

Trần Đăng Khoa
B2014926
M02

Repo: <https://github.com/23-24Sem1Courses/ct313hm02-contactbook-bekhoadangtran.git>

<https://github.com/23-24Sem1Courses/ct313hm02-contactbook-fe-khoadangtran.git>

CT313H: WEB TECHNOLOGIES AND SERVICES

Building Contactbook App - Backend - Part 1

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 screenshot 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.

Step 0: Install node and git

Download and install nodejs: <https://nodejs.org/en/download/>. You can download and install nodejs directly or use nvm (<https://github.com/coreybutler/nvm-windows>).

Download and install git: <https://git-scm.com/download/win>. When you are installing git on Windows, please check the option to install **Git Credential Manager (GCM)**. This option will help you log in to GitHub easier from command-line.

Verify your setup by typing the `node` and `git` commands in a terminal:

```
C:\Users\PC>node -v
v18.17.1

C:\Users\PC>git --version
git version 2.42.0.windows.2

C:\Users\PC>
```

(In case that you can't run the commands in a terminal, verify that the PATH environment variable on your machine includes the paths to `node` and `git` binaries).

Step 1: Create a node project

Clone the GitHub repo to your local machine:

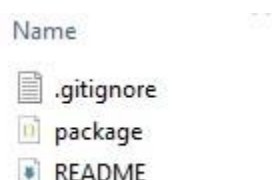
```
git clone <đường-link-đến-repo-GitHub-đã-nhận> contactbook-backend
```

(Of course, you can use whatever name you like but it shouldn't include spaces or special, non-ascii characters).

GitHub will ask for login information. If you have installed **Git Credential Manager (GCM)**, the login will be done via the browser. In case you forget or cannot install GCM, you need to create a personal access token (PAT) on GitHub and use it as a password (checkout a separate guide on how to create a PAT token). Another option is to use [GitHub Desktop](#).

Go to the project directory and init a nodejs project:

```
cd contactbook-backend npm
init -y
```



A `package.json` file will be created in the project directory.

Step 2: Manage the source code with git and GitHub

Download a `.gitignore` file: `npx gitignore node`. The `.gitignore` file list files and directories that will be ignored by git (e.g., `node_modules`).

In the project directory, run `git status`. Git shows that `.gitignore` and `package.json` files are currently not managed by git.

Ask git to manage these files:

```
git add .gitignore package.json
```

Run `git status` to verify:

```
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   .gitignore
    new file:   package.json
```










Run `git commit -m "Project setup"` to save the changes. The `-m` option allows to specify a comment for the commit. If it's the first time `git commit` is used on the machine, git will ask for a name and an email address. Run the following two git config commands to give git your name and email address:

```
git config --global user.email "you@example.com" git
config --global user.name "Your Name"
```

Rerun the `git commit` command.

Push local commits to GitHub as follows:

```
git push origin master
```

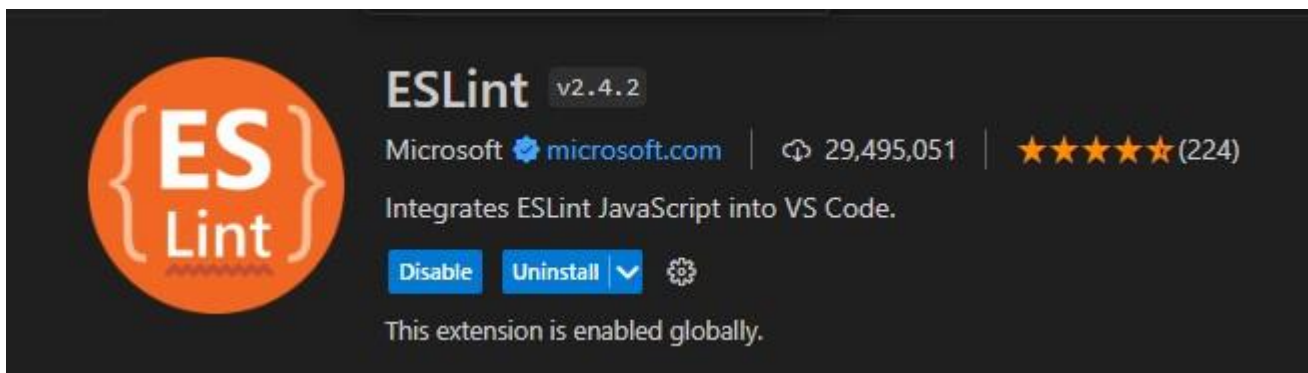
 khoadangtran baitap	d44f07d 32 minutes ago	 14 commits
 src	Project setup	47 minutes ago
 .eslintrc.js	Project setup	52 minutes ago
 .gitignore	Project setup	2 hours ago
 README.md	Update README.md	3 days ago
 package-lock.json	Configure eslint and prettier	1 hour ago
 package.json	Configure eslint and prettier	1 hour ago
 server.js	Project setup	51 minutes ago

in which, `origin` is the default name given to the remote git repo on GitHub when the repo is cloned.

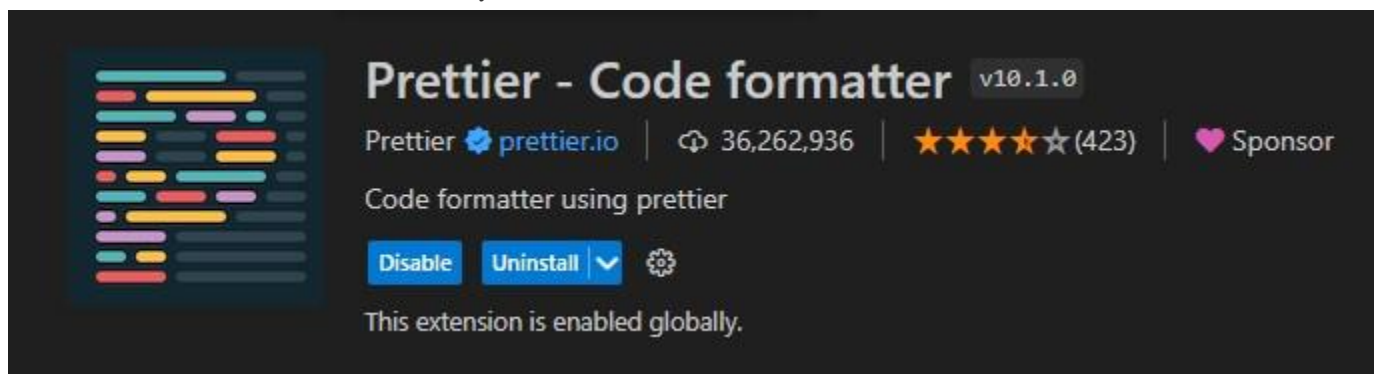
Verify that the files are uploaded to GitHub.

Step 3: Setup Visual Studio Code and ESLint and Prettier

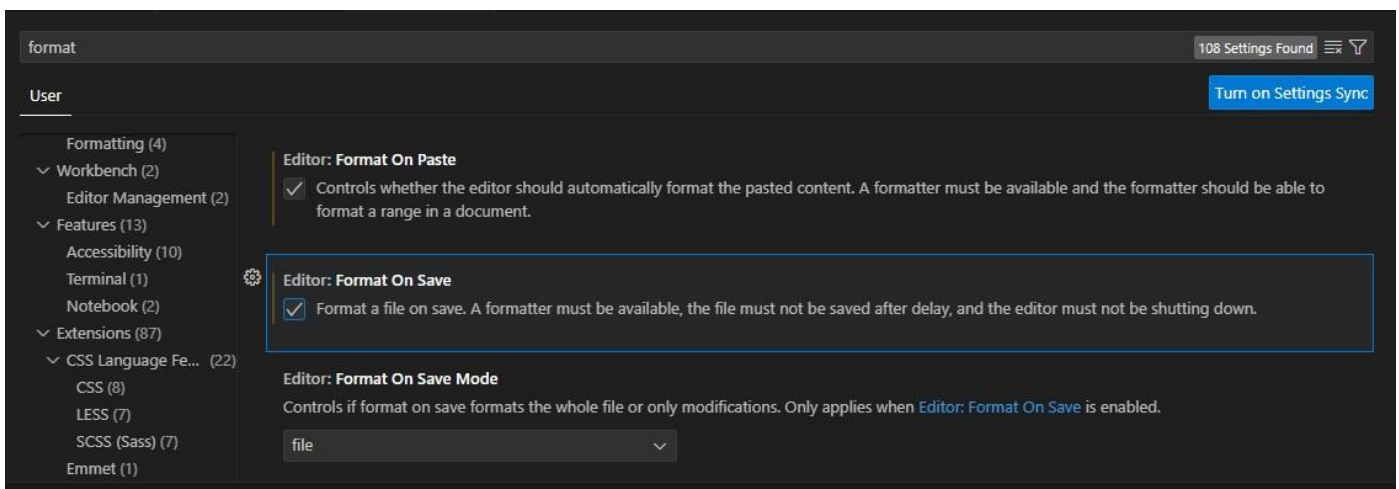
ESLint is a tool that statically analyzes JavaScript code and helps find and fix problems in the source code. Install [ESLint extension](#) for Visual Studio Code.



Prettier is a tool to automatically format the source code. Install [Prettier extension](#) for VSCode.



Configure VSCode to automatically format the source code on save or paste: Go to File > Settings, search for "format" and then check "Editor: Format on Paste" and "Editor: Format on Save":



In order for eslint and prettier work with VSCode, you also need to install eslint and prettier packages globally or locally in each project. In the project directory, install the following packages:

```
npm i -D eslint prettier eslint-config-prettier
```

```
PS C:\Users\PC\contactbook-backend> npm i -D eslint prettier eslint-config-prettier
Debugger attached.

added 100 packages, and audited 101 packages in 40s

24 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
Waiting for the debugger to disconnect...
PS C:\Users\PC\contactbook-backend>
```

(Eslint rules may conflict with prettier rules. To avoid this, you can install `eslint-config-prettier` package. This package turns off all eslint rules related to formatting the source code).

Then create a file named `.eslintrc.js` in the project directory as follows:

```
module.exports = { env: {
  node: true,
  commonjs: true,
  es2021: true,
},
  extends: ['eslint:recommended', 'prettier'],
};
```

(The `.eslintrc.js` file can be generated by issuing the command: `npx eslint --init` and answering some configuration questions).

