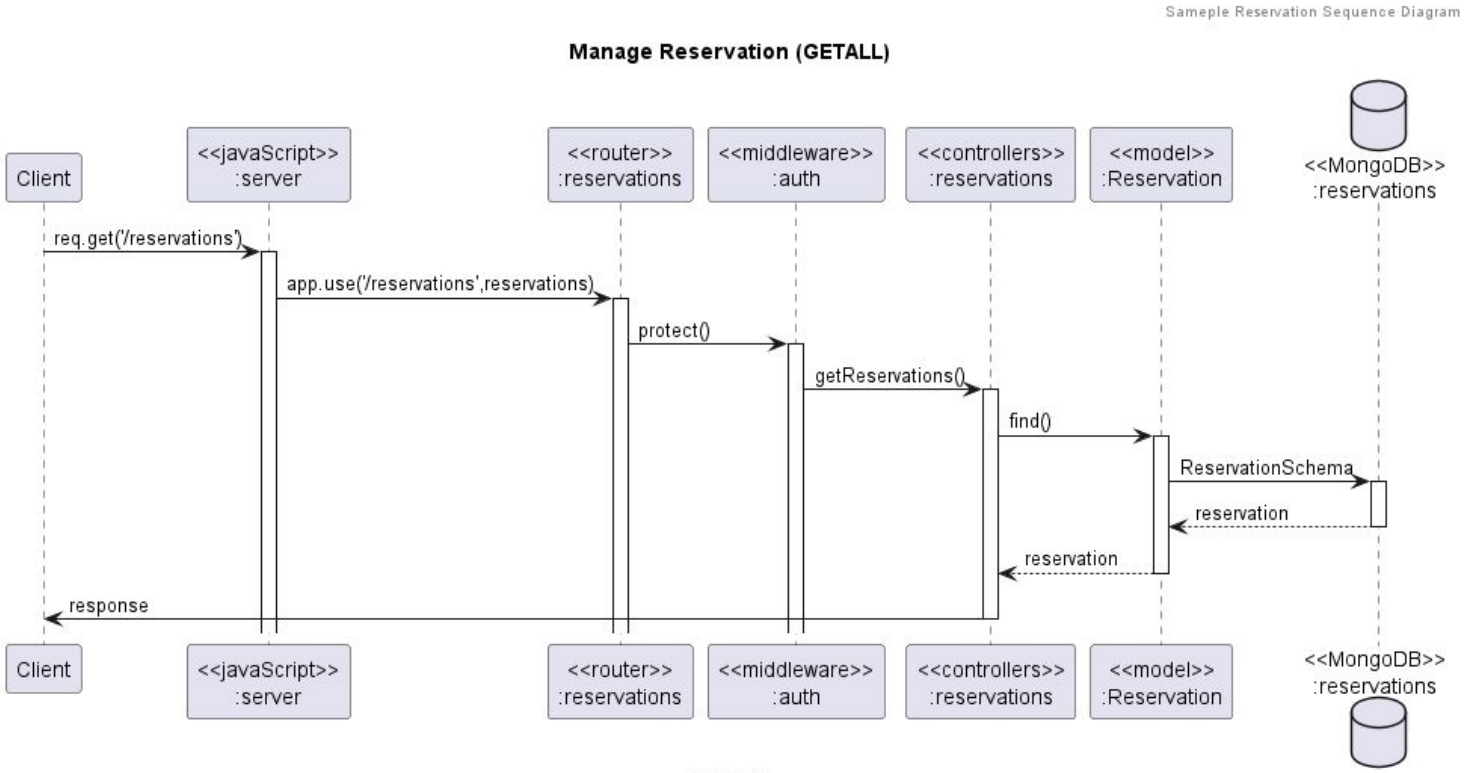
PUT Restaurant



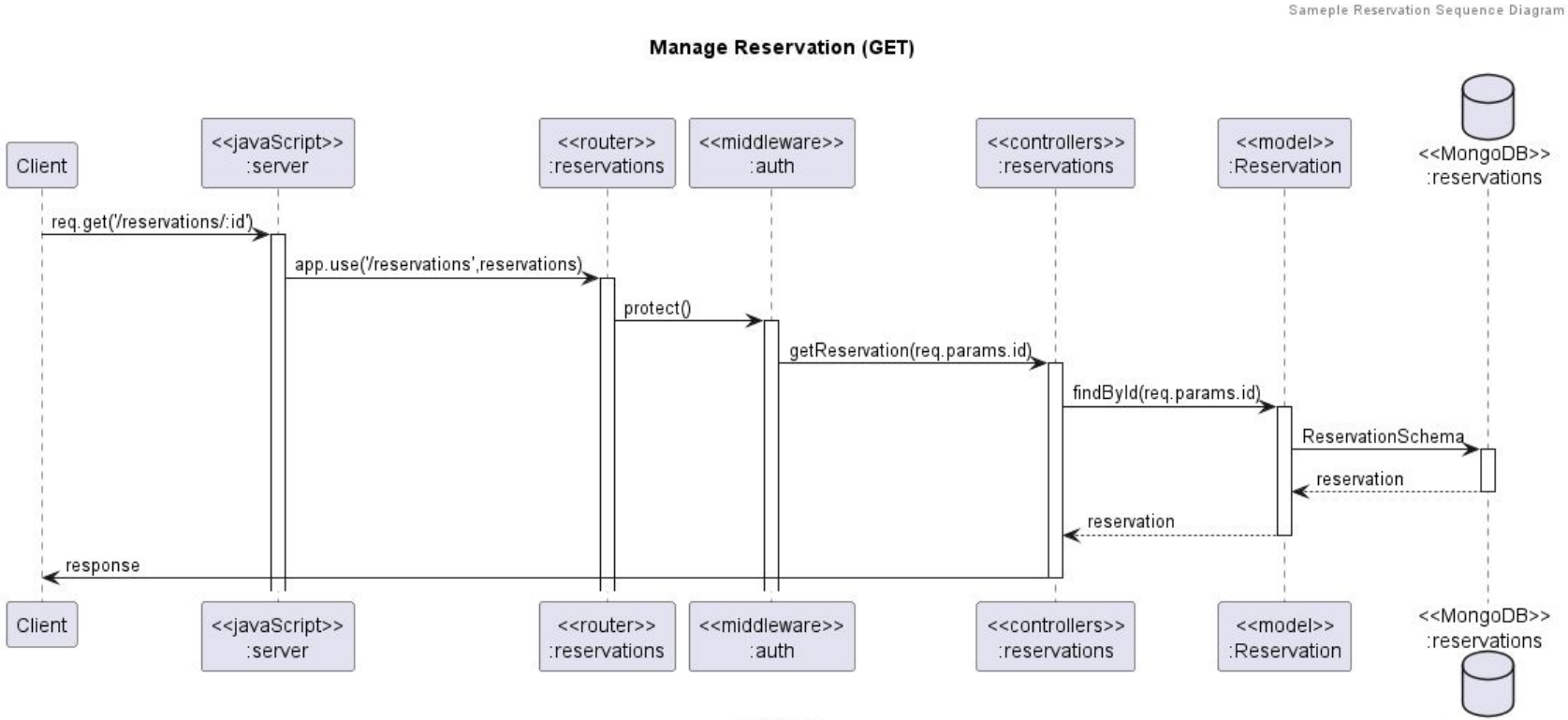
DELETE Restaurant



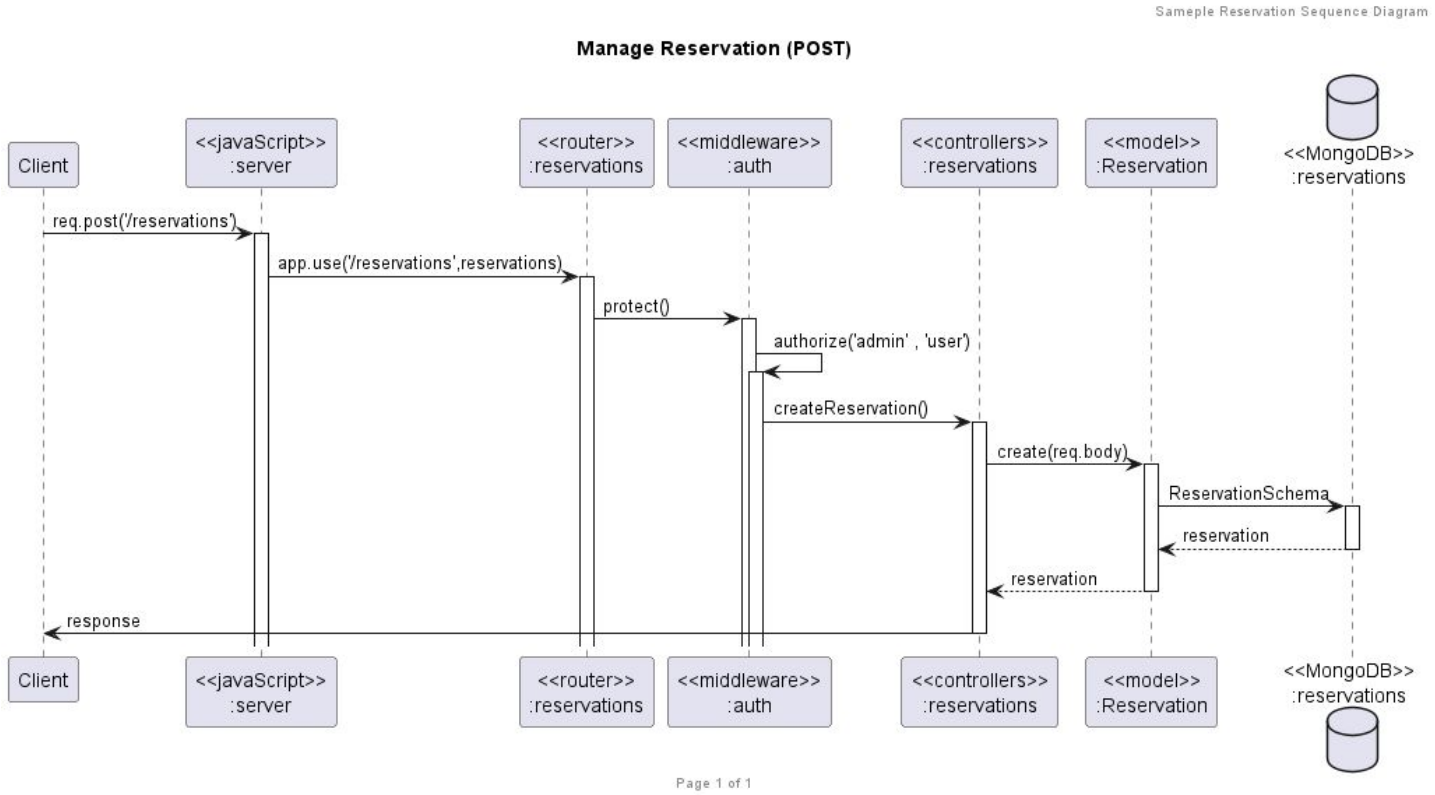
GET All Reservation



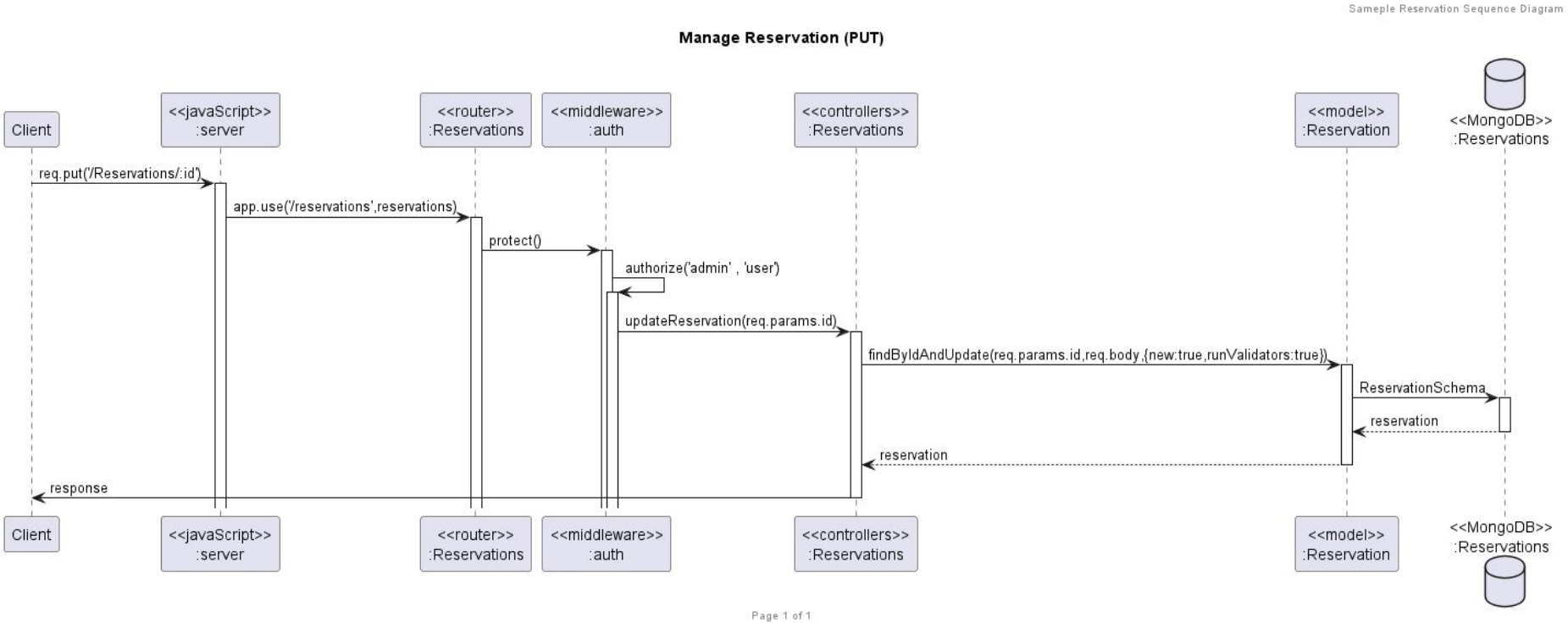
GET Reservation



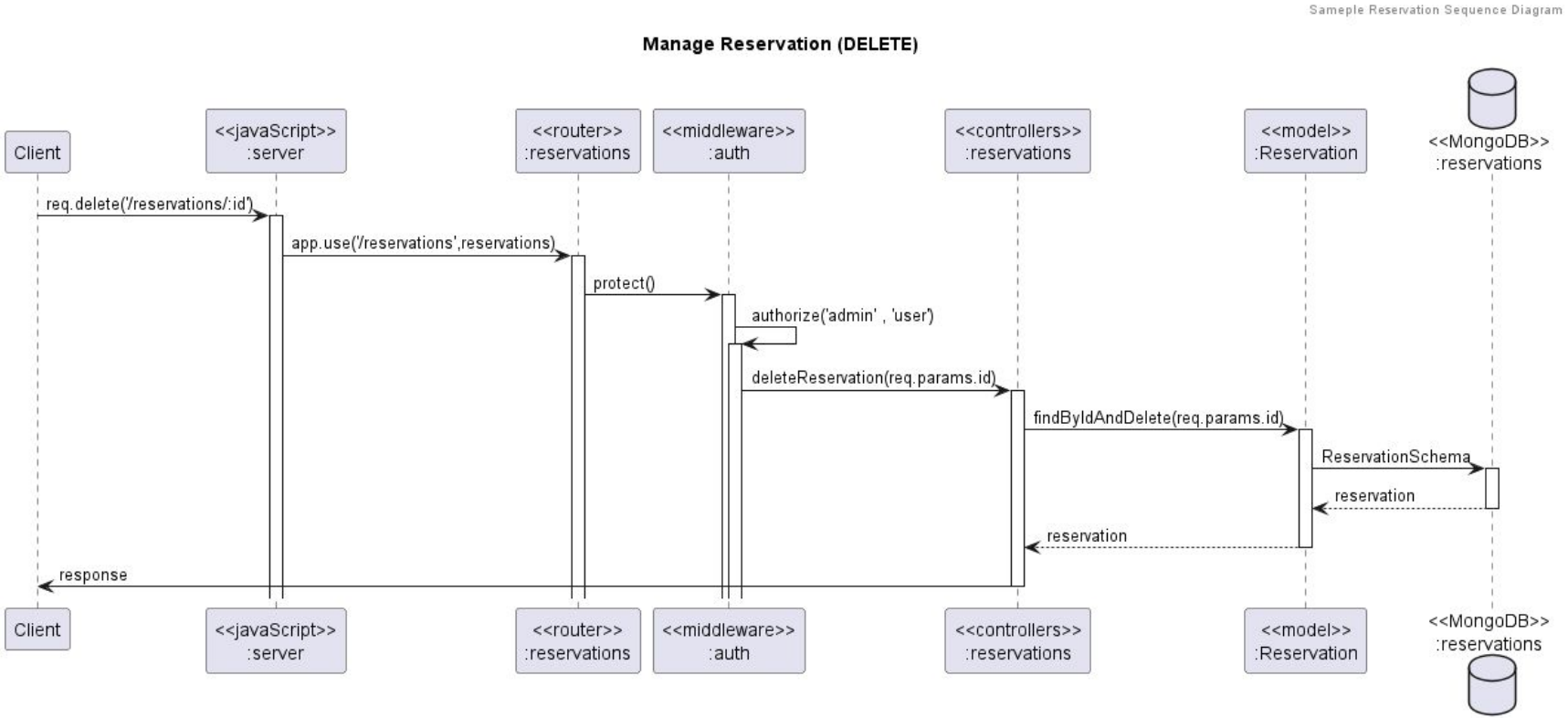
POST Reservation



PUT Reservation



DELETE Reservation



Extra

server.js

```
const restaurants = require('./routes/restaurants');

const reservations = require('./routes/reservations');

const reviews = require('./routes/reviews');

const auth = require('./routes/auth');
```

```
app.use('/api/v1/restaurants', restaurants);
app.use('/api/v1/reservations', reservations);
app.use('/api/v1/reviews', reviews);
app.use('/api/v1/auth', auth);
```

models/Review.js

```
const mongoose=require('mongoose');

const ReviewSchema = new mongoose.Schema({
    rating: {
        type: Number,
