Trần Đăng Khoa
B2014926
M02
Repo: https://github.com/23-24Sem1-Courses/ct313hm02-contactbook-be-khoadangtran.git
CT313H: WEB TECHNOLOGIES AND SERVICES

Building Contactbook App - Backend - Part 2

You will build a contact management app as a SPA app. The tech stack includes *Nodejs/Express, Knex.js, MySQL/MariaDB* for backend (API server) and *Vue.js* for frontend (GUI). In the first two lab sessions, you will build the API server for the app.

The API server must support the following requests:

- *POST /api/contacts*: creates a new contact
- *GET /api/contacts*: returns all contacts from the database. This endpoint supports the following optional parameters:
  - *favorite* and *name* are for querying favorite contacts and contacts filtered by name. For example, *GET /api/contacts?favorite&name=duy* returns favorite contacts named "duy"
  - *page* and *limit* are for pagination
- *DELETE /api/contacts*: deletes all contacts in the database
- *GET /api/contacts/<contact-id>*: gets a contact with a specific ID
- *PUT /api/contacts/<contact-id>*: updates a contact with a specific ID
- *DELETE /api/contacts/<contact-id>*: deletes a contact with a specific ID
- All requests for undefined URLs will result in a 404 error with the message "Resource not found"

A contact has the following information: *name (string), email (string), address (string), phone (string), favorite (boolean)*. **Data format used for client-server communication is JSON**. The source code is managed by git and uploaded to GitHub.

This step-by-step guide will help implement all the above requirements. However, students are free to make their own implementation as long as the requirements are met.

Requirements for the lab report:

- The submitted report file is a PDF file containing images showing the results of your works (e.g., images showing the implemented functionalities, successful and failed scenarios, results of the operations, ...). **You should NOT screenshoot the source code**.
- **You only need to create ONE report for the whole four lab sessions**. At the end of each lab session, students need to (1) submit the work-in-progress

report and (2) push the code to the GitHub repository given by the instructor.

The report should also filled with student information (student ID, student name, class ID) and the links to the GitHub repositories.

Plagiarism will result in 0.

(Continue from the result of Part 1)

Implement route handlers

Define a module that creates a knex object representing the connection to the database in *src/database/knex.js*:

```
const { DB_HOST, DB_PORT, DB_USER, DB_PASS, DB_NAME } = process.env;

module.exports = require('knex')({
    client: 'mysql',
    connection: {
        host: DB_HOST,
        port: DB_PORT,
        user: DB_USER,
        password: DB_PASS,
        database: DB_NAME,
    },
    pool: { min: 0, max: 10 },
});
```

Create *src/services/contacts.service.js* file to define a set of functions for accessing the database:

```
const knex = require('../database/knex');

function makeContactsService() {
    //Define functions for accessing the database

    return {

    };
}

module.exports = createContactsService;
```

Implement createContact handler

Edit *src/controllers/contacts.controller.js*:

```
const makeContactsService = require('../services/contacts.service');
const ApiError = require('../api-error');

// Create and Save a new Contact
async function createContact(req, res, next) {
    if (!req.body?.name) {
        return next(new ApiError(400, 'Name can not be empty'));
    }

    try {
        const contactsService = makeContactsService();
        const contact = await contactsService.createContact(req.body);
        return res.send(contact);
    } catch (error) {
        console.log(error);
        return next(
            new ApiError(500, 'An error occurred while creating the contact')
        );
    }
}
```

In case of error, the call *next(error)* will transfer the execution to the error handling middleware defined in *src/app.js* will be called.

contactsService.createContact() stores the submitted contact to the database. The function *createContact()* is defined (in *src/services/contacts.service.js*) as follows:

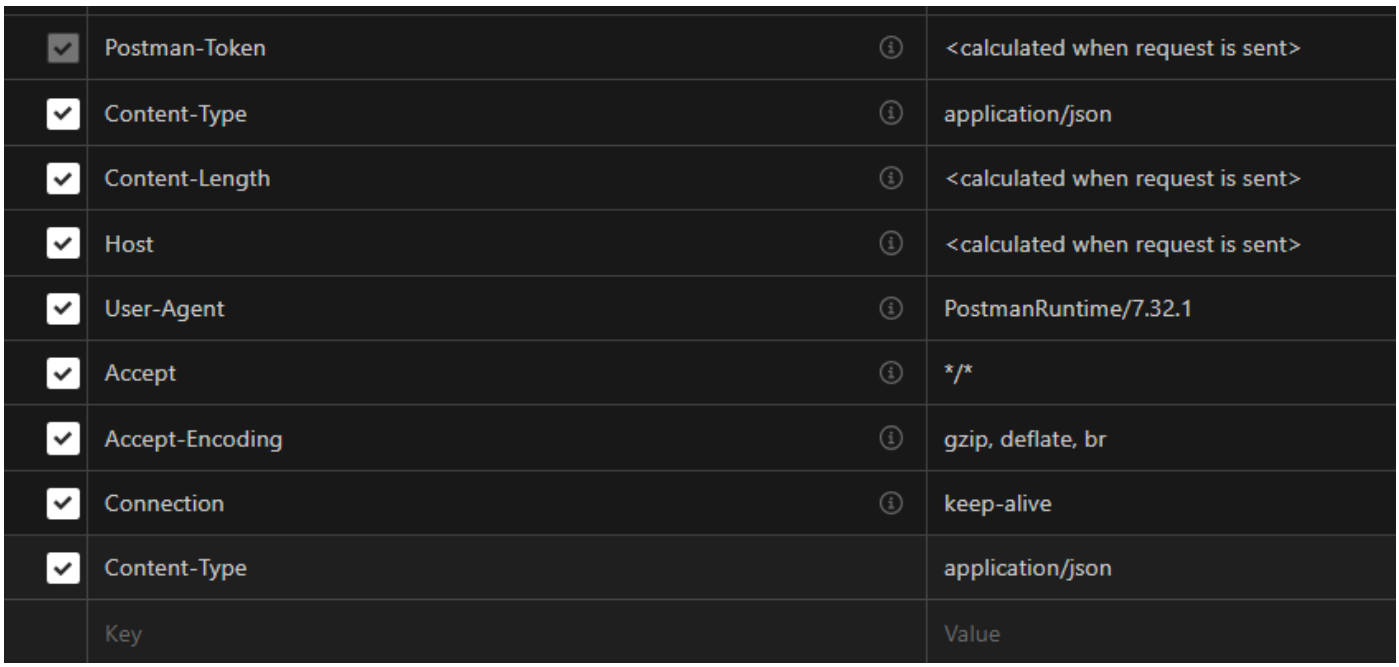
```
const knex = require('../database/knex');

function makeContactsService() {
    function readContact(payload) {
        const contact = {
            name: payload.name,
            email: payload.email,
            address: payload.address,
            phone: payload.phone,
            favorite: payload.favorite,
        };
        // Remove undefined fields
        Object.keys(contact).forEach(
            (key) => contact[key] === undefined && delete contact[key]
        );
        return contact;
    }

    async function createContact(payload) {
        const contact = readContact(payload);
            const [id] = await knex('contacts').insert(contact);
        return { id, ...contact };
    }

    return {
```

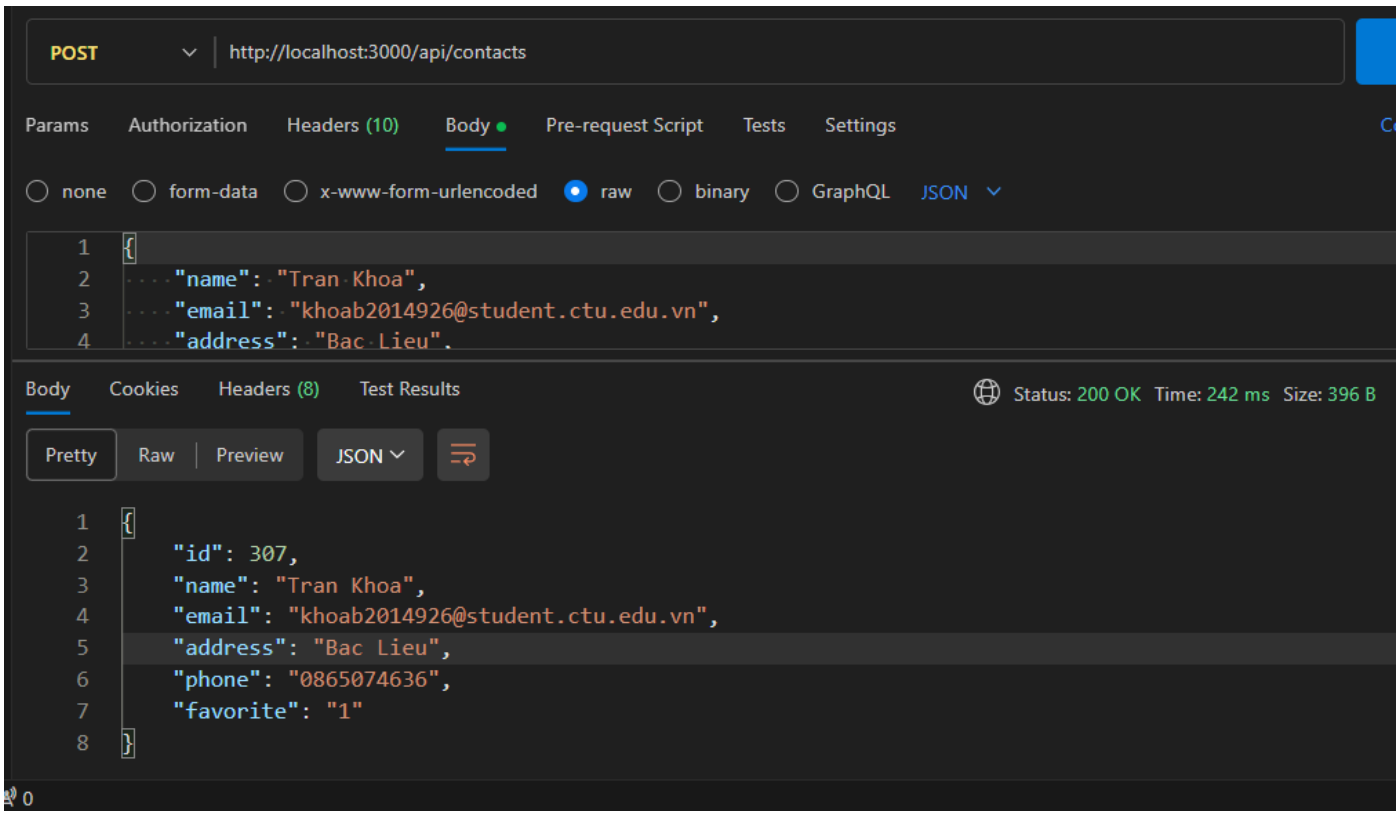
```
        createContact,
    };
}
module.exports = createContactsService;
```

Use a HTTP client to verify the handler works as expected.