Commit changes to git:

Add changes from previously committed files to git










`git add -u`

Add `.eslintrc.js` to git `git`

`add .eslintrc.js`

Save changes `git commit -m "Configure`

`eslint and prettier"`

 khoadangtran baitap	d44f07d 32 minutes ago	 14 commits
 src	Project setup	47 minutes ago
 .eslintrc.js	Project setup	52 minutes ago
 .gitignore	Project setup	2 hours ago
 README.md	Update README.md	3 days ago
 package-lock.json	Configure eslint and prettier	1 hour ago
 package.json	Configure eslint and prettier	1 hour ago
 server.js	Project setup	51 minutes ago

Step 4: Install Express

Install the following packages: `npm install express cors`.

```
PS C:\Users\PC\contactbook.backend> npm install express cors
added 65 packages, and audited 166 packages in 7s
32 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
```

In the project directory, create a `src` directory and a `src/app.js` file:

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

const app = express();

app.use(cors());
app.use(express.json());

app.get('/', (req, res) => {
  res.json({ message: 'Welcome to contact book application.' });
});

module.exports = app;
```

Install `dotenv` package: `npm i dotenv`. This package loads environment variables defined in a `.env` file to `process.env` object. In the project directory, create a `.env` file containing environment variables used by the server (note that the `.env` file will not be managed by git):

`PORT=3000`

In the project directory, create `server.js` to run the app:

```
require('dotenv').config();
const app = require('./src/app');

// start server
const PORT = process.env.PORT;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Open a terminal in the project directory and start the server: `node server.js`. Access to <http://localhost:3000/> to verify the result.

localhost:3000

Install `nodemon` to help monitor changes in the source code and restart the server automatically:

`npm install nodemon --save-dev`

```
PS C:\Users\PC\contactbook.backend> npm install nodemon --save-dev
added 26 packages, and audited 193 packages in 4s
36 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
```

Open `package.json` and change the "scripts" section as follows:

```
...
"scripts": {
  "start": "nodemon server.js"
},
```


...

With the above configuration, the server can be started by issuing the command `npm run start` instead of `node server.js` and every time a project file changes, `nodemon` will restart the server for us.

Commit changes to git repo:

Add changes from previously committed files to git

`git add -u`










Add newly created files and directories to git `git`

`add src/ server.js package-lock.json`

Save changes

`git commit -m "Install express and show a welcome message"`

Before committing changes, it's good idea to check if there are missing files by running `git status`.

 khoadangtran baitap	d44f07d 32 minutes ago	 14 commits
 src	Project setup	47 minutes ago
 .eslintrc.js	Project setup	52 minutes ago
 .gitignore	Project setup	2 hours ago
 README.md	Update README.md	3 days ago
 package-lock.json	Configure eslint and prettier	1 hour ago
 package.json	Configure eslint and prettier	1 hour ago
 server.js	Project setup	51 minutes ago

Step 5: Define controller and routes

Create `src`, `src/controllers` directories and `src/controllers/contacts.controller.js` file as follows:

```

function createContact(req, res) {
  return res.send({ message: 'createContact handler' });
}

function getContactsByFilter(req, res) {
  let filters = [];
  const { favorite, name } = req.query;

  if (favorite === undefined) {
    filters.push('favorite=${favorite}');
  }
  if (name) {
    filters.push('name=${name}');
  }
  return res.send({
    message: 'getContactsByFilter handler with query {
      ${filters.join(', ')}
    }',
  });
}

function getContact(req, res) {
  return res.send({ message: 'getContact handler' });
}

function updateContact(req, res) {
  return res.send({ message: 'updateContact handler' });
}

function deleteContact(req, res) {
  return res.send({ message: 'deleteContact handler' });
}

function deleteAllContacts(req, res) {
  return res.send({ message: 'deleteAllContacts handler' });
}

module.exports = {
  getContactsByFilter,
  deleteAllContacts,
  getContact,
  createContact,
  updateContact,
  deleteContact,
};

```

Create `src/routes` directory and `src/routes/contacts.router.js` file:

```

const express = require('express'); const contactsController =
require('../controllers/contacts.controller'); const router =

```

```

express.Router();

```

```

router

```

```

  .route('/')
  .get(contactsController.getContactsByFilter)
  .post(contactsController.createContact)
  .delete(contactsController.deleteAllContacts)

```

```

router

```

```

  .route('/:id')
  .get(contactsController.getContact)
  .put(contactsController.updateContact)

```



```
.delete(contactsController.deleteContact)
```

```
module.exports = router;
```

In the above code, we define routes for managing contacts required by the app. Each route is a combination of a path, a HTTP method (GET, POST, PATCH, PUT, DELETE) and a handler. Next, register the routes to the express app by updating *src/app.js* as follows:

```
...
const contactsRouter = require('./routes/contacts.router');
...
app.get('/', (req, res) => {
  res.json({ message: 'Welcome to contactbook application.' });
});

app.use('/api/contacts', contactsRouter);

module.exports = app;
```

URIs for contact resource will be started with */api/contacts*. For example, to ask the server to return a list of favorite contacts, client needs to issue a HTTP GET request to */api/contacts?favorite*.

Use a HTTP client to verify all the defined routes.

If the code works correctly, commit changes to git:

```
git add -u git
add src/
git commit -m "Define routes for managing contacts"
```

Step 6: Implement error handlers

Create *src/api-error.js* file:

```
class ApiError extends Error {
  constructor(statusCode, message, headers = {}) {
    super();
    this.statusCode = statusCode;
    this.message = message;
    this.headers = headers;
  }
}

module.exports = ApiError;
```

Create *src/controller/errors.controller.js* file:

```

controllers > JS errors.controller.js > ...
const ApiError = require('../api-error');

function methodNotAllowed(req, res, next){
  if (req.route) {
    const httpMethods = Object.keys(req.route.methods)
      .filter((method) => method !== '_all')
      .map((method) => method.toUpperCase());
    return next(
      new ApiError(405, 'Method Not Allowed', {
        Allow: httpMethods.json(', '),
      })
    );
  }
  return next();
}

function resourceNotFound(req, res, next) {
  return next(new ApiError(404, 'Resource not found'));
}

function handleError(error, req, res, next) {
  if (res.headersSent) {
    return next(error);
  }

  return res
    .status(error.statusCode || 500)
    .set(error.headers || {})
    .json({
      message: error.message || 'Internal Server Error',
    });
}

```

Open `src/routes/contacts.router.js` and add error handling middlewares as follows:

```

... const { methodNotAllowed } =
require('../controllers/errors.controller'); ...

```

```

router
  .route('/')
  ...
  .all(methodNotAllowed);

```

```

router
  .route('/:id')
  ...
  .all(methodNotAllowed);

```

Edit `src/app.js` to add error handling middlewares:

```

...
const { resourceNotFound,
  handleError
} = require('../controllers/errors.controller');
...
app.use('/api/contacts', contactsRouter);

```

```
// Handle 404 response app.use(resourceNotFound);
```

```
// Define error-handling middleware last  
app.use(handleError);
```

...









Use a HTTP client and verify the followings:

Send a GET request to an unknown path, verify that the response is a 404 error with the "Resource Not Found" message.

Send a request to a known path but with an unsupported HTTP method (e.g., PUT /api/contacts), verify that the response is a 405 error with the "Method Not Allowed" message.

Commit changes to git and GitHub:

```
git add -u  
git add src/controllers/errors.controller.js src/api-error.js  
git commit -m "Implement an error handling middlewares" git  
push origin master # Upload local commits to GitHub
```

 khoadangtran baitap	d44f07d 9 hours ago	🕒 14 commits
 src	Project setup	9 hours ago
 .eslintrc.js	Project setup	9 hours ago
 .gitignore	Project setup	10 hours ago
 README.md	Update README.md	3 days ago
 package-lock.json	Configure eslint and prettier	10 hours ago
 package.json	Configure eslint and prettier	10 hours ago
 server.js	Project setup	9 hours ago

Step 7: Prepare database

Install MySQL or MariaDB on your machine if needed.

Use a MySQL client (phpMyAdmin, HeidiSQL, ...) to create a database named *ct313h_labs*. Next, create a contacts table as follows (you can also leverage knex migration for this task):

```
CREATE TABLE `contacts` (  
  `id` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,  
  `name` VARCHAR(255) NOT NULL,  
  `email` TEXT NULL DEFAULT NULL,  
  `address` VARCHAR(255) NULL DEFAULT NULL, `phone`  
  TINYTEXT NOT NULL,  
  `favorite` TINYINT(1) UNSIGNED NOT NULL DEFAULT '0', PRIMARY  
  KEY (`id`) USING BTREE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

Install `knex`, `mysql` and `faker-js` (checkout fake-js API [here](#)):

```
npm install knex mysql npm install  
@faker-js/faker --save-dev
```

Edit `.env` and add the database connection parameters:

```
PORT=3000
```

```
DB_HOST=localhost
DB_PORT=3306
DB_USER=root
DB_PASS=root
DB_NAME=ct313h_labs
```

Make sure to update the above parameters (user/password) according to your database setup.

In the project directory, create directory *seeds* and run `npm run knex init` to create *knexfile.js* file. Edit *knexfile.js* as follows:

```
require('dotenv').config(); const { DB_HOST, DB_PORT, DB_USER, DB_PASS, DB_NAME } =
    process.env;

/**
 * @type { import("knex").Knex.Config }
 */ module.exports = { client:
'mysql', connection: { host:
DB_HOST, port: DB_PORT,
user: DB_USER, password:
DB_PASS, database: DB_NAME,
    },
    pool: { min: 0, max: 10 },
    seeds: { directory:
        './seeds',
    },
};
```

Run `npm run knex seed:make contacts_seed` to create a seeding script for contacts table (*./seeds/contacts_seed.js*).

Edit the seeding script as follows:

```
const { faker } = require('@faker-js/faker');

function createContact() { return { name:
    faker.person.fullName(), email:
    faker.internet.email(), address:
    faker.location.streetAddress(), phone:
    faker.phone.number('09#####'), favorite:
    faker.number.int({ min: 0, max: 1,
        }),
    };
}

