DentService

6432022721 Jirayut Phothawilkiat

GitHub Link: https://github.com/JirayutP/Dentist

Project#4: Dentist Booking

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 logout.
3. After login, the system shall allow the registered user to book only ONE session by specifying the date and the preferred dentist. The dentist list is also provided to the user. A dentist information includes the dentist's name, years of experience, and area of expertise.
4. The system shall allow the registered user to view his booking.
5. The system shall allow the registered user to edit his booking.
6. The system shall allow the registered user to delete his booking.
7. The system shall allow the admin to view any bookings.
8. The system shall allow the admin to edit any bookings.
9. The system shall allow the admin to delete any bookings

models/Booking.js

```
const BookingSchema=new mongoose.Schema({
    bookingDate: {
        type: Date,
        required: [true, 'Please add a booking date']
    },
    user:{
        type:mongoose.Schema.ObjectId,
        ref: 'User',
        required: true
    },
    dentist:{
        type:mongoose.Schema.ObjectId,
        ref: 'Dentist',
        required: true
    },
    createdAt:{
        type: Date,
        default: Date.now
    }
});
```

models/Dentist.js

```
const DentistsSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true, 'Please add a name'],
        uniqe: true,
        trim: true,
        maxlenght: [50, 'Name can not be more than 50 characters']
    },
    yearsOfExperience: {
      type: Number,
      required: [true, 'Please add years of experience']
    },
    areaOfExpertise: {
      type: String,
      required: [true, 'Please add specify area of expertise']
    }
});

//Cascade delete booking when a dentist is deleted
DentistsSchema.pre('deleteOne', {document:true, query:false}, async function(next){
    console.log(`Bookings being removed from dentist ${this._id}`);
    await this.model('Booking').deleteMany({dentist: this._id});
    next();
});
```

models/User.js

```
const UserSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true, 'Please add a name']
    },
    telephone: {
        type: String,
        required: [true, 'Please add a telephone number']
      },
    email: {
        type: String,
        required: [true, 'Please add an email'],
        uniqe: 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
    },
```

controller/auth.js

```
//@dest     Register user
//@route    Post /api/v1/auth/register
//@access   Public
exports.register = async (req,res,next) => {
    try{
        const {name, telephone, email, password, role} = req.body;

        //Create user
        const user = await User.create({
            name,
            telephone,
            email,
            password,
            role
        });
        //Create token
        // const token = user.getSignedJwtToken();
        // res.status(200).json({success:true,token});
        sendTokenResponse(user,200,res);
    }catch(err){
        res.status(400).json({success:false});
        console.log(err.stack);
    }
}
```

controller/bookings.js

```
exports.addBooking=async (req,res,next)=>{
    try{
        req.body.dentist=req.params.dentistId;
        const dentist=await Dentist.findById(req.params.dentistId);

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

        //add user Id to req.body
        req.body.user=req.user.id;

        //Check for existed appointment
        const existedBooking = await Booking.find({user:req.user.id});

        //if the user is not an admin, they can only create 1 appointment.
        if(existedBooking.length >= 1 && req.user.role !== 'admin'){
            return res.status(400).json({
                success:false,
                message: `The user with ID ${req.user.id} has alrady made 1 booking`
            });
        }

        const booking = await Booking.create(req.body);

        res.status(201).json({
            success:true,
            data:booking
        });
```

controller/dentist.js

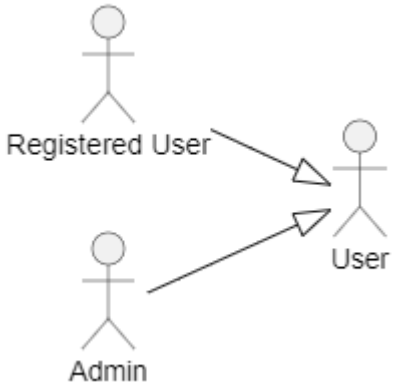
```
exports.getAreaOfExpertise = async (req, res, next) => {
  try {
    // Use distinct() method to get unique areaOfExpertise values
    const distinctExpertise = await Dentist.distinct('areaOfExpertise');
```