        required: true,
        min: [0,"Value must in between 0 - 5"],
        max: [5,"Value must in between 0 - 5"]
    },
    description: {
        type: String,
        maxlength: [255,'Description cannot be morethan 255 characters']
    },
    user: {
        type: mongoose.Schema.ObjectId,
        ref: 'User',
        required: true
    },
    restaurant:{
        type: mongoose.Schema.ObjectId,
        ref: 'Restaurant',
        required: true
    },
    createdAt:{
        type:Date,
        default: Date.now
    }
});

module.exports = mongoose.model('Review', ReviewSchema);
```

routes/reviews.js

```
const express = require('express');

const {getReviews, getReview, createReview, updateReview, deleteReview } = require('../controllers/reviews');

const router = express.Router({mergeParams: true});

const {protect, authorize} = require('../middleware/auth');

router.route('/').get(protect, getReviews).post(protect,authorize('admin','user'), createReview);

router.route('/:id').get(protect, getReview).put(protect,authorize('admin','user'), updateReview).delete(protect,authorize('admi

module.exports = router;
```

models\Restaurant

```
RestaurantSchema.pre('deleteOne', {document: true, query: false}, async function(next){
    console.log(`Reservations and Reviews being removed from restaurant ${this._id}`);

    await this.model('Reservation').deleteMany({restaurant: this._id});

    await this.model('Review').deleteMany({restaurant: this._id});

    next();

});
```

routes/restaurants.js

```
const reservationRouter = require('./reservations');

const reviewRouter = require('./reviews');

const router = express.Router();

const {protect, authorize} = require('../middleware/auth');