/**
 * @param { import("knex").Knex } knex
 * @returns { Promise<void> }
 */ exports.seed = async function
(knex) { await knex('contacts').del();
    await knex('contacts').insert(Array(100).fill().map(createContact)); };
```

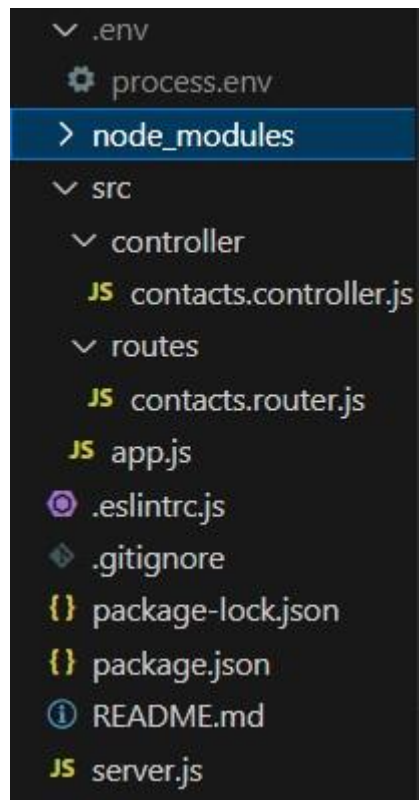
Run the seeding scripts in the seeds directory by the command: `npm run knex seed:run`. Verify that fake data are inserted into the database.

After verification, commit changes to git and GitHub:

```
git add -u git add seeds knexfile.js git commit -m
"Setup knex.js and insert fake data"
```

`git push origin master` # Upload local commits to GitHub The directory

structure for the project currently is as follows:



WEB TECHNOLOGIES AND SERVICES

Building Contactbook App - Backend - Part 2

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;

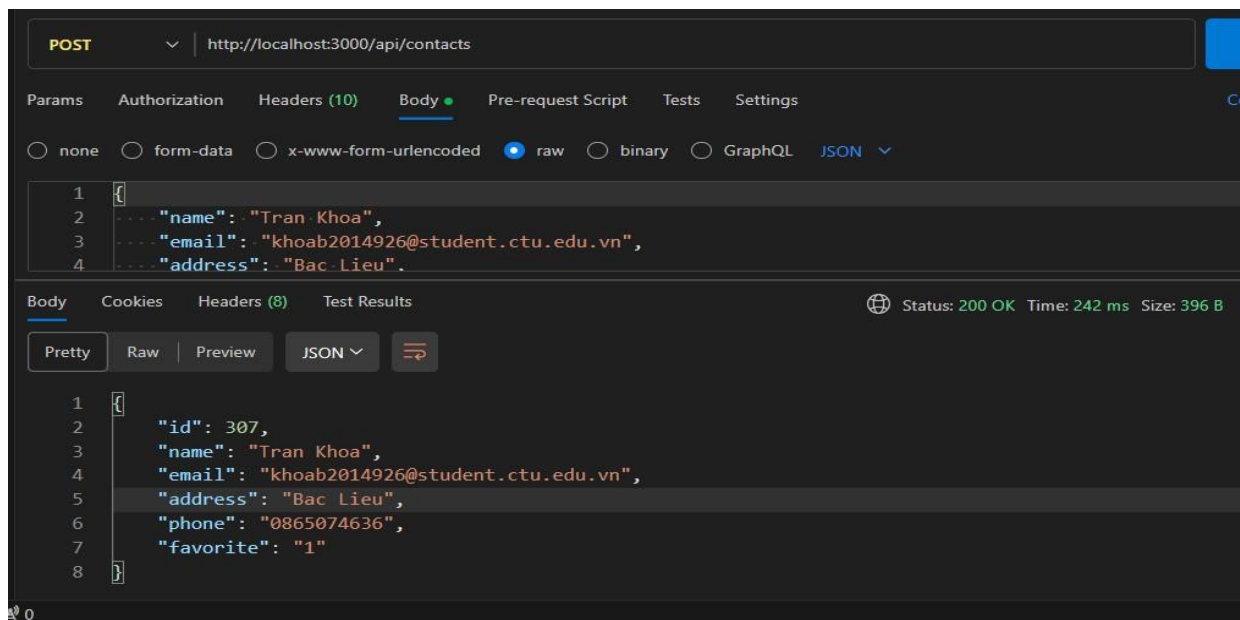
```

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

<input checked="" type="checkbox"/>	Postman-Token		<calculated when request is sent>
<input checked="" type="checkbox"/>	Content-Type		application/json
<input checked="" type="checkbox"/>	Content-Length		<calculated when request is sent>
<input checked="" type="checkbox"/>	Host		<calculated when request is sent>
<input checked="" type="checkbox"/>	User-Agent		PostmanRuntime/7.32.1
<input checked="" type="checkbox"/>	Accept		*/*
<input checked="" type="checkbox"/>	Accept-Encoding		gzip, deflate, br
<input checked="" type="checkbox"/>	Connection		keep-alive
<input checked="" type="checkbox"/>	Content-Type		application/json
	Key		Value



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.

The screenshot shows an HTTP client interface with the following details:

- URL:** `http://localhost:3000/api/contacts/307`
- Method:** `GET`
- Query Params:** A table with two columns: **Key** and **Value**. The first row contains the text "Key" in both columns.
- Body:** The response is displayed in JSON format:


```

{
  "id": 307,
  "name": "Tran Khoa",
  "email": "khoab2014926@student.ctu.edu.vn",
  "address": "Bac Lieu",
  "phone": "0865074636"
}

```
- Status:** `200 OK`, **Time:** `14 ms`, **Size:** `39`

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.

The screenshot shows a web browser's developer tools interface. The top bar indicates a GET request to `http://localhost:3000/api/contacts?page=1&limit=5`. Below this, the 'Query Params' section is expanded, showing a table with two parameters: 'page' with value '1' and 'limit' with value '5'. The 'Body' section is also expanded, showing the JSON response in 'Pretty' format. The response is an object with 'metadata' and 'contacts' properties. The 'metadata' object contains 'totalRecords' (107), 'firstPage' (1), 'lastPage' (22), and 'page' (1). The 'contacts' array is empty.

Key	Value
page	1
limit	5

```

{
  "metadata": {
    "totalRecords": 107,
    "firstPage": 1,
    "lastPage": 22,
    "page": 1,
  },
  "contacts": []
}

```

Implement getContact handler

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

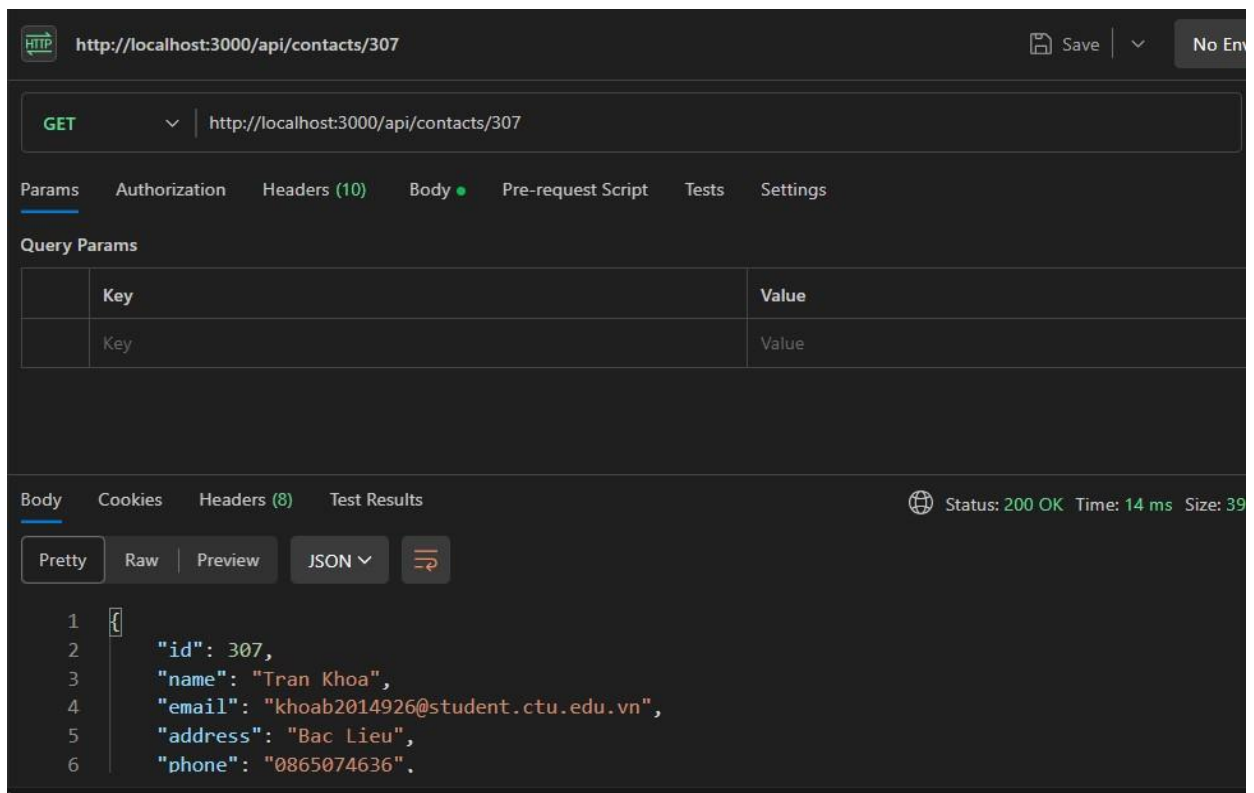
```
// Find a single contact with an id
async function getContact(req, res, next) {
  try {
    const contactsService = makeContactsService();
    const contact = await contactsService.getContactById(req.params.id);
    if (!contact) {
      return next(new ApiError(404, 'Contact not found'));
    }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

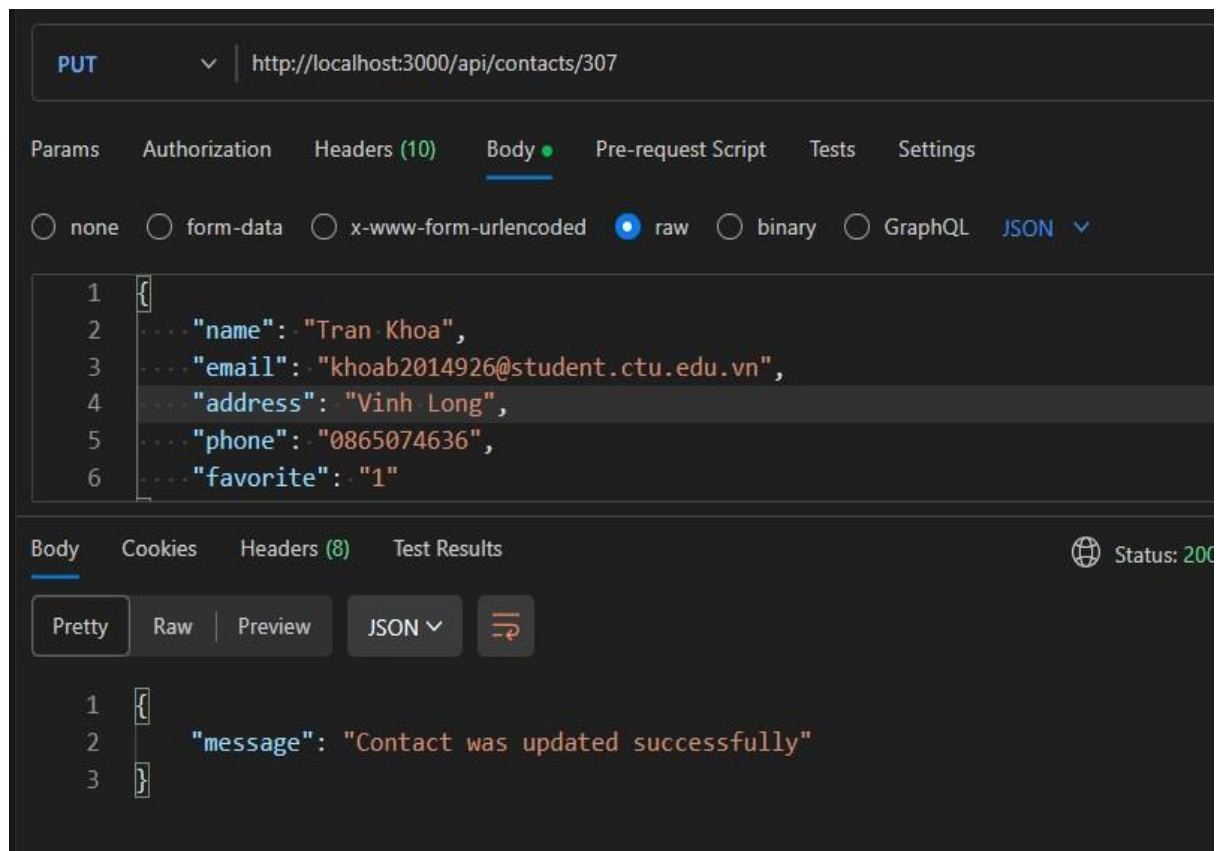
```