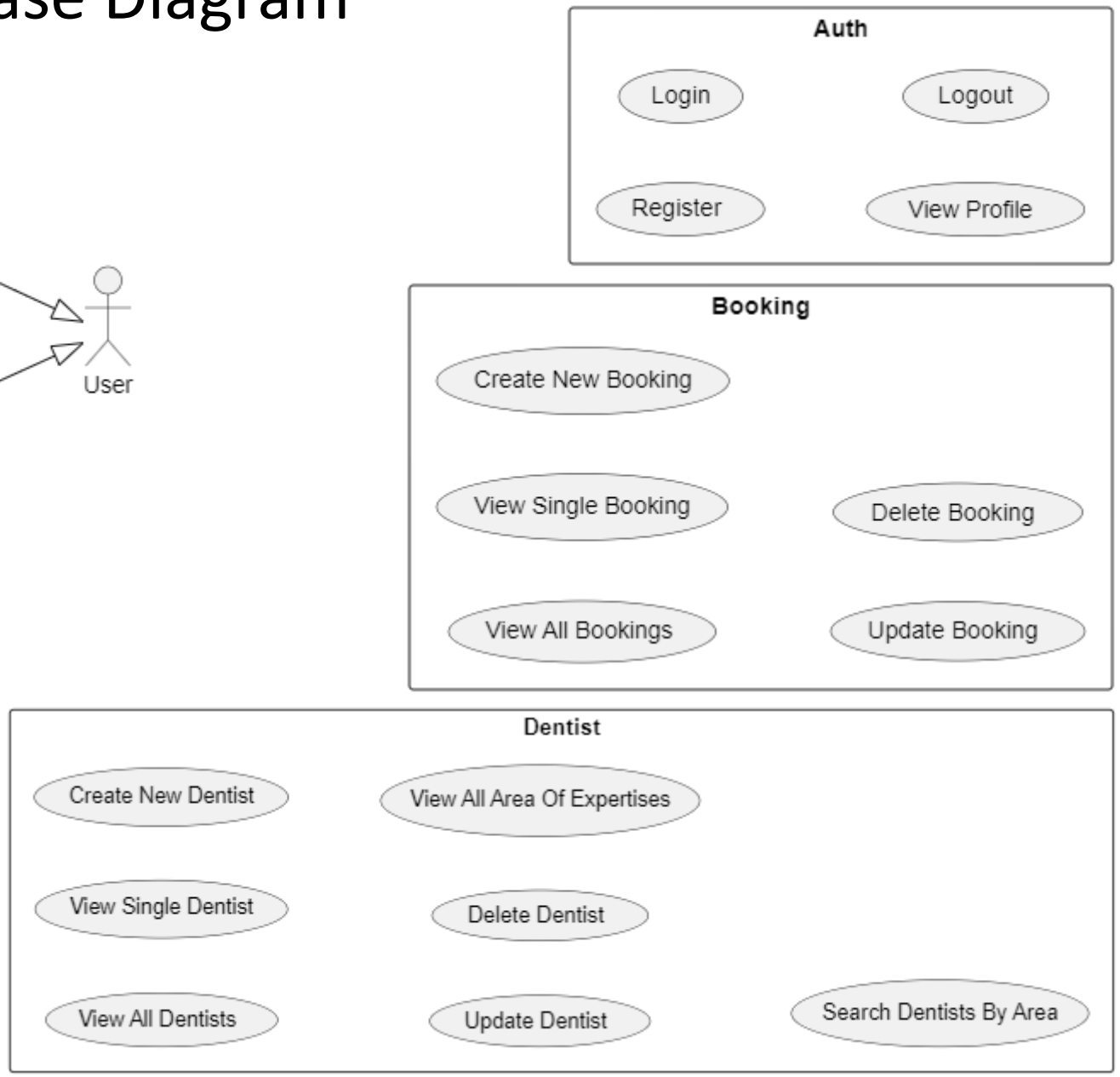
```
exports.getByArea = async (req, res, next) => {
  try {
    // Find dentists with matching areaOfExpertise using a query
    const dentists = await Dentist.find({ areaOfExpertise: req.params.area }); // Case-sensitive

    // Check if any dentists were found
    if (!dentists.length) {
      return res.status(404).json({ success: false, message: 'No dentists found for this area' });
    }
```

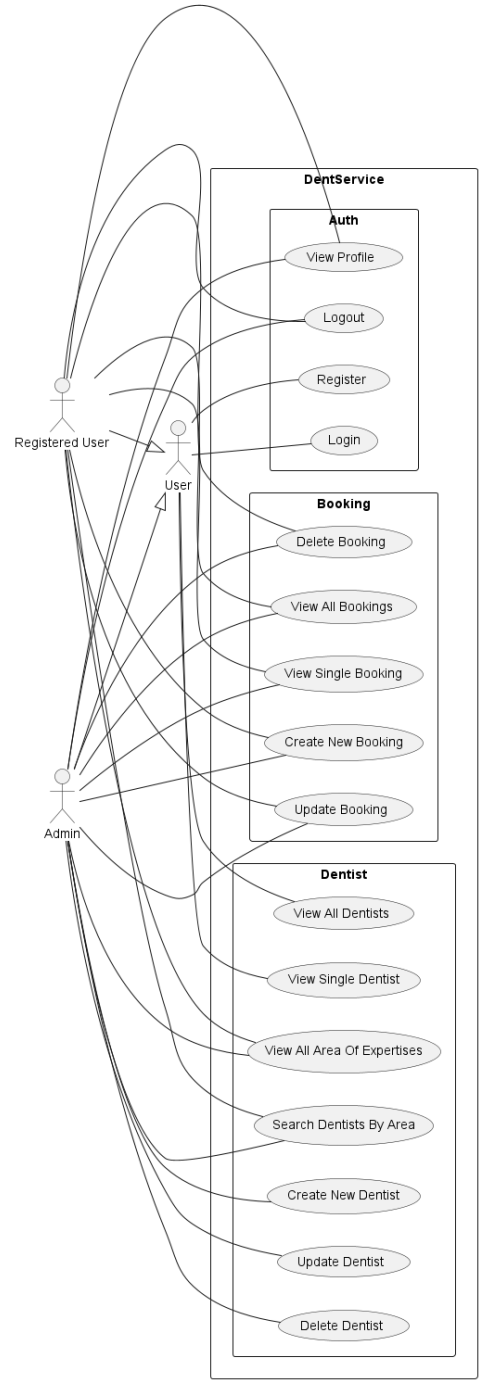
routes/dentist.js

```
//Include other resource routers
const bookingRouter = require('./bookings');

const router = express.Router();

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

//Re-route into other resource routers
router.use('/:dentistId/bookings/', bookingRouter);

router.route('/').get(getDentists).post(protect, authorize('admin'), createDentist);
router.route('/areaOfExpertise').get(protect, getAreaOfExpertise);
router.route('/byArea/:area').get(protect, getByArea);
router.route('/:id').get(getDentist).put(protect, authorize('admin'), updateDentist).delete(protect, authorize('admin'), deleteDentist);
```