In order to send JSON data to the server with a HTTP client, make sure to set the header "Content-Type: application/json" and put JSON data in the request body, for example:

Put JSON data inside Body





Implement getContactsByFilter handler

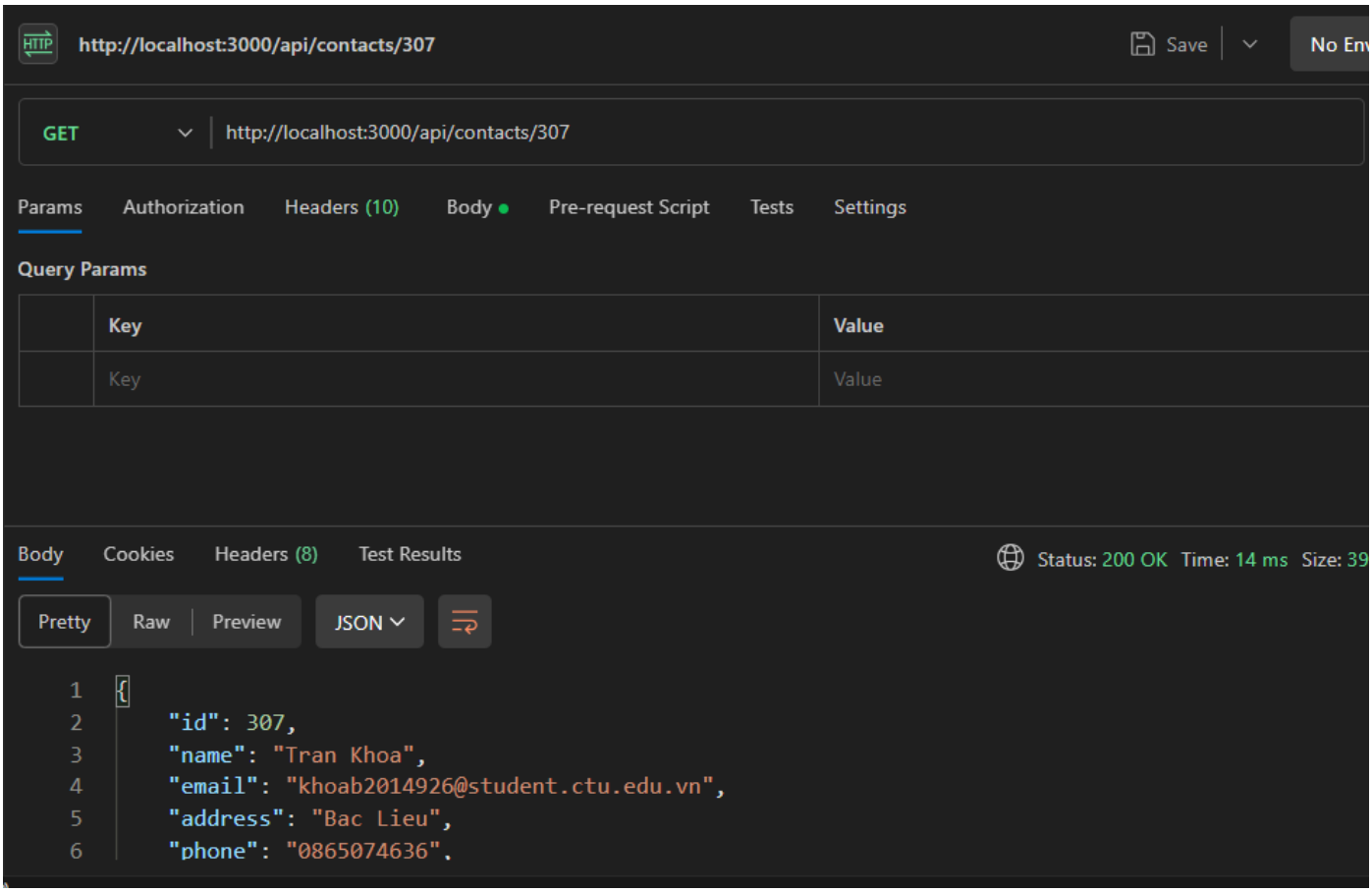
Edit *src/controllers/contacts.controller.js*:

```
// Retrieve contacts of a user from the database
async function getContactsByFilter(req, res, next) {
    let contacts = [];

    try {
        const contactsService = makeContactsService();
        contacts = await contactsService.getManyContacts(req.query);
    } catch (error) {
        console.log(error);
        return next(
            new ApiError(500, 'An error occurred while retrieving contacts')
        );
    }

    return res.send(contacts);
}
```

contactsService.getManyContacts(query) returns contacts filtered the *query* (name and favorite). This function can be defined as follows:

```
...
function makeContactsService()
    { ...
    async function getManyContacts(query) {
        const { name, favorite } = query;
        return knex('contacts')
            .where((builder) => {
                if (name) {
                    builder.where('name', 'like', `%${name}%`);
                }
                if (favorite !== undefined) {
                        builder.where('favorite', 1);
                }
            })
            .select('*');
    }

    return {
        createContact,
        getManyContacts,
    };
}
...
```

Use a HTTP client to verify the handler works as expected.