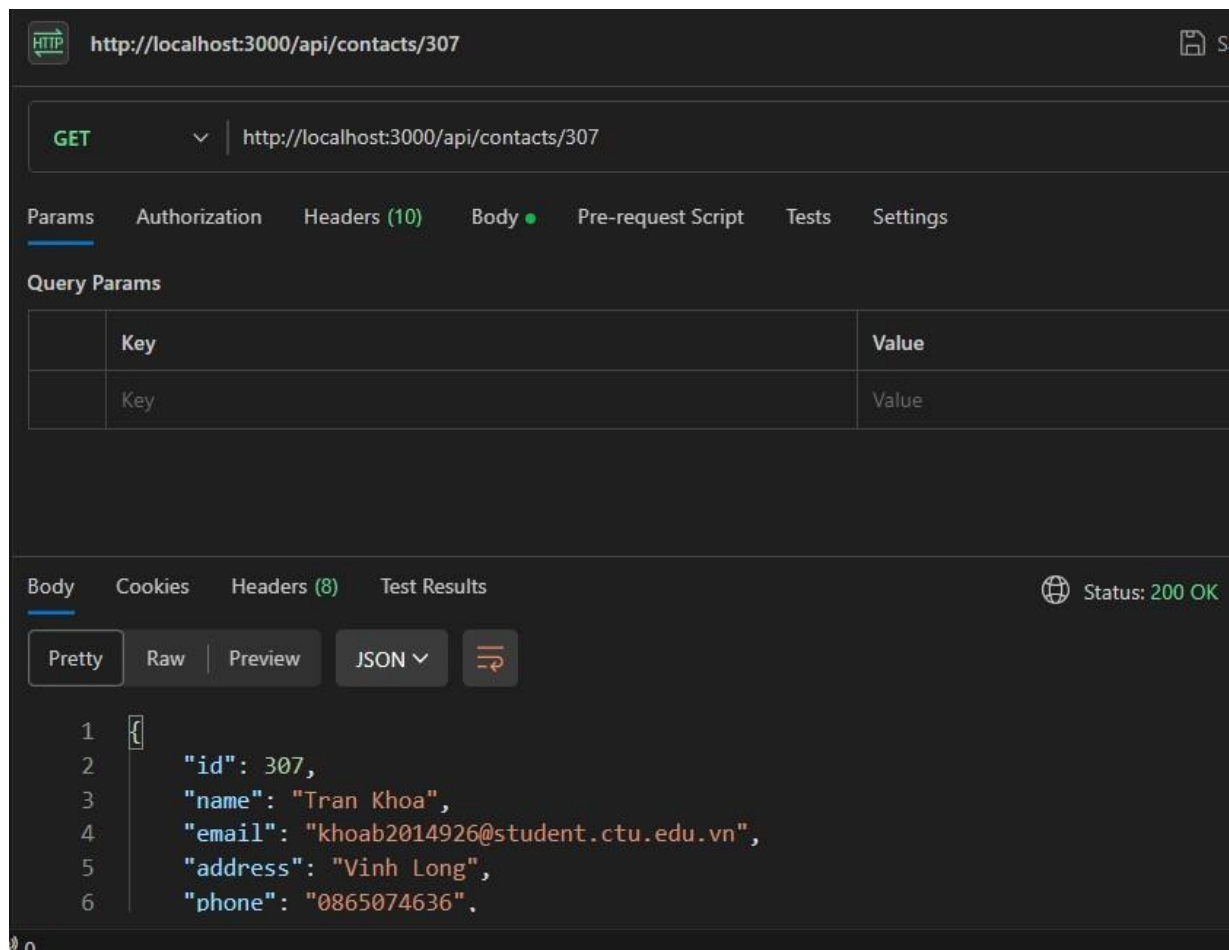
    }

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

```

Use a HTTP client to verify the handler works correctly.





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();
  }
}
```

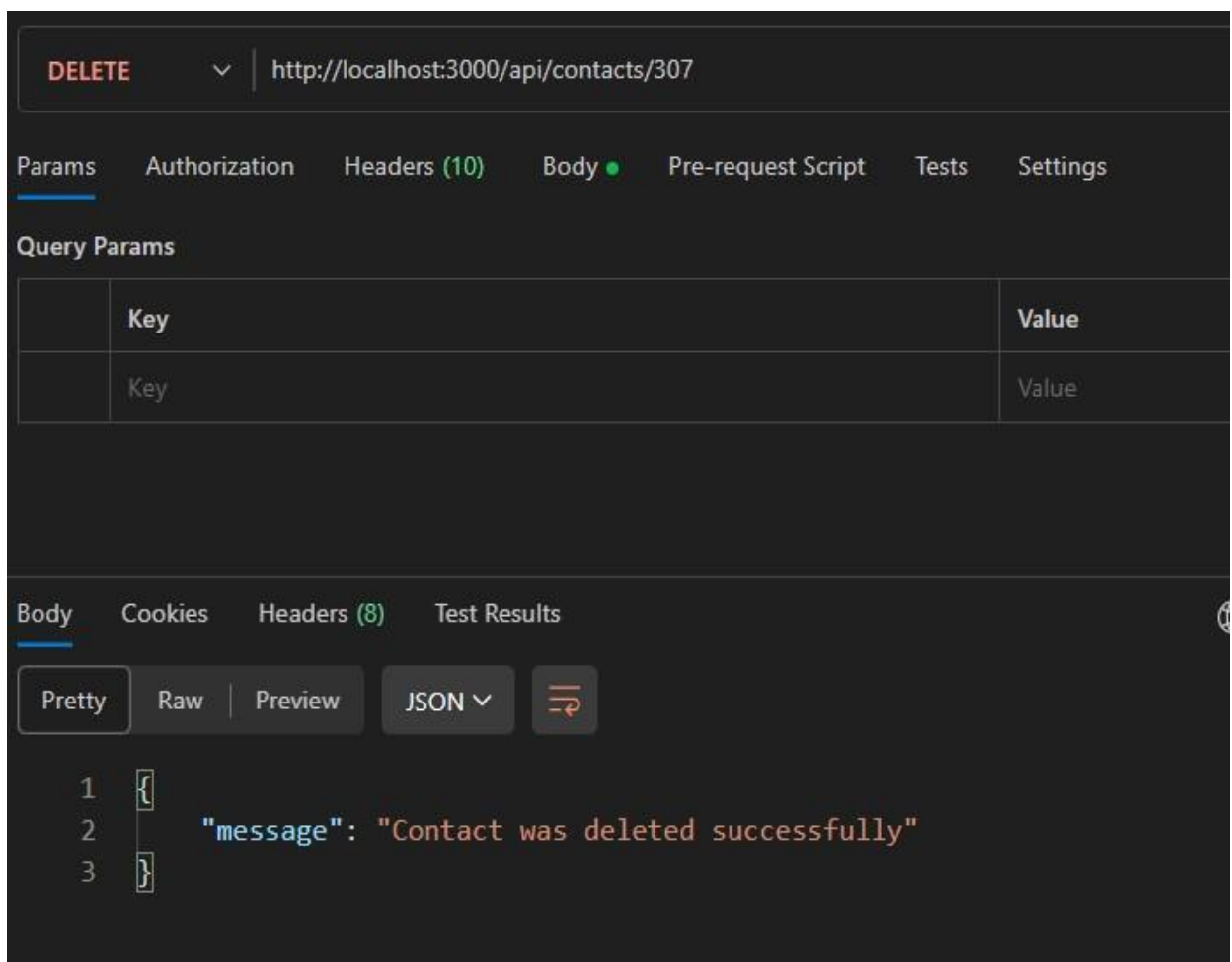
```

    }

    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.

DELETE | http://localhost:3000/api/contacts

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

Query Params

	Key	Value
	Key	Value

Body Cookies Headers (8) Test Results

Pretty Raw Preview JSON ↕