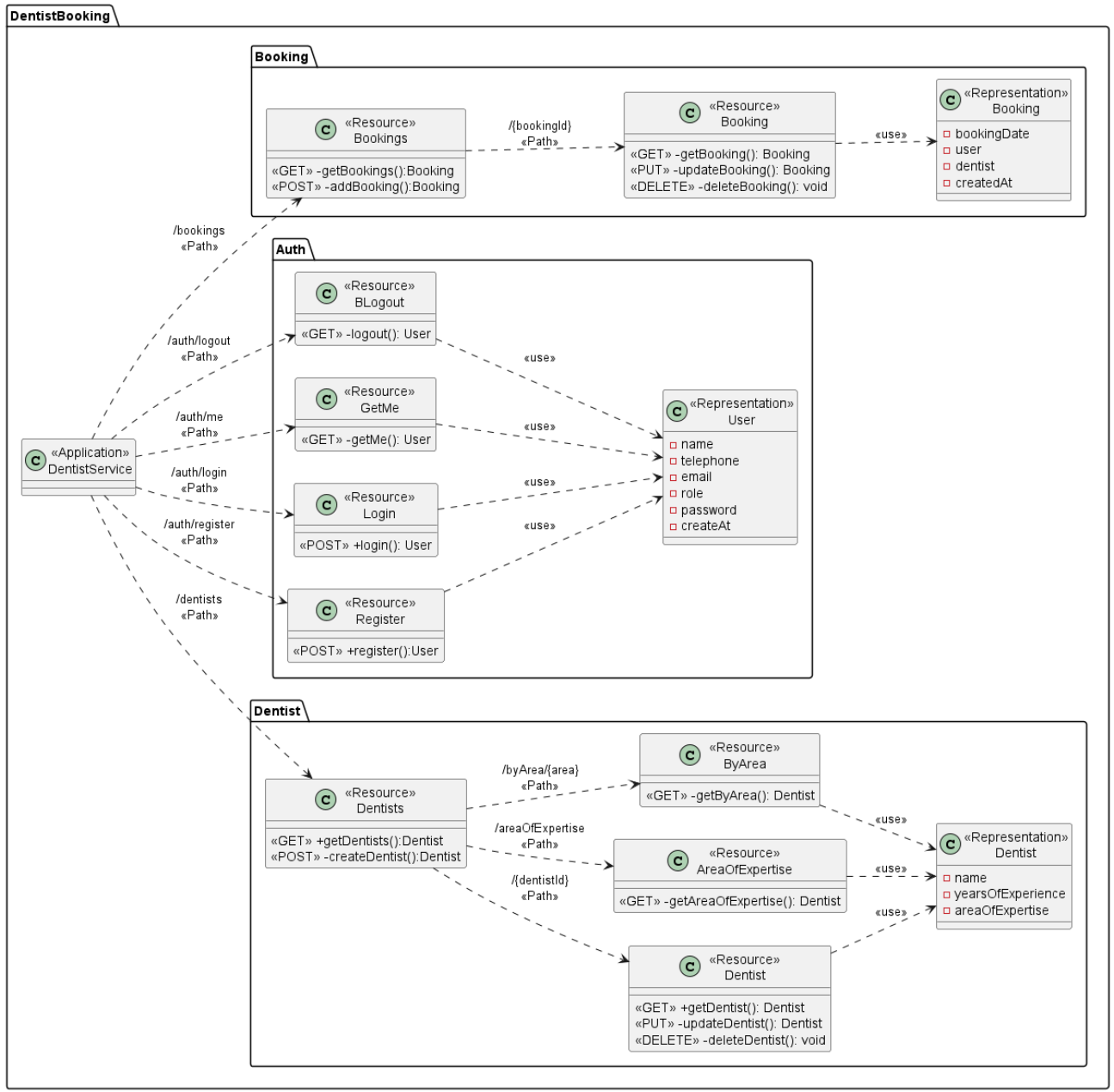
UML Use Case Diagram



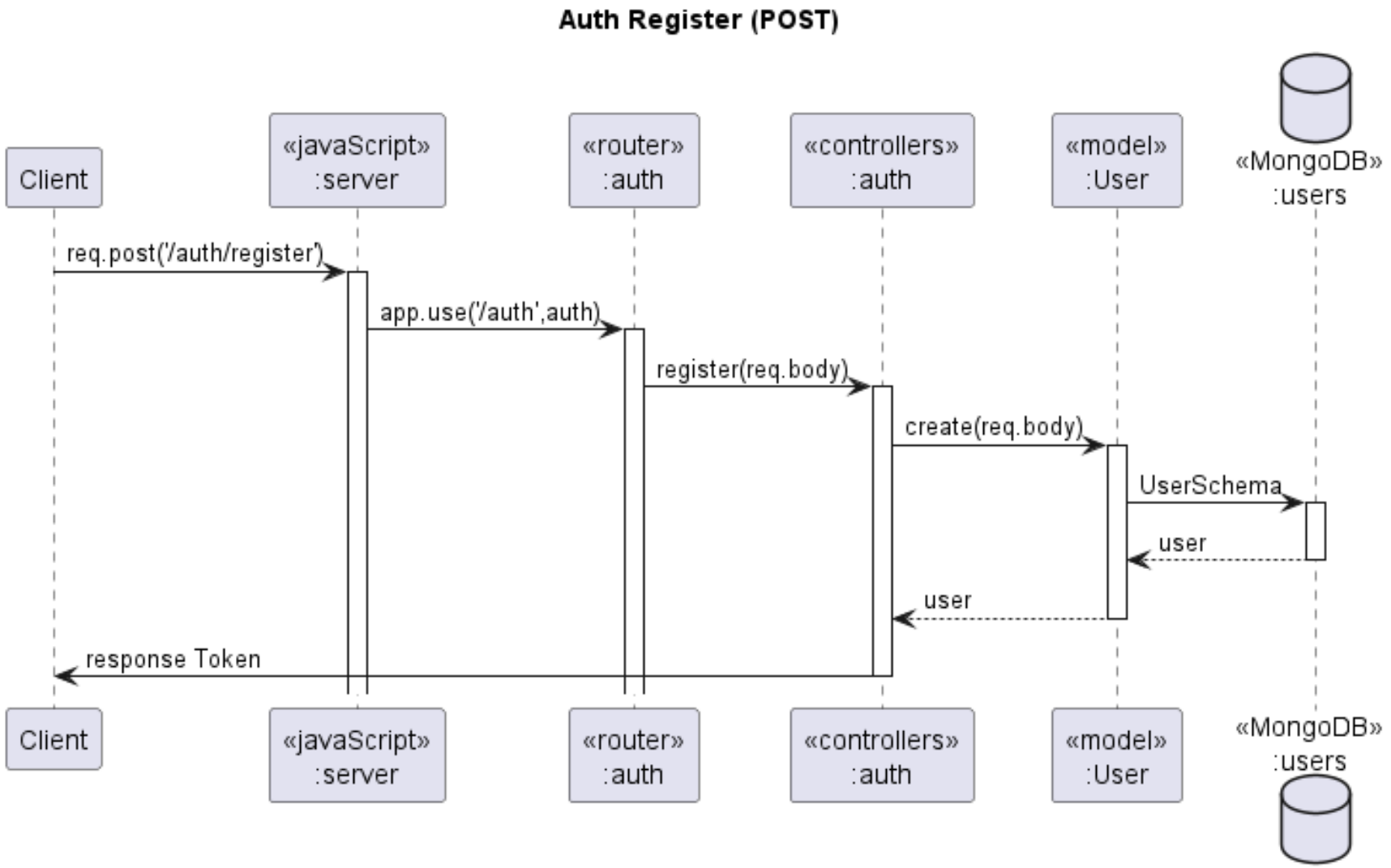