router.use('/:restaurantId/reviews/', reviewRouter);
router.use('/:restaurantId/reservations/', reservationRouter);
```

controllers/restaurants.js

```
const Review = require('../models/Review');
```

```
try{
    const total = await Restaurant.countDocuments();
    query = query.skip(startIndex).limit(limit);

    const restaurants = await query;

    for (let i = 0; i < restaurants.length; i++) {
        const avgRating = await Review.aggregate([
            {
                $match: { restaurant: restaurants[i]._id }
            },
            {
                $group: {
                    _id: null,
                    averageRating: { $avg: "$rating" }
                }
            }
        ]);

        restaurants[i] = { ...restaurants[i]._doc, reservations: restaurants[i].reservations,
            averageRating: avgRating.length > 0 ? avgRating[0].averageRating.toFixed(1) : 'No Review' };
    }

    const pagination = {};
```

controllers/restaurants.js (GET ONE)

```
exports.getRestaurant=async(req,res,next)=>{
    try{
        const restaurant = await Restaurant.findById(req.params.id);

        if(!restaurant){
            return res.status(400).json({success:false});
        }

        const avgRating = await Review.aggregate([
            {
                $match: { restaurant: restaurant._id }
            },
            {
                $group: {
                    _id: null,
                    averageRating: { $avg: "$rating" }
                }
            }
        ]);

        res.status(200).json({success:true, data: restaurant, averageRating: avgRating.length > 0 ? avgRating[0].averageRating.toFixed(1) : 'No Review'});
    }catch(err){
        res.status(400).json({success:false});
    }
};
```