```
1 {
2   "message": "106 contacts were deleted successfully"
3 }
```

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

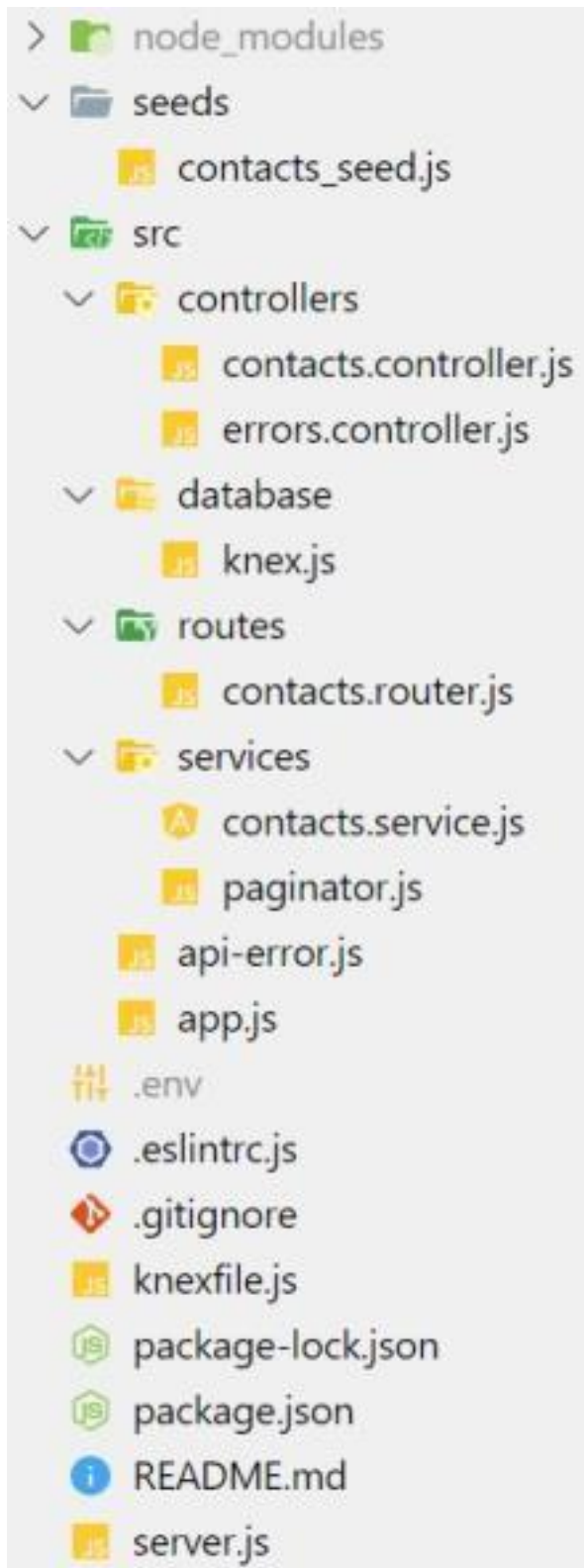
`git add -u`

`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:



CT313H: WEB TECHNOLOGIES AND SERVICES

Building Contactbook App - Frontend

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 next two lab sessions, you will build the app frontend.

The app is built as a SPA with Vue.js and has the following features:

A page showing a list of contacts, support pagination and the ability to filter any piece of information of a contact (name, email, phone, address, favorite) on each page of contacts.

A page to edit a contact.

A page to add a new contact.

Support for deleting a contact, all contacts.

A 404 error page for unknown paths.

The app uses the HTTP API built in the first two lab sessions. The source code is managed by git and uploaded to GitHub.

This step-by-step tutorial 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. Each functionality on a page may need several images to illustrate (e.g., images showing the implemented functionalities, successful and failed scenarios, results of the operations, ...). **You should NOT screenshot 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.

Step 0: Before getting started...

For easier debugging, [Vue.js devtools](#) extension should be installed in the browser. The HTTP API server built in the first two lab sessions should work correctly.

Step 1: Clone the repo from GitHub

Clone the GitHub repo to your machine:

```
git clone <đường-link-đến-repo-GitHub-đã-nhận> contactbook-frontend
```

Install dependencies

and

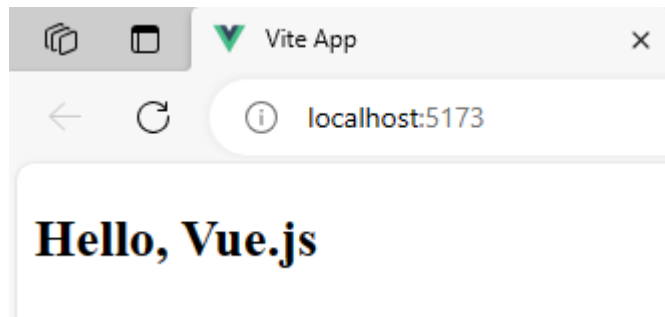
run the project in dev mode:

```
cd contactbook-frontend
```

```
npm install
```

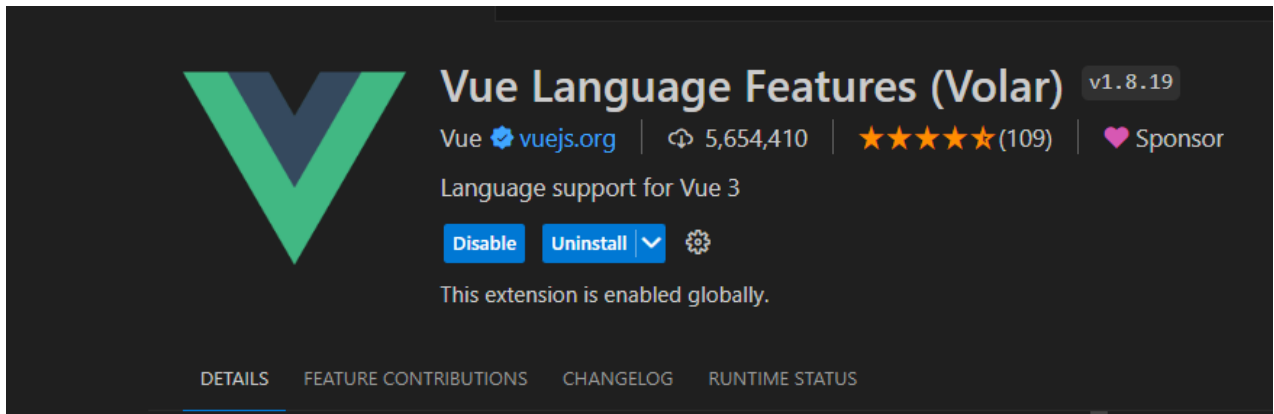
```
npm run dev
```

Open a browser, go to <http://localhost:5173/> to see the result.

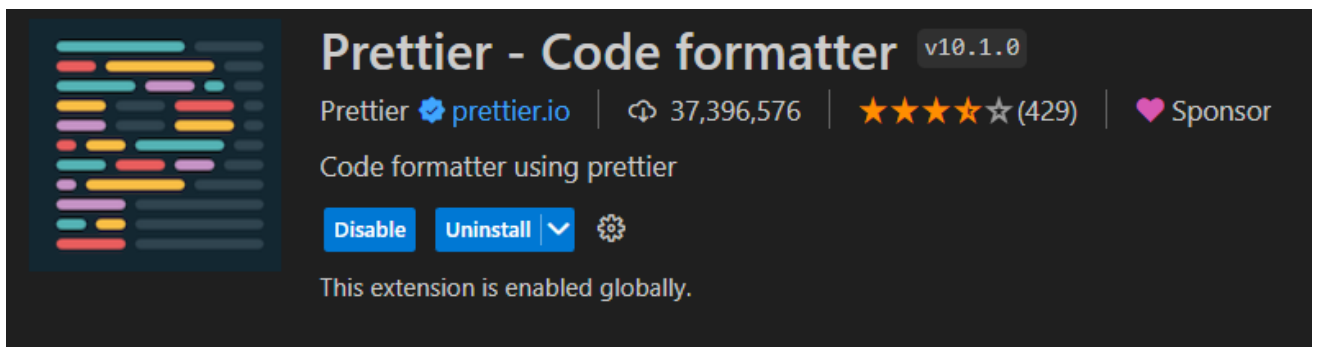
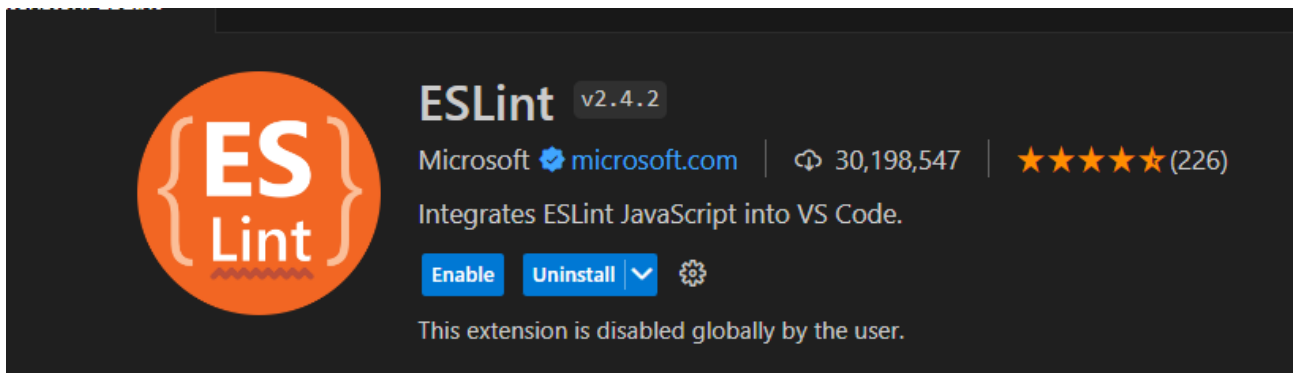


Step 2: Setup VSCode

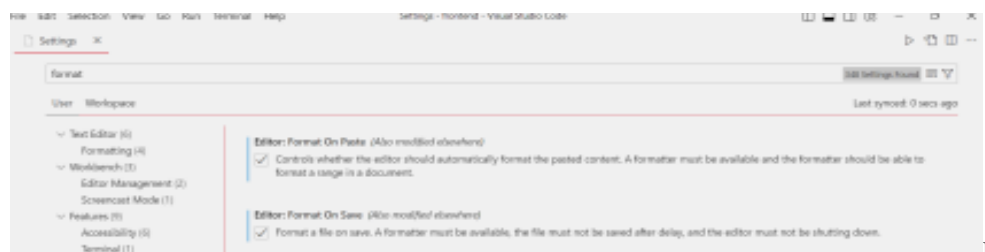
Install [Vue Language Features \(Volar\)](#) extension for VSCode.



Install [ESLint](#), [Prettier](#) extensions for VSCode if needed.



Configure VSCode to automatically format the source code on save or paste: Go to File > Settings, search for "format" and then check "Editor: Format on Paste" and "Editor: Format on Save":



In the project directory, add `jsconfig.json` file. This file helps VSCode understand the project structure:

```

{
  "compilerOptions": {
    "baseUrl": "./",
    "paths": {
      "@/*": ["src/*"]
    }
  }
}












```

Commit changes and push to GitHub:

```

cd contactbook-frontend
git add -A
git commit -m "Setup VSCode"
git push origin master

```

	khoadangtran Setup VSCode	ce2d04e now	🕒 2 commits
	public	Initial commit	17 minutes ago
	src	Initial commit	17 minutes ago
	.eslintrc.cjs	Initial commit	17 minutes ago
	.gitignore	Initial commit	17 minutes ago
	README.md	Initial commit	17 minutes ago
	index.html	Initial commit	17 minutes ago
	jsconfig.json	Setup VSCode	now
	package-lock.json	Setup VSCode	now
	package.json	Initial commit	17 minutes ago
	vite.config.js	Initial commit	17 minutes ago

Step 3: Create functions to get data from server

In the project directory, configure proxy to the API server in *vite.config.js*:

```

export default defineConfig({
  ...
  server: {
    proxy: {
      '/api': {
        target: 'http://localhost:3000/',
        changeOrigin: true,
      },
    },
  },
});

```

The above configuration means that when there is a HTTP request with an URI starting with */api* coming from the app (<http://localhost:5173/>), the target host of the request will be replaced with the address of the API server (<http://localhost:3000/>). For example, a request to <http://localhost:5173/api/contacts> will become

a request to <http://localhost:3000/api/contacts>.

Create *src/services/contacts.service.js* as follows:

```
function makeContactsService() {
  const baseUrl = '/api/contacts';

  const headers = {
    'Content-Type': 'application/json',
  };

  async function getContacts(page, limit = 5) {
    let url = `${baseUrl}?page=${page}&limit=${limit}`;
    return await fetch(url).then((res) => res.json());
  }

  async function createContact(contact) {
    return await fetch(baseUrl, {
      method: 'POST',
      headers,
      body: JSON.stringify(contact),
    }).then((res) => res.json());
  }

  async function deleteAllContacts() {
    return await fetch(baseUrl, {
      method: 'DELETE',
    }).then((res) => res.json());
  }

  async function getContact(id) {
    return await fetch(`${baseUrl}/${id}`).then((res) => res.json());
  }

  async function updateContact(id, contact) {
    return await fetch(`${baseUrl}/${id}`, {
      method: 'PUT',
      headers,
      body: JSON.stringify(contact),
    }).then((res) => res.json());
  }

  async function deleteContact(id) {
    return await fetch(`${baseUrl}/${id}`, {
      method: 'DELETE',
    }).then((res) => res.json());
  }

  return {
    getContacts,
    deleteAllContacts,
```

```

    getContact,
    createContact,
    updateContact,
    deleteContact,
  };
}

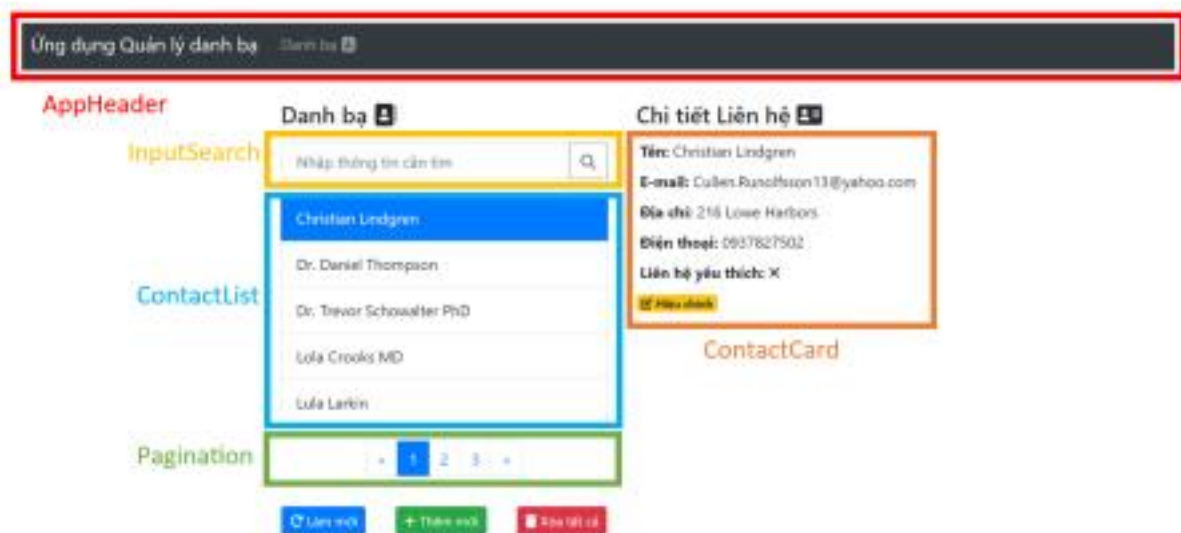
export default makeContactsService();

```

The *contacts.service.js* module defines functions interacting with the API server (built in lab sessions 1 and 2) by issuing the corresponding HTTP requests.

Step 4: Implement a page showing a list of contacts

The UI presented in this tutorial is as follows:



Bootstrap 4 and Font Awesome:

Install

```
npm i bootstrap@4 jquery popper.js @fortawesome/fontawesome-free
```

Edit *src/main.js* to

```

to the legacy v1
added 4 packages, and audited 191 packages in 13s
56 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\PC\contactbook-frontend>

```

import Bootstrap and Font Awesome:

```

import { createApp } from 'vue';
import 'bootstrap/dist/css/bootstrap.min.css';
import '@fortawesome/fontawesome-free/css/all.min.css';
import App from './App.vue';
...

```

Install *vue-router* : `npm i vue-router@4` , and create *src/router/index.js* as follows:


```
PS C:\Users\PC\contactbook-frontend> npm i vue-router@4
added 2 packages, and audited 193 packages in 6s

57 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\PC\contactbook-frontend>
```

```
import { createWebHistory, createRouter } from 'vue-router';

import ContactBook from '@/views/ContactBook.vue';

const routes = [
  {
    path: '/',
    name: 'contactbook',
    component: ContactBook,
  },
];

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes,
});

export default router;
```

The *import.meta.env* object contains environment variables for the app managed by Vite. *env.BASE_URL* returns the base URL of the app on the web server. This value comes from the "base" option in the *vite.config.js* file ("/" by default - the app is deployed right in the document root on the web server).

Open *src/main.js* and add the router to the app:

```
...

import router from './router';

createApp(App)
  .use(router)
  .mount('#app');
```

Define a placeholder page in *src/views/ContactBook.vue* as follows:

```
<template>
  <h2>ContactBook page</h2>
</template>
```

The root component in *src/App.vue* is updated as follows:

```
<script setup>
```

```

import AppHeader from '@components/AppHeader.vue';

</script>

<template>
  <AppHeader />

  <div class="container mt-3">
    <router-view />
  </div>
</template>

<style>
.page {
  max-width: 400px;
  margin: auto;
}
</style>

```

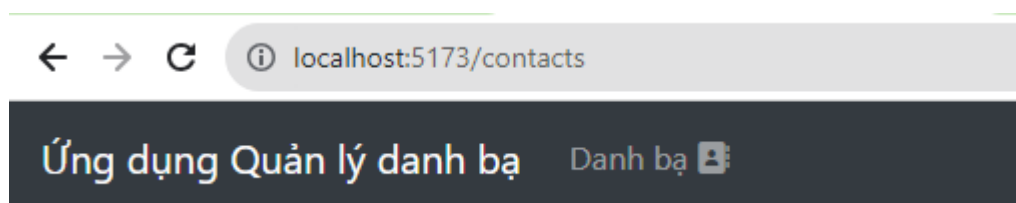
AppHeader component (*src/components/AppHeader.vue*) defines the app navigation bar:

```

<template>
  <nav class="navbar navbar-expand navbar-dark bg-dark">
    <a
      href="/"
      class="navbar-brand"
    >Ứng dụng Quản lý danh bạ</a>
    <div class="mr-auto navbar-nav">
      <li class="nav-item">
        <router-link
          :to="{ name: 'contactbook' }"
          class="nav-link"
        >
          Danh bạ
          <i class="fas fa-address-book"></i>
        </router-link>
      </li>
    </div>
  </nav>
</template>

```

Go to <http://localhost:5173/> to check the page showing the string "ContactBook page".



As shown in the figure, the *ContactBook* page use 4 components: *InputSearch*, *ContactList*, *ContactCard* and *Pagination*. Let's create these components:

1. *InputSearch* component (*src/components/InputSearch.vue*):

```
<script setup>
defineProps({
  modelValue: { type: String, default: " " },
});

const $emit = defineEmits(['submit', 'update:modelValue']);
</script>

<template>
  <div class="input-group">
    <input
      type="text"
      class="form-control px-3"
      placeholder="Nhập thông tin cần tìm"
      :value="modelValue"
      @input="(e) => $emit('update:modelValue', e.target.value)"
      @keyup.enter="$emit('submit')"
    />
    <div class="input-group-append">
      <button
        class="btn btn-outline-secondary"
        type="button"
        @click="$emit('submit')"
      >
        <i class="fas fa-search"></i>
      </button>
    </div>
  </div>
</template>
```

The component has a property named *modelValue*. This property is bound to the input value. An event named *update:modelValue* is emitted when the input value changes. These conditions enable the use of v-model on *InputSearch* to create a two-way binding, i.e., *<InputSearch v-model="..." />*.

2. *ContactList* component (*src/components/ContactList.vue*):

```
<script setup>
defineProps({
  contacts: { type: Array, default: () => [] },
  selectedIndex: { type: Number, default: -1 },
});

const $emit = defineEmits(['update:selectedIndex']);
</script>
```

```

<template>
  <ul class="list-group">
    <li
      class="list-group-item px-3"
      v-for="(contact, index) in contacts"
      :class="{ active: index === selectedIndex }"
      :key="contact.id"
      @click="$emit('update:selectedIndex', index)"
    >
      {{ contact.name }}
    </li>
  </ul>
</template>

```

The component has a property named *activeIndex*. An event named *update:activeIndex* is emitted when a list element is selected. These conditions enable the use of v-model on *ContactList* to create a two-way binding, i.e., `<ContactList v-model:activeIndex="..." />`.

3. *ContactCard* component (*src/components/ContactCard.vue*):