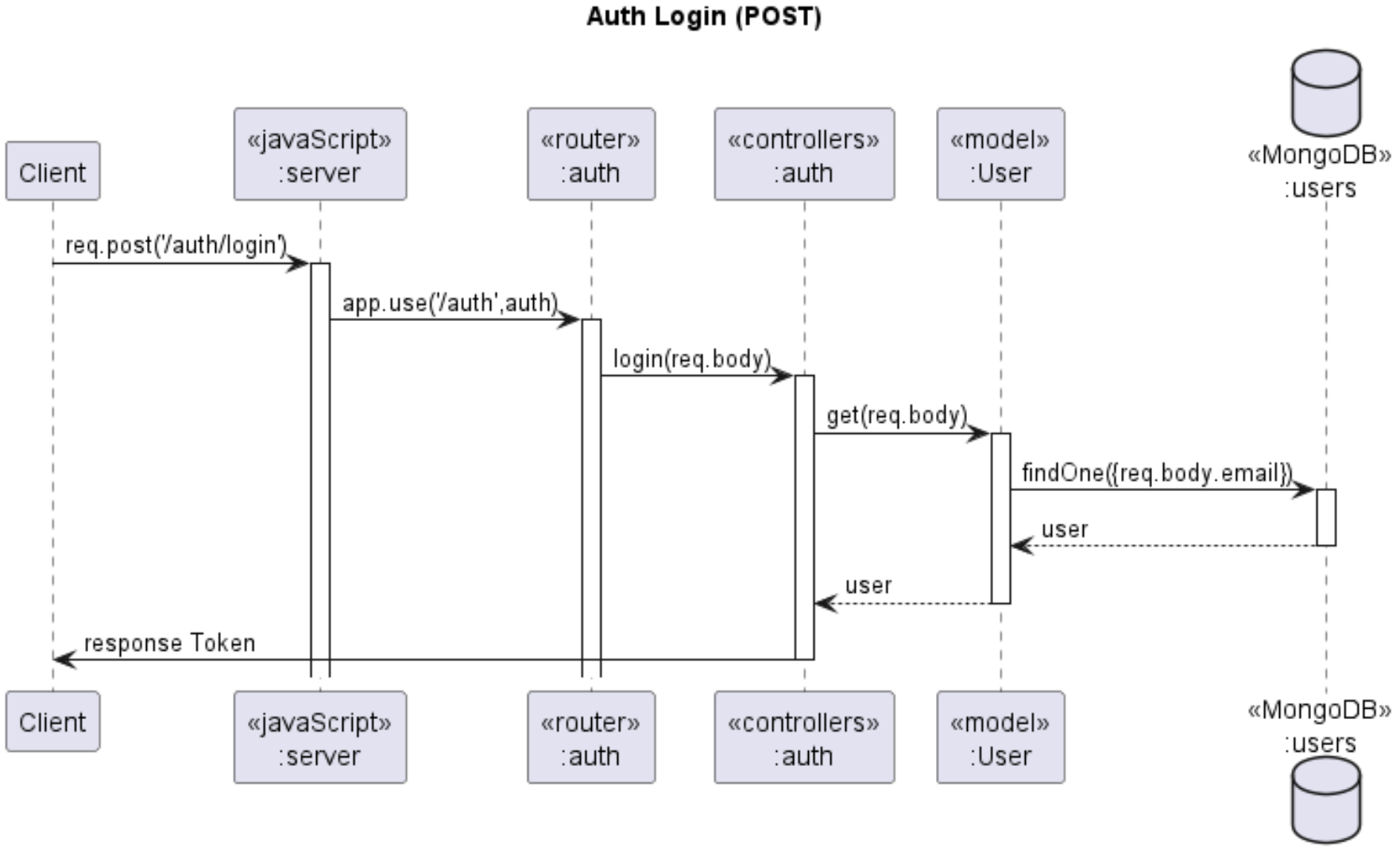
Class Diagram: UML Profile



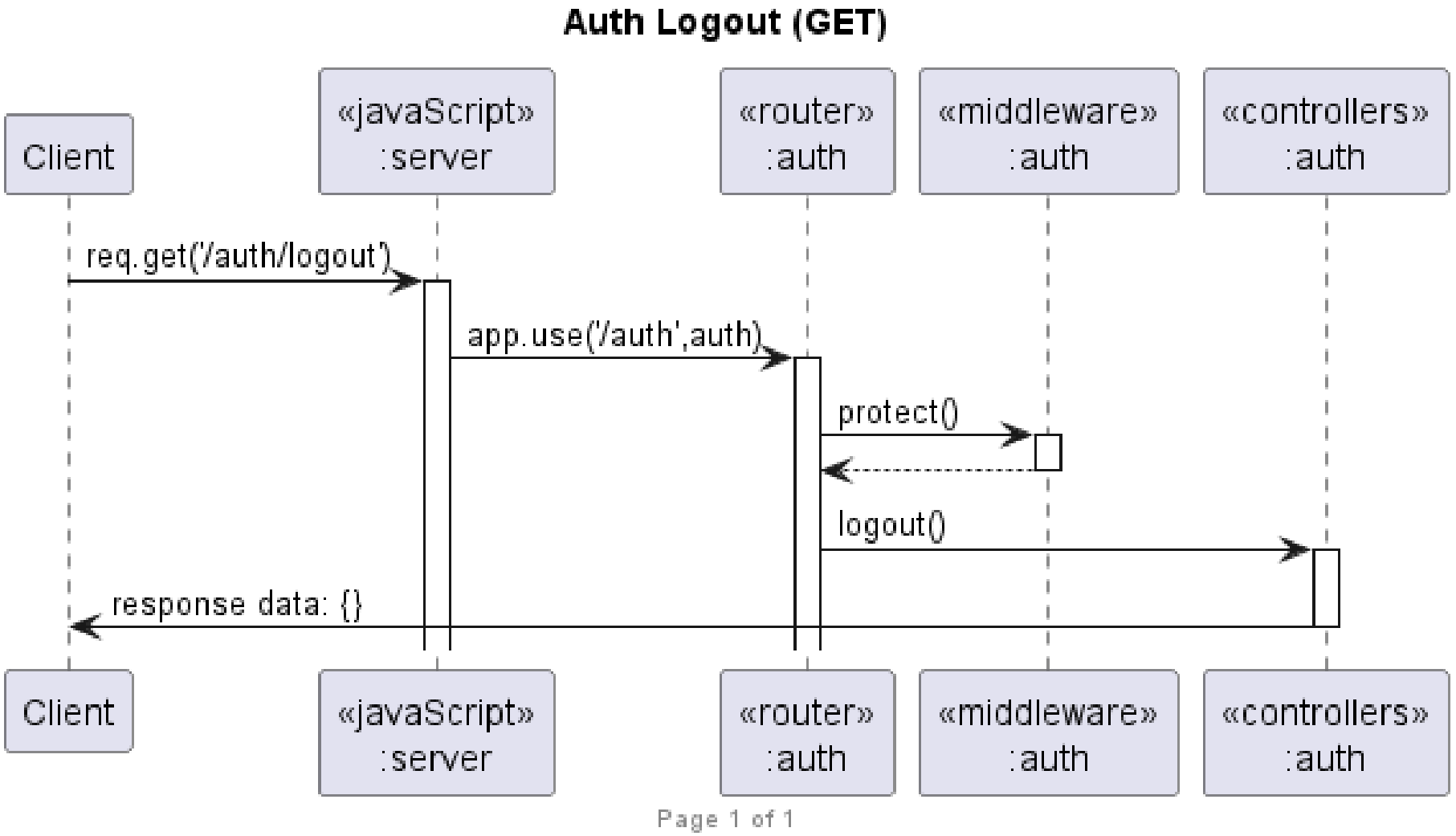