Paginate records for *getManyContacts(query)*:

Define a class named *Paginator* (in *src/services/paginator.js*):

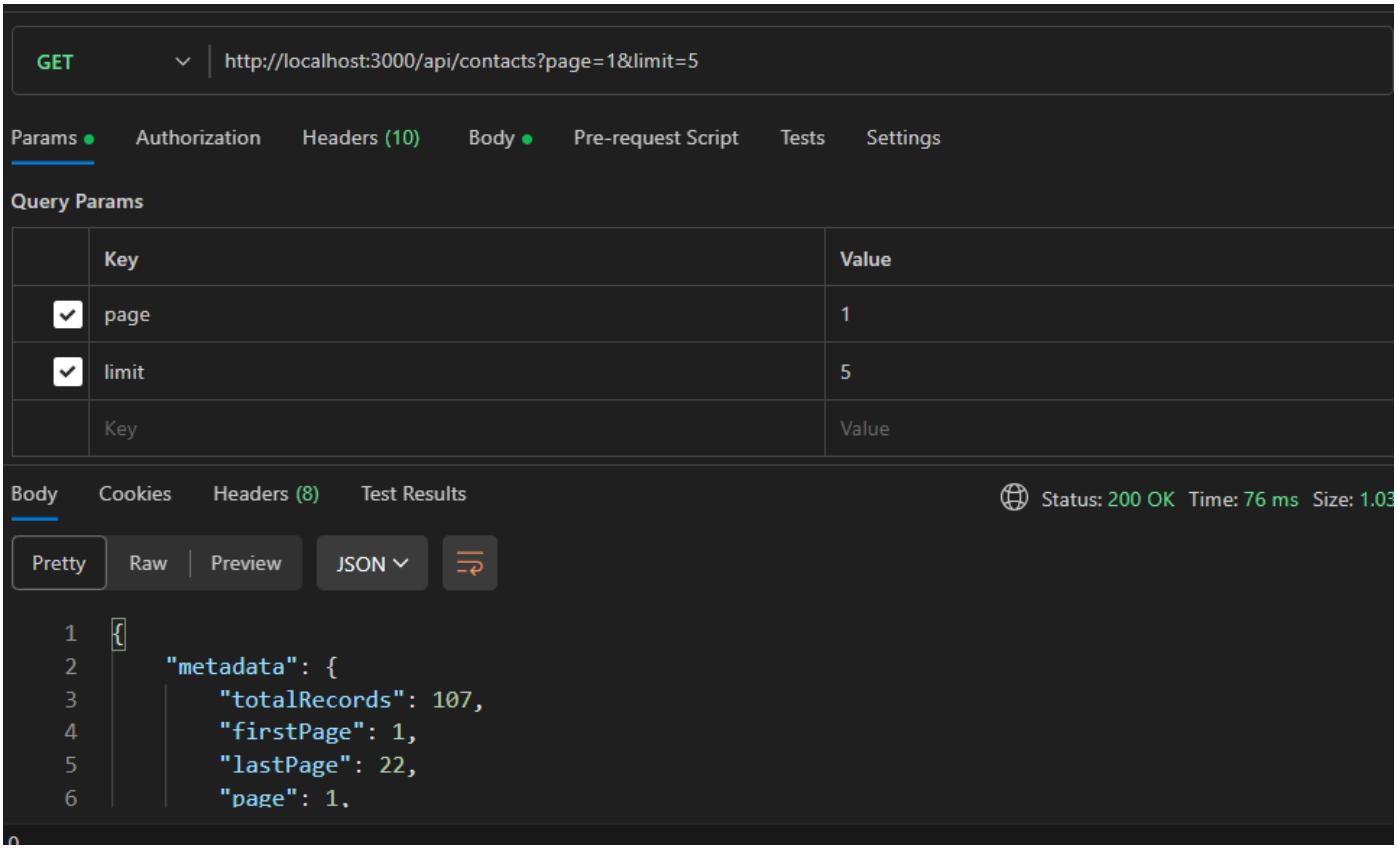
```
class Paginator {
    constructor(page = 1, limit = 5) {
        this.limit = parseInt(limit, 10);
            if (isNaN(this.limit) || this.limit < 1) {
        this.limit = 5;
    }

    this.page = parseInt(page, 10);
        if (isNaN(this.limit) || this.page < 1) {
        this.page = 1;
    }

        this.offset = (this.page - 1) * this.limit;
}

getMetadata(totalRecords) {
    if (totalRecords === 0) {
        return {};
    }

    let totalPages = Math.ceil(totalRecords / this.limit);
    return {
        totalRecords,
        firstPage: 1,
```

```
                lastPage: totalPages,
                page: this.page,
                limit: this.limit,
            };
        }
    }

    module.exports = Paginator;
```