```

<script setup>
defineProps({
  contact: { type: Object, required: true },
});
</script>

<template>
  <div>
    <div class="p-1">
      <strong>Tên:</strong>
      {{ contact.name }}
    </div>
    <div class="p-1">
      <strong>E-mail:</strong>
      {{ contact.email }}
    </div>
    <div class="p-1">
      <strong>Địa chỉ:</strong>
      {{ contact.address }}
    </div>
    <div class="p-1">
      <strong>Điện thoại:</strong>
      {{ contact.phone }}
    </div>
    <div class="p-1">
      <strong>Liên hệ yêu thích:</strong> <i
        v-if="contact.favorite"

```

```

                class="fas fa-check"
            ></i>
        <i
            v-else
            class="fas fa-times"
        ></i>
    </div>
</div>
</template>

```

4. *Pagination* component (src/components/Pagination.vue):

```

<script setup>
import { computed } from 'vue';

const props = defineProps({
  totalPages: {
    type: Number,
    required: true,
  },
  length: {
    type: Number,
    default: 3,
  },
  currentPage: {
    type: Number,
    default: 1,
  },
});

const $emit = defineEmits(['update:currentPage']);

const pages = computed(() => {
  const pages = [];

  const half = Math.floor(props.length / 2); let start =
  props.currentPage - half;

  let end = props.currentPage + half;

  if (start <= 0) {
    start = 1;
    end = props.length;
  }

  if (end > props.totalPages) {
    end = props.totalPages;

    start = end - props.length + 1;
    if (start <= 0) start = 1;
  }

```

```

    }

    for (let i = start; i <= end; i++) {
        pages.push(i);
    }

    return pages;
});

</script>

<template>

    <nav>

        <ul class="pagination">

            <li

                class="page-item"

                :class="{ disabled: currentPage == 1 }"

            >

                <a

                    role="button"

                    class="page-link"

                    @click.prevent="$emit('update:currentPage', currentPage - 1)" >

                        <span>&laquo;</span>

                    </a>

                </li>

                <li

                    v-for="page in pages"

                    :key="page"

                    class="page-item"

                    :class="{ active: currentPage == page }"

                >

                    <a

                        role="button"

                        class="page-link"

                        @click.prevent="$emit('update:currentPage', page)"

                        >{{ page }}</a>

                    >

                </li>

                <li

                    class="page-item"

                    :class="{ disabled: currentPage == totalPages }"

                >

                    <a

                        role="button"

                        class="page-link"

                        @click.prevent="$emit('update:currentPage', currentPage + 1)" >

                            <span>&raquo;</span>

                        </a>

                    </li>

                </ul>

            </nav>

        </template>
    
```

```

        </a>
      </li>
    </ul>
  </nav>
</template>

```

Edit `src/views/ContactBook.vue` to define a page showing a list of contacts:

```

<script setup>
import { ref, computed, onMounted, watch } from 'vue';
import { useRouter } from 'vue-router';
import ContactCard from '@/components/ContactCard.vue';
import InputSearch from '@/components/InputSearch.vue';
import ContactList from '@/components/ContactList.vue';
import Pagination from '@/components/Pagination.vue';
import contactsService from '@/services/contacts.service';

// The full code will be presented below
</script>

<template>
  <div class="page row mb-5">
    <div class="mt-3 col-md-6">
      <h4>
        Danh bạ
        <i class="fas fa-address-book"></i>
      </h4>
      <div class="my-3">
        <InputSearch v-model="searchText" />
      </div>
      <ContactList
        v-if="filteredContacts.length > 0"
        :contacts="filteredContacts"
        v-model:selectedIndex="selectedIndex"
      />
      <p v-else>
        Không có liên hệ nào.
      </p>
      <div class="mt-3 d-flex justify-content-center align-items-center"> <Pagination
        :totalPages="totalPages"
        v-model:currentPage="currentPage"
      />
    </div>
    <div class="mt-3 row justify-content-around align-items-center"> <button
      class="btn btn-sm btn-primary"

```



```

        @click="retrieveContacts(currentPage)"
      >
        <i class="fas fa-redo"></i> Làm mới
      </button>

      <button
        class="btn btn-sm btn-success"
        @click="goToAddContact"
      >
        <i class="fas fa-plus"></i> Thêm mới
      </button>

      <button
        class="btn btn-sm btn-danger"
        @click="onDeleteContacts"
      >
        <i class="fas fa-trash"></i> Xóa tất cả
      </button>
    </div>
  </div>
<div class="mt-3 col-md-6">
  <div v-if="selectedContact">
    <h4>
      Chi tiết Liên hệ
      <i class="fas fa-address-card"></i>
    </h4>
    <ContactCard :contact="selectedContact" />
  </div>
</div>
</div>
</template>

<style scoped>
.page {
  text-align: left;
  max-width: 750px;
}
</style>

```

The *ContactBook* page has the following variables:

totalPages: stores the total pages (in this case, paginating records happens on the server).

currentPage: stores the current page.

contacts: stores a list of contacts on a page. This list is loaded with the data from the server when *ContactBook* is mounted.

selectedIndex: index of the selected contact in the list. *selectedIndex* identifies the contact object passed to *ContactCard* for displaying detailed information.

searchText: stores text entered from the search box.

The *ContactBook* page relies on *contactsService* for accessing data on the server:

```
// src/views/ContactBook.vue

<script setup>
...

const $router = useRouter();
const totalPages = ref(1);
const currentPage = ref(1);

const contacts = ref([]);
const selectedIndex = ref(-1);
const searchText = ref("");

// Map each contact to a string for searching
const searchableContacts = computed(() =>
  contacts.value.map((contact) => {
    const { name, email, address, phone } = contact;
    return [name, email, address, phone].join("");
  })
);

// Contacts filtered by searchText
const filteredContacts = computed(() => {
  if (!searchText.value) return contacts.value;
  return contacts.value.filter((contact, index) =>
    searchableContacts.value[index].includes(searchText.value) );
});

const selectedContact = computed(() => {
  if (selectedIndex.value < 0) return null;
  return filteredContacts.value[selectedIndex.value]; });

// Get contacts for a specific pages and order them by name async function
retrieveContacts(page) {
  try {
    const chunk = await contactsService.getContacts(page); totalPages.value =
    chunk.metadata.lastPage ?? 1;

    contacts.value = chunk.contacts.sort((current, next) =>
      current.name.localeCompare(next.name)
    );

    selectedIndex.value = -1;
  } catch (error) {
    console.log(error);
  }
}
```

```

// Handle delete all contacts event
async function onDeleteContacts() {
  if (confirm("Bạn muốn xóa tất cả Liên hệ?")) {
    try {
      await contactsService.deleteAllContacts();

      totalPages.value = 1;
      currentPage.value = 1;
      contacts.value = [];
      selectedIndex.value = -1;
    } catch (error) {
      console.log(error);
    }
  }
}

function goToAddContact() {
  $router.push({ name: 'contact.add' });
}

// When this component is mounted, load the first page of contacts onMounted(() =>
retrieveContacts(1));

// Whenever searchText changes, reset selectedIndex
watch(searchText, () => (selectedIndex.value = -1));

// When currentPage changes, fetch contacts for currentPage
watchEffect(() => retrieveContacts(currentPage.value));
</script>

```

Start the API server at port 3000 and then start the Vue app: `npm run dev` (if it's not started yet). Open a browser, go to <http://localhost:5173/> and verify that: (1) a list of contacts is shown, (2) the detailed information of a contact is shown when selecting a contact in the list, and (3) the search functionality works correctly. If the database contains no data, add some example data (use a HTTP client to send requests to the API server).

Danh bạ

🔍

Darrell Bergnaum

Ivan Kris Sr.

Olga Bosco

Rachael VonRueden

Sergio Hahn

<<

7

8

9

>>

↻ Làm mới

+ Thêm mới

🗑 Xóa tất cả

After making sure the code works, commit changes to git and upload to GitHub:

```
git add -u
git add .env src/components/ src/router/ src/services/ src/views/
git commit -m "Create a contact listing page"
git push origin master
```

🔗 master ▾

🌿 1 branch

🏷 0 tags

Go to file

Add file ▾

Code ▾

	khoadangtran Create a contact listing page	e4a2666 now	🕒 6 commits
📁 public	Initial commit		2 weeks ago
📁 src	Create a contact listing page		now
📄 .eslintrc.cjs	Initial commit		2 weeks ago
📄 .gitignore	Initial commit		2 weeks ago
📄 README.md	Initial commit		2 weeks ago
📄 index.html	Initial commit		2 weeks ago
📄 jsconfig.json	Create a contact listing page		now
📄 package-lock.json	Create a contact listing page		24 minutes ago
📄 package.json	Create a contact listing page		24 minutes ago
📄 vite.config.js	Create a contact listing page		24 minutes ago

Step 5: Create a 404 error page

Add a route definition matching all the paths (*src/router/index.js*):

```
...  
  
const routes = [  
  ...  
  {  
    path: '/:pathMatch(.*)*',  
    name: 'notfound',  
    component: () => import('@/views/NotFound.vue'),  
  },  
];  
...
```

Create the *NotFound* page in *src/views/NotFound.vue*:


```
<template>  
  <div class="page">  
    <p>  
      Oops, không thể tìm thấy trang. Trở về  
      <router-link to="/">  
        trang chủ.  
      </router-link>  
    </p>  
  </div>  
</template>
```

Open a browser, access to an unknown path and check that the error page is shown. Commit changes and upload to GitHub:

```
git add src/router/index.js src/views/NotFound.vue
```

```
git commit -m "Implement the 404 page"
```

```
git push origin master
```

 khoadangtran	Create a contact listing page	e4a2666 1 hour ago	🕒 6 commits
📁 public	Initial commit		2 weeks ago
📁 src	Create a contact listing page		1 hour ago
📄 .eslintrc.cjs	Initial commit		2 weeks ago
📄 .gitignore	Initial commit		2 weeks ago
📄 README.md	Initial commit		2 weeks ago
📄 index.html	Initial commit		2 weeks ago
📄 jsconfig.json	Create a contact listing page		1 hour ago
📄 package-lock.json	Create a contact listing page		1 hour ago
📄 package.json	Create a contact listing page		1 hour ago
📄 vite.config.js	Create a contact listing page		1 hour ago

Step 6: Create form for adding and updating a contact

The edit page and the add page need a form namely *ContactForm*. *ContactForm* receives a contact object as its property. If the contact object exists on the server (i.e., the id field has a valid value) then *ContactForm* will be in edit mode. On the other hand, it will be in add mode. Only in edit mode, the delete button on the form is shown.

When working with form in Vue, you can use [vee-validate](#) and [yup](#) to easily validate the form data. [vee-validate](#) provides customized form and input components supporting data validation by rules. [yup](#) helps create these validation rules.

Please not that the use of [vee-validate](#) and [yup](#) is not required, you can use normal form and input tags and standard JavaScript code for form validation.

Install [vee-validate](#) and [yup](#): `npm i vee-validate yup`.