Sequence Diagram: Register (POST)



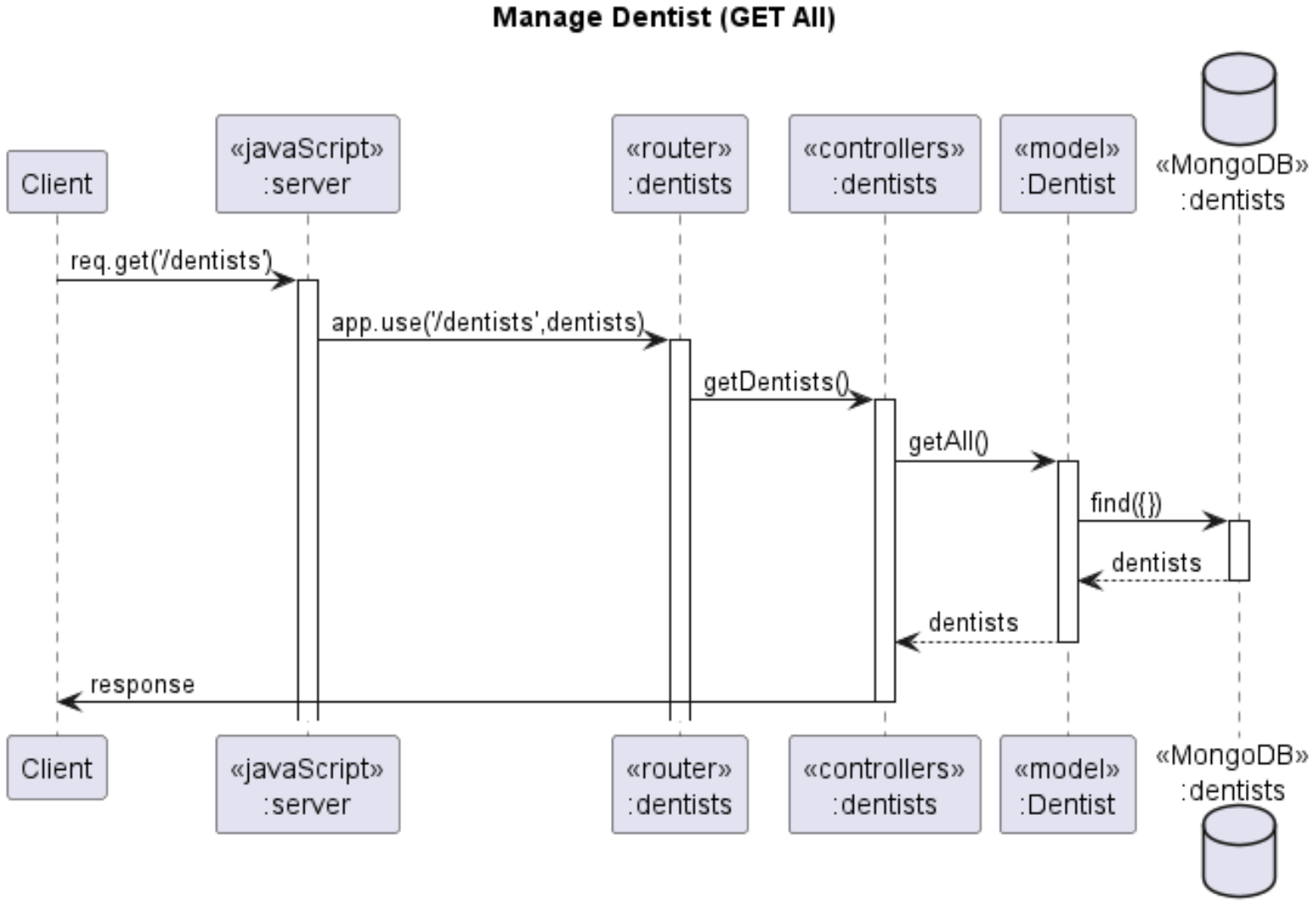
Sequence Diagram: Login (POST)