controllers/reviews.js (GET ALL Reviews)

```
const Review = require('../models/Review');

const Restaurant = require('../models/Restaurant')

exports.getReviews = async (req, res, next) => {
    let query;

    if(req.params.restaurantId) {
        console.log(req.params.restaurantId);

        query = Review.find({
            restaurant: req.params.restaurantId
        }).populate({
            path: 'restaurant',
            select: 'name'
        });
    }else{
        query = Review.find().populate({
            path: 'restaurant',
            select: 'name'
        });
    }
    try{
        const reviews = await query;
        res.status(200).json({
            success: true,
            count: reviews.length,
            data: reviews
        });
    }catch(err){
        console.log(err);
        return res.status(500).json({
            success: false,
            message: 'Cannot find Review'
        });
    }
};
```

controllers/reviews.js (GET ONE Review)

```
exports.getReview = async (req, res, next) => {
    try{
        const review = await Review.findById(req.params.id).populate({
            path: 'restaurant',
            select: 'name'
        });


        if(!review){
            return res.status(404).json({success:false, message: `No review with the id of ${req.params.id}`});
        }

        res.status(200).json({
            success: true,
            data: review
        });
    }catch (error){
        console.log(error);
        return res.status(500).json({success: false, message: 'Cannot find Review'});
    }
};
```

controllers/reviews.js (CREATE Review)

```
exports.createReview = async (req, res, next) => {
    try {
        req.body.restaurant = req.params.restaurantId;

        const restaurant = await Restaurant.findById(req.params.restaurantId);

        if(!restaurant){
            return res.status(404).json({success: false, message: `No restaurant with the id of ${req.params.restaurantID}`});
        }

        req.body.user = req.user.id;

        const existingReview = await Review.find({ user: req.body.user, restaurant: req.params.restaurantId });

        if (existingReview.length >= 1 && req.user.role !== 'admin') {
            return res.status(400).json({ success: false, message: `The user with ID ${req.user.id} has already made a review for this restaurant` });
        }

        const review = await Review.create(req.body);

        res.status(200).json({
            success:true,
            data: review
        });
    }catch(err){
        console.log(err);
        if(req.body.rating < 0 || req.body.rating > 5){
            return res.status(400).json({ success: false, message: `rating value can only between 0-5`});
        }
        return res.status(500).json({success: false, message: 'Cannot create Review'});
    }
};
```

controllers/reviews.js (UPDATE Review)

```
exports.updateReview = async (req, res, next) => {
    try{
        let review = await Review.findById(req.params.id);

        if(!review){
            return res.status(404).json({
                success: false, message: `No review with the id of ${req.params.id}`
            });
        }

        if(review.user.toString() !== req.user.id && req.user.role !== 'admin'){
            return res.status(401).json({success: false, message: `User ${req.user.id} is not authorized to update this review`});
        }

        review = await Review.findByIdAndUpdate(req.params.id, req.body,{
            new:true,
            runValidators: true
        });
        res.status(200).json({
            success: true,
            data: review
        });
    }catch (error){
        console.log(error);
        return res.status(500).json({success: false, message: 'Cannot update Review'});
    }
};
```

controllers/reviews.js (DELETE Review)

```
exports.deleteReview = async (req, res, next) => {
    try {
        const review = await Review.findById(req.params.id);
        if (!review) {
            return res.status(404).json({ success: false, message: `No review with the id of ${req.params.id}` });
        }

        if(review.user.toString() !== req.user.id && req.user.role !== 'admin'){
            return res.status(401).json({success: false, message: `User ${req.user.id} is not authorized to delete this review` });
        }

        await review.deleteOne();

        res.status(200).json({ success: true, data: {} });
    } catch (err) {
        res.status(500).json({ success: false, msg: 'Cannot delete Review' });
    }
};
```

Member

1.	Tanathorn	Piyachart	6633097121
2.	Suchaya	Limprasitissara	6633265221
3.	Aiyapat	Vorasaran	6633288721

GitHub Link

https://github.com/2110503TACEDT66/presentation-day-1-mylittlepony/graphs/contributors