Edit *getManyContacts(query)* (in *src/services/contacts.service.js*) as follows:

```
    async function getManyContacts(query) {
        const { name, favorite, page = 1, limit = 5 } = query;
        const paginator = new Paginator(page, limit);

        let results = await knex('contacts')
            .where((builder) => {
                if (name) {
                    builder.where('name', 'like', `%${name}%`);
                }
                if (favorite !== undefined) {
                    builder.where('favorite', 1);
                }
            })
            .select(
              knex.raw('count(id) OVER() AS recordsCount'),
                'id',
                'name',
                'email',
                'address',
                'phone',
                'favorite'
            )
            .limit(paginator.limit)
            .offset(paginator.offset);

        let totalRecords = 0;
        results = results.map((result) => {
            totalRecords = result.recordsCount;
            delete result.recordsCount;
            return result;
        });

        return {
          metadata: paginator.getMetadata(totalRecords),
            contacts: results,
        };
    }
```

Use a HTTP client to verify the handler works correctly with different sets of page and limit parameters.



Implement getContact handler

Edit *src/controllers/contacts.controller.js*:

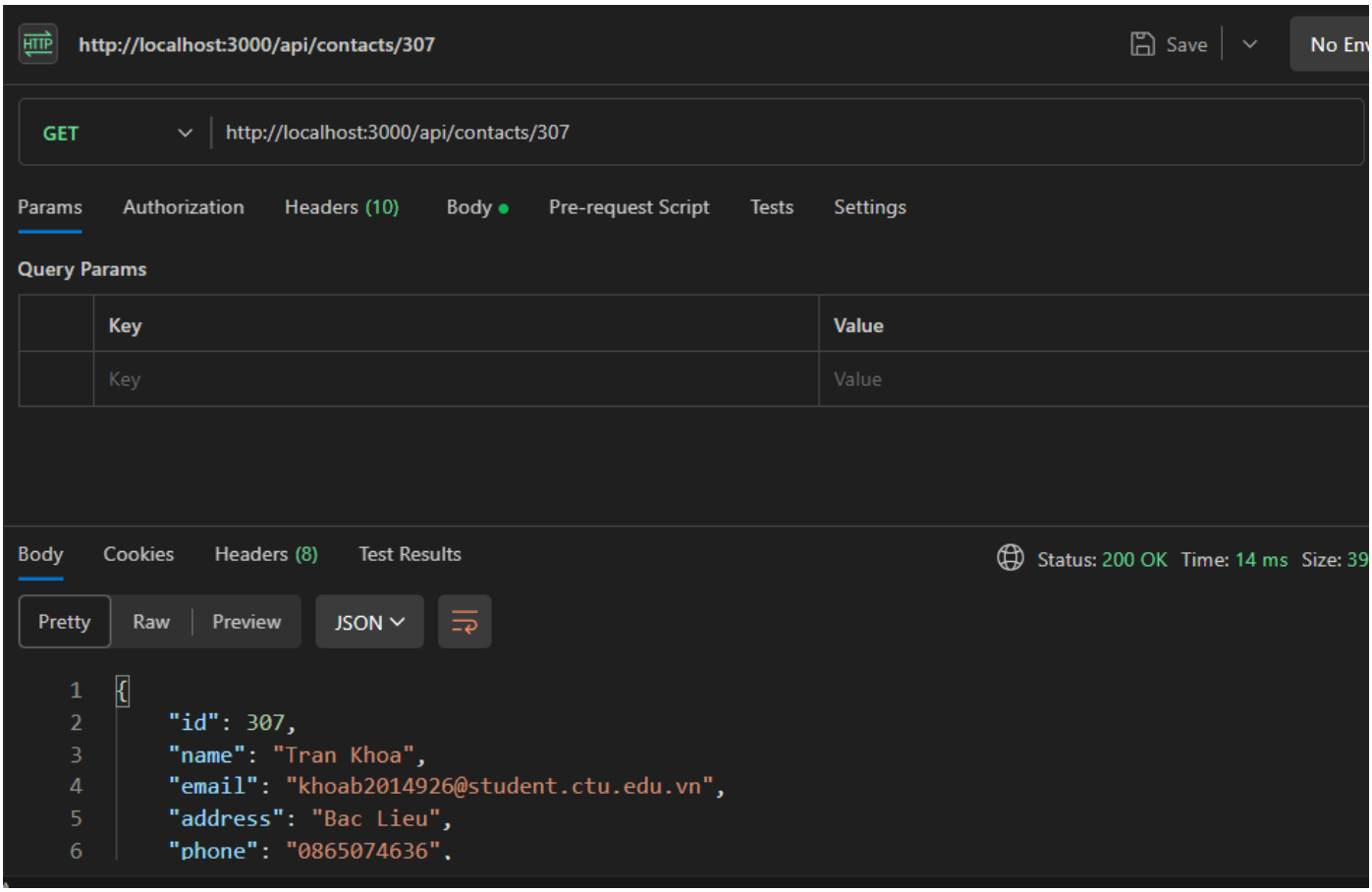
```
// Find a single contact with an id
async function getContact(req, res, next) {
    try {
        const contactsService = makeContactsService();
        const contact = await contactsService.getContactById(req.params.id);
        if (!contact) {
            return next(new ApiError(404, 'Contact not found'));
        }
        return res.send(contact);
    } catch (error) {
        console.log(error);
        return next(
            new ApiError(
                500,
                `Error retrieving contact with id=${req.params.id}`
            )
        );
    }
}
```

contactsService.getContactById(id) searches a contact by ID. The function *getContactById(id)* can be defined as follows:

```
...
function makeContactsService() {
    ...
    async function getContactById(id) {
```

```
            return knex('contacts').where('id', id).select('*').first();
        }

        return {
            createContact,
            getManyContacts,
            getContactById,
        };
    }
...
```

Use a HTTP client to verify the handler works correctly.