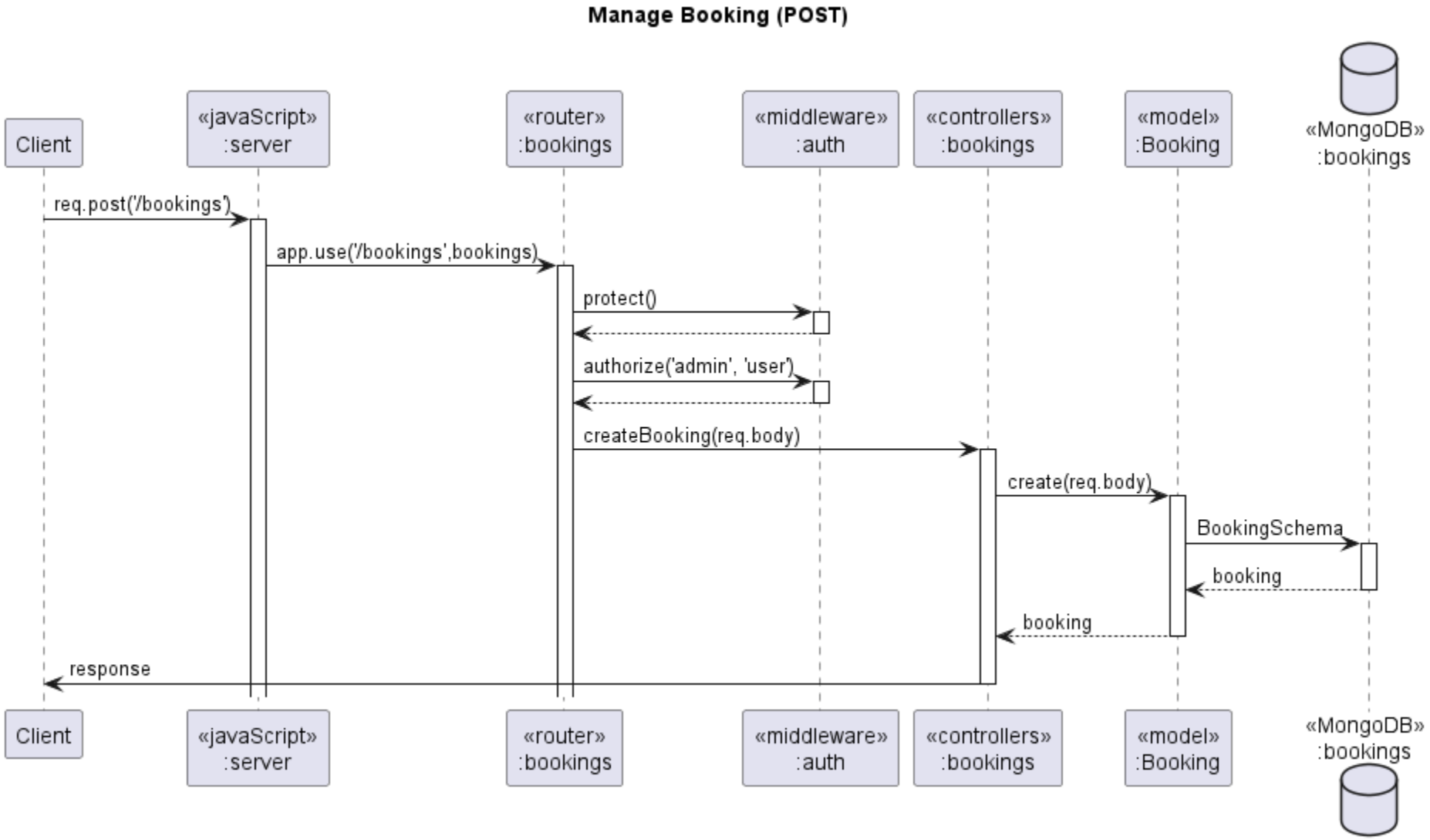
Sequence Diagram: Logout (GET)



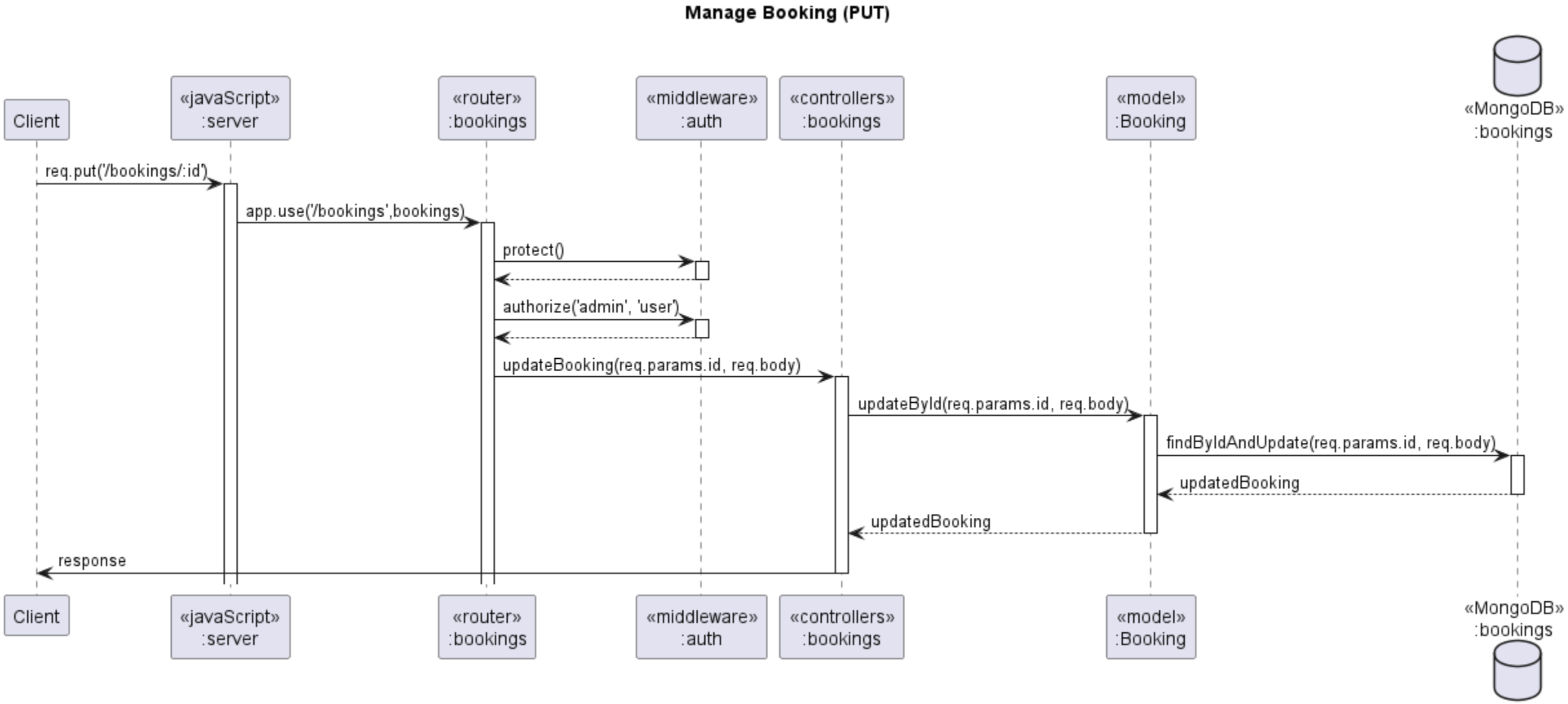
Sequence Diagram: Get All Dentist (GET)



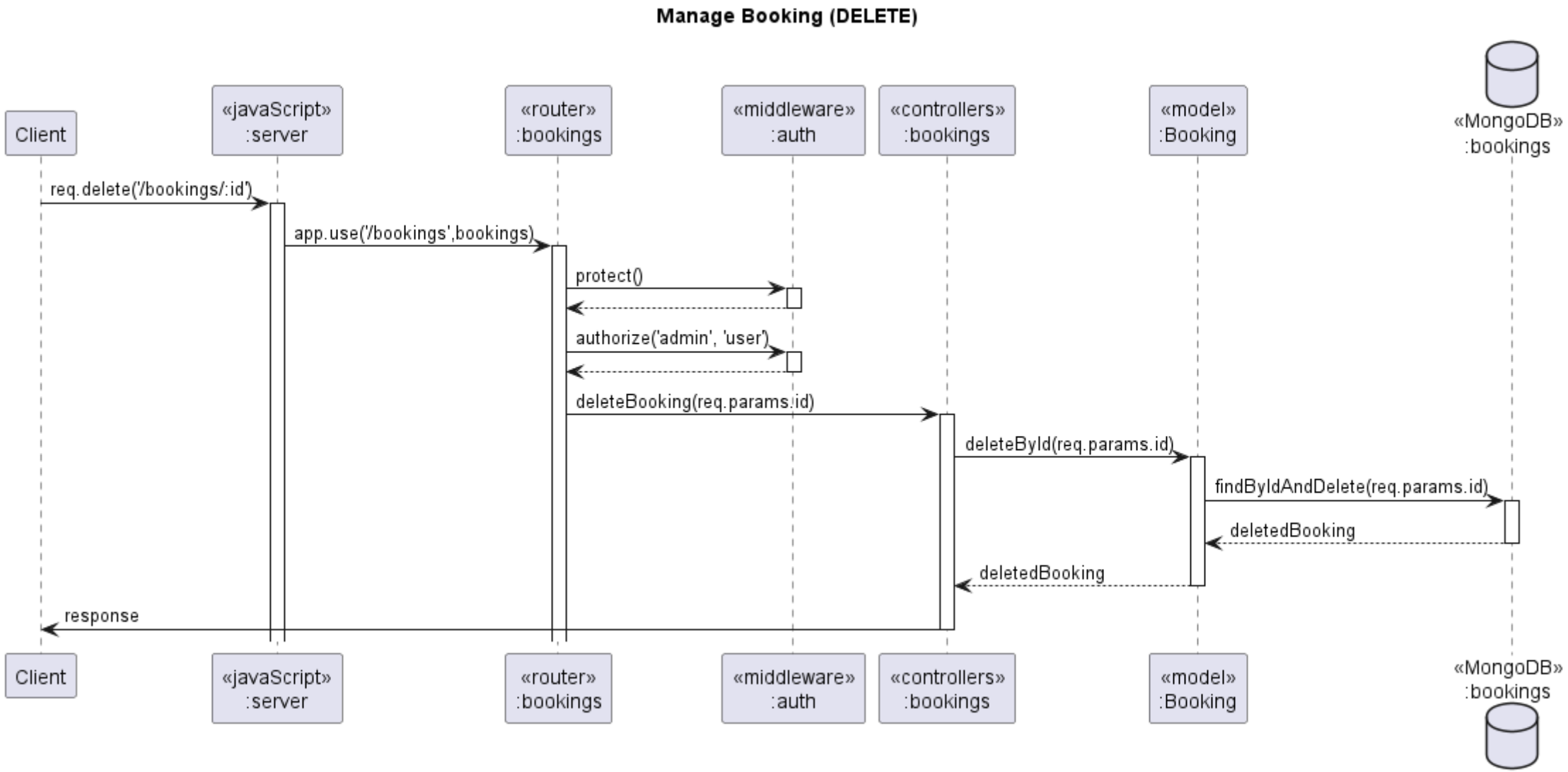
Sequence Diagram: Create Booking (POST)