Implement updateContact handler

Edit *src/controllers/contacts.controller.js*:

```
// Update a contact by the id in the request
async function updateContact(req, res, next) {
    if (Object.keys(req.body).length === 0) {
        return next(new ApiError(400, 'Data to update can not be empty'));
    }

    try {
        const contactsService = makeContactsService();
        const updated = await contactsService.updateContact(
            req.params.id,
            req.body
        );
        if (!updated) {
            return next(new ApiError(404, 'Contact not found'));
        }
        return res.send({ message: 'Contact was updated successfully' });
    } catch (error) {
        console.log(error);
        return next(
            new ApiError(500, `Error updating contact with id=${req.params.id}`)
        );
    }
}
```

contactsService.updateContact(id, payload) searches contact by ID and update this contact with *payload*. The function *updateContact(id, payload)* can be defined as follows:

```
...
function makeContactsService() {
    ...
    async function updateContact(id, payload) {
        const update = readContact(payload);
            return knex('contacts').where('id', id).update(update);
    }

    return {
        createContact,
        getManyContacts,
        getContactById,
        updateContact,
    };
}
...
```

Use a HTTP client to verify the handler works correctly.

PUT http://localhost:3000/api/contacts/307

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "name": "Tran Khoa",
    "email": "khoab2014926@student.ctu.edu.vn",
    "address": "Vinh Long",
    "phone": "0865074636",
    "favorite": "1"
```

Body Cookies Headers (8) Test Results Status: 200

Pretty Raw Preview JSON

```
{
    "message": "Contact was updated successfully"
}
```

http://localhost:3000/api/contacts/307

GET http://localhost:3000/api/contacts/307

Params Authorization Headers (10) Body Pre-request Script Tests Settings

Query Params

	Key	Value
	Key	Value

Body Cookies Headers (8) Test Results Status: 200 OK

Pretty Raw Preview JSON

```
{
    "id": 307,
    "name": "Tran Khoa",
    "email": "khoab2014926@student.ctu.edu.vn",
    "address": "Vinh Long",
    "phone": "0865074636",
```

Implement deleteContact handler

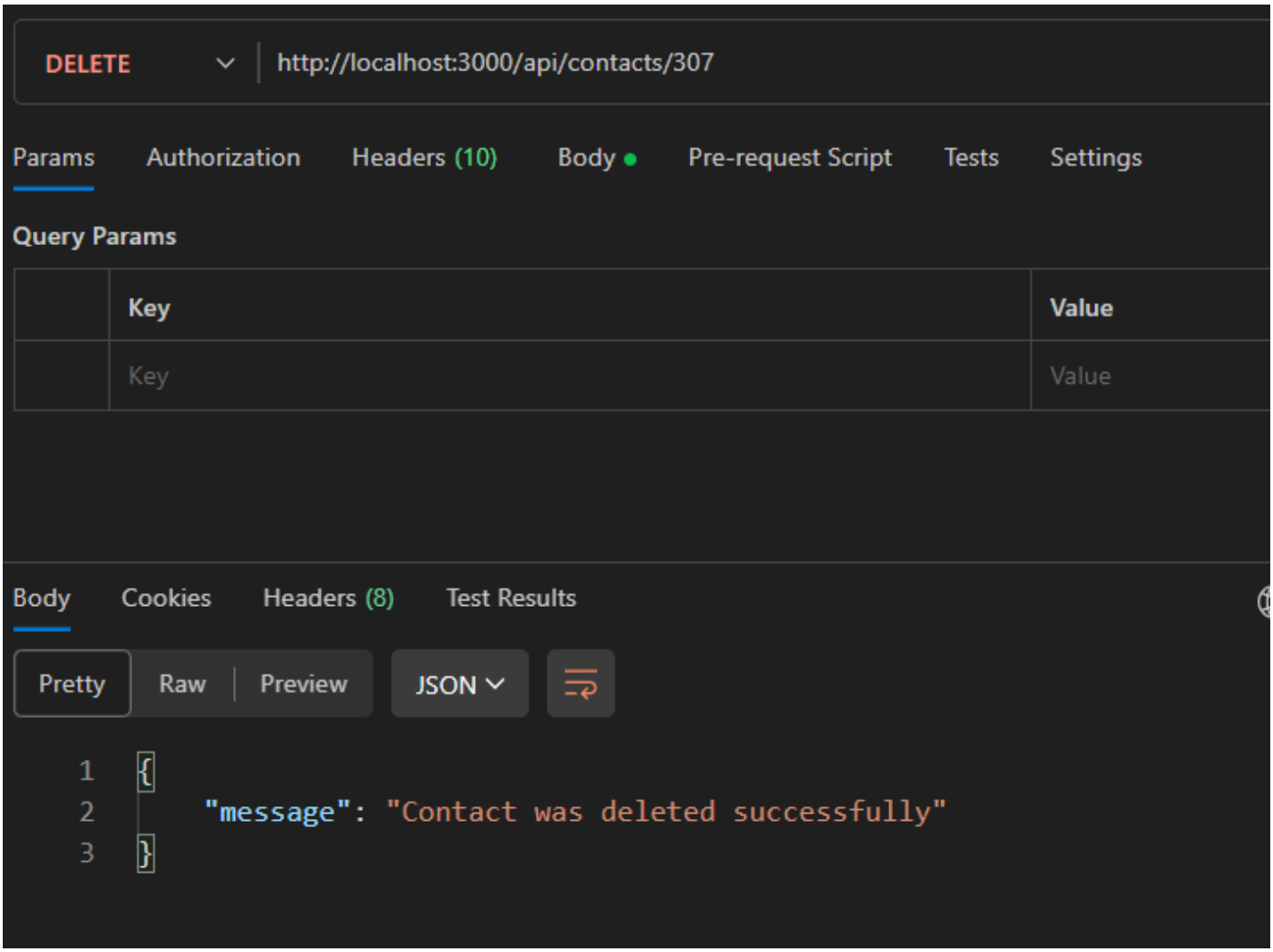
Edit *src/controllers/contacts.controller.js*:

```
// Delete a contact with the specified id in the request
async function deleteContact(req, res, next) {
    try {
        const contactsService = makeContactsService();
        const deleted = await contactsService.deleteContact(req.params.id);
        if (!deleted) {
            return next(new ApiError(404, 'Contact not found'));
        }
        return res.send({ message: 'Contact was deleted successfully' });
    } catch (error) {
        console.log(error);
        return next(
            new ApiError(
                500,
                `Could not delete contact with id=${req.params.id}`
            )
        );
    }
}
```

contactsService.deleteContact(id) searches contact by ID and deletes this contact. The function *deleteContact(id)* can be defined as follows:

```
...
function makeContactsService() {
    ...
    async function deleteContact(id) {
        return knex('contacts').where('id', id).del();
    }

    return {
        createContact,
        getManyContacts,
        getContactById,
        updateContact,
        deleteContact,
    };
}
...
```

Use a HTTP client to verify the handler works correctly.



Implement deleteAllContacts handler

Edit *src/controllers/contacts.controller.js*:

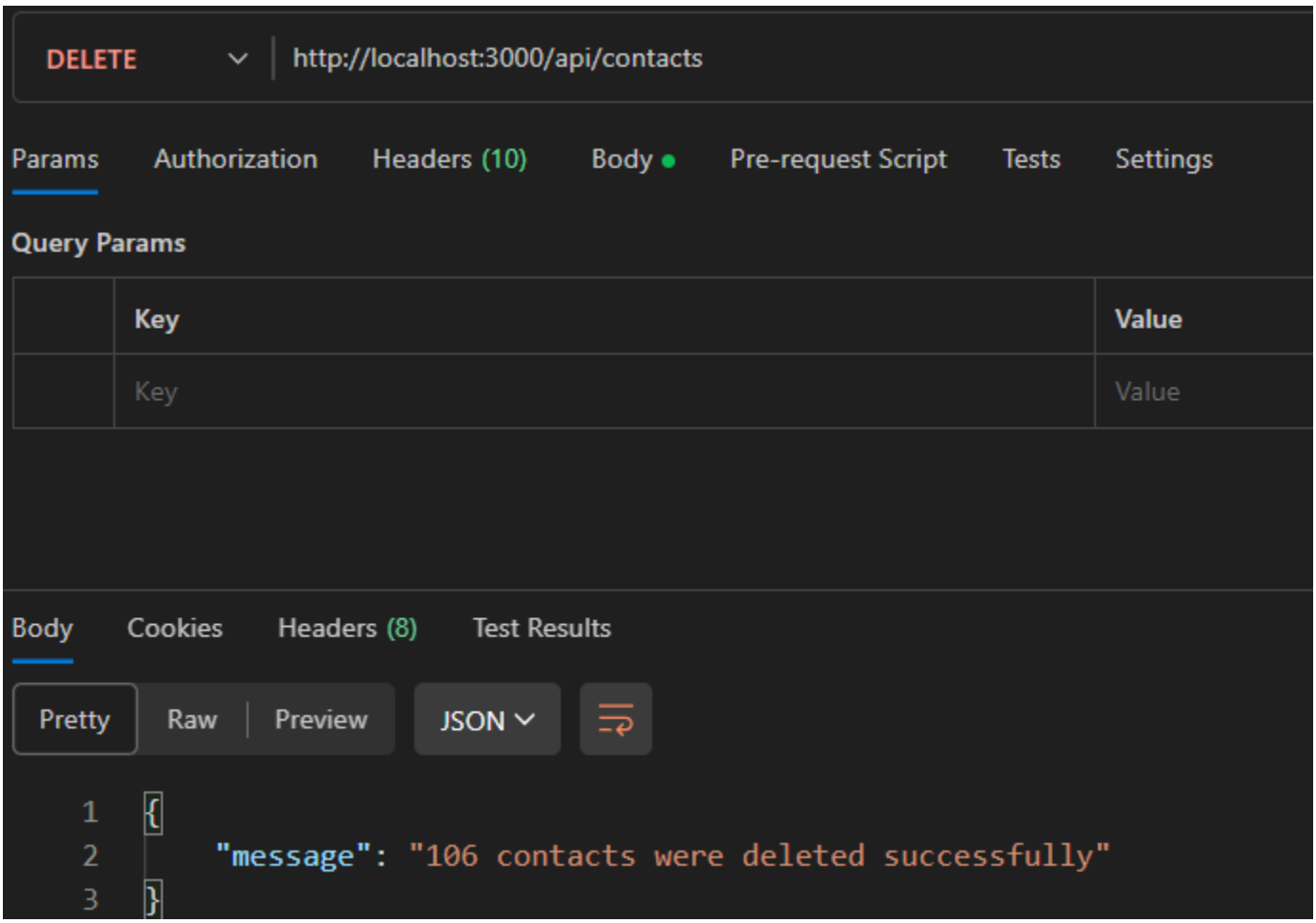
```
// Delete all contacts of a user from the database
async function deleteAllContacts(req, res, next) {
    try {
        const contactsService = makeContactsService();
        const deleted = await contactsService.deleteAllContacts();
        return res.send({
            message: `${deleted} contacts were deleted successfully`,
        });
    } catch (error) {
        console.log(error);
        return next(
            new ApiError(500, 'An error occurred while removing all contacts')
        );
    }
}
```

contactsService.deleteAllContacts() removes all contacts. The function *deleteAllContacts()* can be defined as follows:

```
...
function makeContactsService() {
    ...
    async function deleteAllContacts() {
        return knex('contacts').del();
    }
```

```
    return {
        createContact,
        getManyContacts,
        getContactById,
        updateContact,
        deleteContact,
        deleteAllContacts,
    };
}
...
```

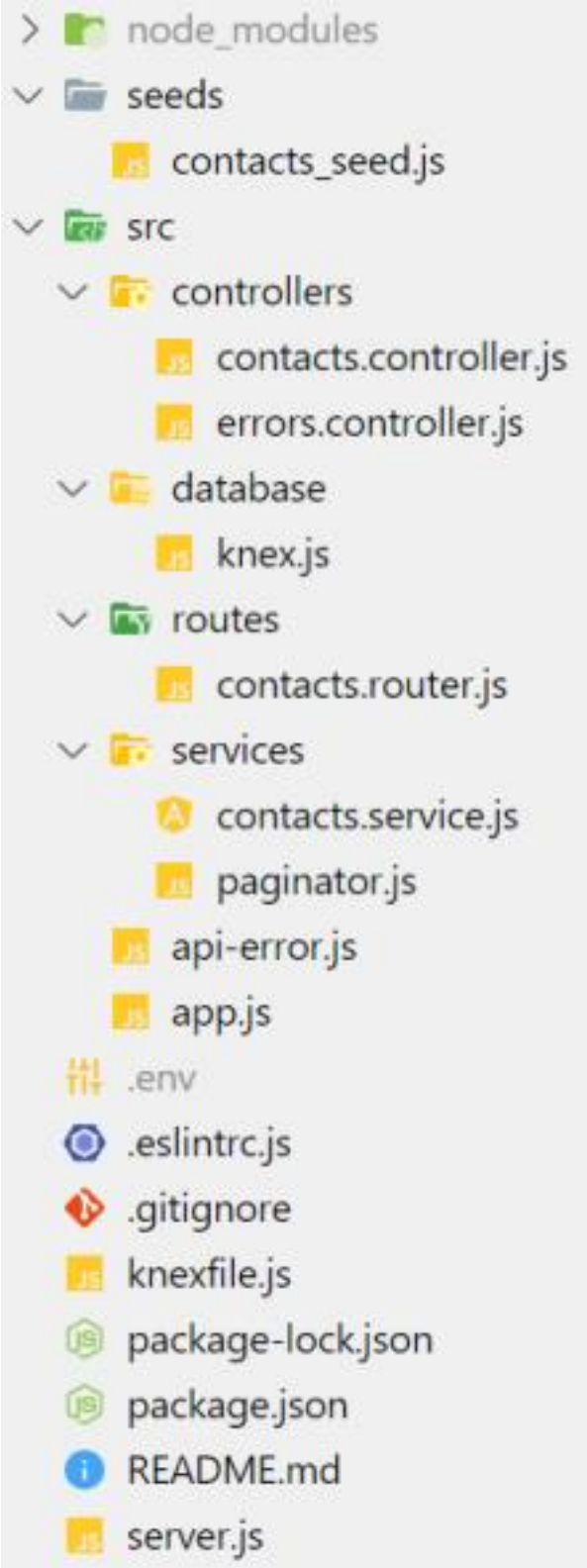
Use a HTTP client to verify the handler works correctly.



Make sure all handlers work correctly, then commit changes to git and GitHub:

```
git add -u
git add src/database src/services
git commit -m "Implement handlers"
git push origin master ## Upload local commits to GitHub
```

The directory struture for the project currently is as follows:


node_modules
seeds
contacts_seed.js
src
controllers
contacts.controller.js
errors.controller.js
database
knex.js
routes
contacts.router.js
services
contacts.service.js
paginator.js
api-error.js
app.js
.env
.eslintrc.js
.gitignore
knexfile.js
package-lock.json
package.json
README.md
server.js