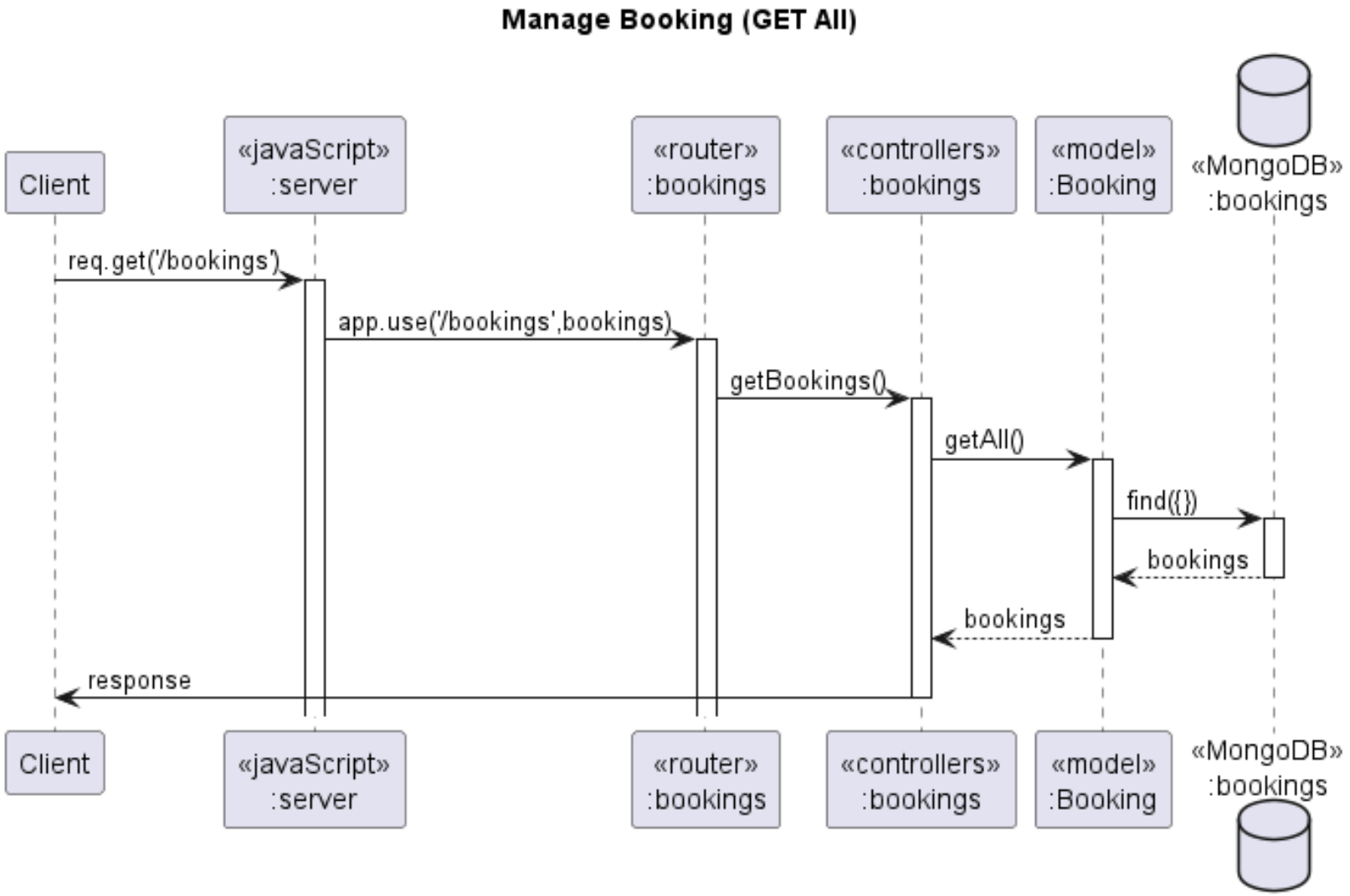
Sequence Diagram: Update Booking (PUT)



Sequence Diagram: Delete Booking (DELETE)



Sequence Diagram: Get All Booking (GET)



Additional Requirement

- The system shall allow the admin to create dentist information.
- The system shall allow the admin to delete any dentist information.
- The system shall allow the registered user to search dentist information by area of expertise. (getAreaOfExpertise and getByArea: The area used to search must match an existing area only to be found.)

GitHub Link: https://github.com/JirayutP/Dentist

Video Link: (Describe additional requirement)

https://chula-my.sharepoint.com/:v:/g/personal/6432022721_student_chula_ac_th/EV4lBv_M9wxIssOnYs14QoIBTVD-ZUKo-b-w3333oMlDig?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=CSVfYY