```
PS C:\Users\PC\contactbook-frontend> npm i vee-validate yup
up to date, audited 205 packages in 3s
63 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

Create/edit *ContactForm.vue* in *src/components/* as follows:

```
<script setup>
import { ref } from 'vue';
import * as yup from 'yup';
import { Form, Field, ErrorMessage } from 'vee-validate';

const props = defineProps({
  initialContact: { type: Object, required: true },
});

const $emit = defineEmits(['submit:contact', 'delete:contact']);

const contactFormSchema = yup.object().shape({
  name: yup
    .string()
    .required('Tên phải có giá trị.')
    .min(2, 'Tên phải ít nhất 2 ký tự.')
    .max(50, 'Tên có nhiều nhất 50 ký tự.'),
  email: yup
    .string()
    .email('E-mail không đúng.')
    .max(50, 'E-mail tối đa 50 ký tự.'),
  address: yup.string().max(100, 'Địa chỉ tối đa 100 ký tự.'), phone: yup
    .string()
    .matches(
      /(03|05|07|08|09|01[2|6|8|9])+([0-9]{8})\b/g,
```

```

        'Số điện thoại không hợp lệ.'
    ),
  });

const editedContact = ref({ ...props.initialContact });

function submitContact() {
  $emit('submit:contact', editedContact.value);
}

function deleteContact() {
  $emit('delete:contact', editedContact.value.id);
}
</script>

<template>
  <Form
    @submit="submitContact"
    :validation-schema="contactFormSchema"
  >
    <div class="form-group">
      <label for="name">Tên</label>
      <Field
        name="name"
        type="text"
        class="form-control"
        v-model="editedContact.name"
      />
      <ErrorMessage name="name" class="error-feedback" /> </div>
    <div class="form-group">
      <label for="email">E-mail</label>
      <Field
        name="email"
        type="email"
        class="form-control"
        v-model="editedContact.email"
      />
      <ErrorMessage name="email" class="error-feedback" /> </div>
    <div class="form-group">
      <label for="address">Địa chỉ</label>
      <Field
        name="address"
        type="text"
        class="form-control"
        v-model="editedContact.address"
      />
    </div>
  </Form>
</template>

```



```

        <ErrorMessage name="address" class="error-feedback" />
    </div>
    <div class="form-group">
        <label for="phone">Điện thoại</label>
        <Field
            name="phone"
            type="tel"
            class="form-control"
            v-model="editedContact.phone"
        />
        <ErrorMessage name="phone" class="error-feedback" />
    </div>
    <div class="form-group form-check">
        <Field
            name="favorite"
            type="checkbox"
            class="form-check-input"
            v-model="editedContact.favorite"
            :value="1"
            :unchecked-value="0"
        />
        <label for="favorite" class="form-check-label">
            <strong>Liên hệ yêu thích</strong>
        </label>
    </div>
    <div class="form-group">
        <button class="btn btn-primary">Lưu</button>
        <button
            v-if="editedContact.id"
            type="button"
            class="ml-2 btn btn-danger"
            @click="deleteContact"
        >
            Xóa
        </button>
    </div>
</Form>
</template>

<style scoped>
@import '@/assets/form.css';
</style>

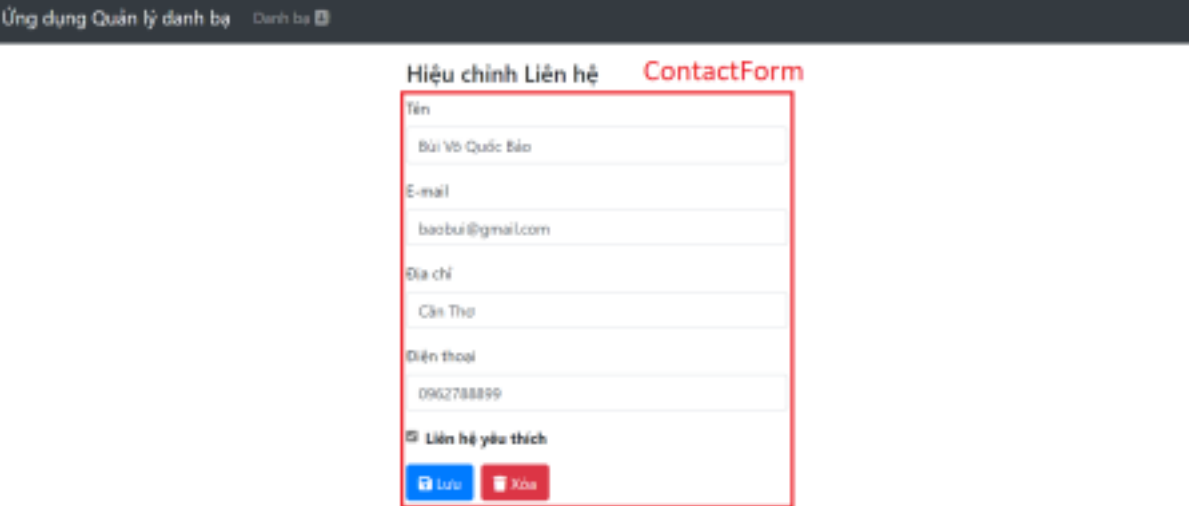
```

In above code, we defined a schema containing validation rules for input data in the form (`validation-schema="contactFormSchema"`). Also note that `ContactForm` can emit two events: `submit:contact` and

delete:contact.

Step 7: Create an edit page

The UI of the page is as follows:



src/views/ContactEdit.vue as follows:

Create

```
<script setup>
import { ref } from 'vue';
import { useRouter, useRoute } from 'vue-router';
import ContactForm from '@/components/ContactForm.vue';
import contactsService from '@/services/contacts.service';

const props = defineProps({
  contactId: { type: String, required: true },
});

const $router = useRouter();
const $route = useRoute();

const contact = ref(null);
const message = ref("");

async function getContact(id) {
  try {
    contact.value = await contactsService.getContact(id);
  } catch (error) {
    console.log(error);
    // Redirect to NotFound page and keep URL intact
    $router.push({
      name: 'notfound',
      params: { pathMatch: $route.path.split('/').slice(1) },
      query: $route.query,
      hash: $route.hash,
    });
  }
}
```

```

    async function onUpdateContact(editedContact) {
      try {
        await contactsService.updateContact(editedContact.id, editedContact); message.value = 'Liên hệ được cập nhật thành công.';
      } catch (error) {
        console.log(error);
      }
    }
  }

  async function onDeleteContact(id) {
    if (confirm('Bạn muốn xóa Liên hệ này?')) {
      try {
        await contactsService.deleteContact(id);
        $router.push({ name: 'contactbook' });
      } catch (error) {
        console.log(error);
      }
    }
  }
}

getContact(props.contactId);
</script>

<template>
  <div v-if="contact" class="page">
    <h4>Hiệu chỉnh Liên hệ</h4>
    <ContactForm
      :initial-contact="contact"
      @submit:contact="onUpdateContact"
      @delete:contact="onDeleteContact"
    />
    <p>{{ message }}</p>
  </div>
</template>

```

The path for accessing to the edit page is `/contacts/:id` with `id` as the id of the contact. Before the page is shown, the `id` property is used to fetch contact data from the server.

Add a route definition for *ContactEdit* in *src/router/index.js*:

```

...
const routes = [
  ...
  {
    path: '/contacts/:id',
    name: 'contact.edit',
    component: () => import('@/views/ContactEdit.vue'),
    props: (route) => ({ contactId: route.params.id })
  }
]

```













```
    },  
  ],  
  ...  
}
```

Add a link to the edit page in ContactBook (*src/views/ContactBook.vue*), just below ContactCard:

```
...  
      <ContactCard :contact="selectedContact" />  
  
      <router-link  
        :to="{  
          name: 'contact.edit',  
          params: { id: selectedContact.id },  
        }"  
      >  
        <span class="mt-2 badge badge-warning">  
          <i class="fas fa-edit"></i> Hiệu chỉnh</span>  
      </router-link>  
    ...
```

Make sure the code works. Commit changes and upload to GitHub:

```
git add -u  
git add src/assets/ src/components/ContactForm.vue src/views/ContactEdit.vue  
git commit -m "Create an edit page"  
git push origin master
```

 khoadangtran	Create a contact listing page	e4a2666 1 hour ago	 6 commits
 public	Initial commit		2 weeks ago
 src	Create a contact listing page		1 hour ago
 .eslintrc.cjs	Initial commit		2 weeks ago
 .gitignore	Initial commit		2 weeks ago
 README.md	Initial commit		2 weeks ago
 index.html	Initial commit		2 weeks ago
 jsconfig.json	Create a contact listing page		1 hour ago
 package-lock.json	Create a contact listing page		1 hour ago
 package.json	Create a contact listing page		1 hour ago
 vite.config.js	Create a contact listing page		1 hour ago

Danh bạ



khoadangtran

khoadangtran

khoadangtran

khoadangtran

khoadangtran

<<

1

2

>>

Chi tiết Liên hệ


Tên: khoadangtran

E-mail: khoahen1508@gmail.com

Địa chỉ: Bạc Liêu


Điện thoại: 0865074636

Liên hệ yêu thích: ✓

 **Hiệu chỉnh**

Step 8: Create an add page

This page is similar to the edit page, so do it on your own:

Ứng dụng Quản lý danh bạ 

Thêm Liên hệ

ContactForm


Tên

E-mail

Địa chỉ

Điện thoại

☐ Liên hệ yêu thích

 Lưu

add page works correctly and then commits changes to GitHub.

Make sure the

Thêm Liên Hệ

Tên

E-mail

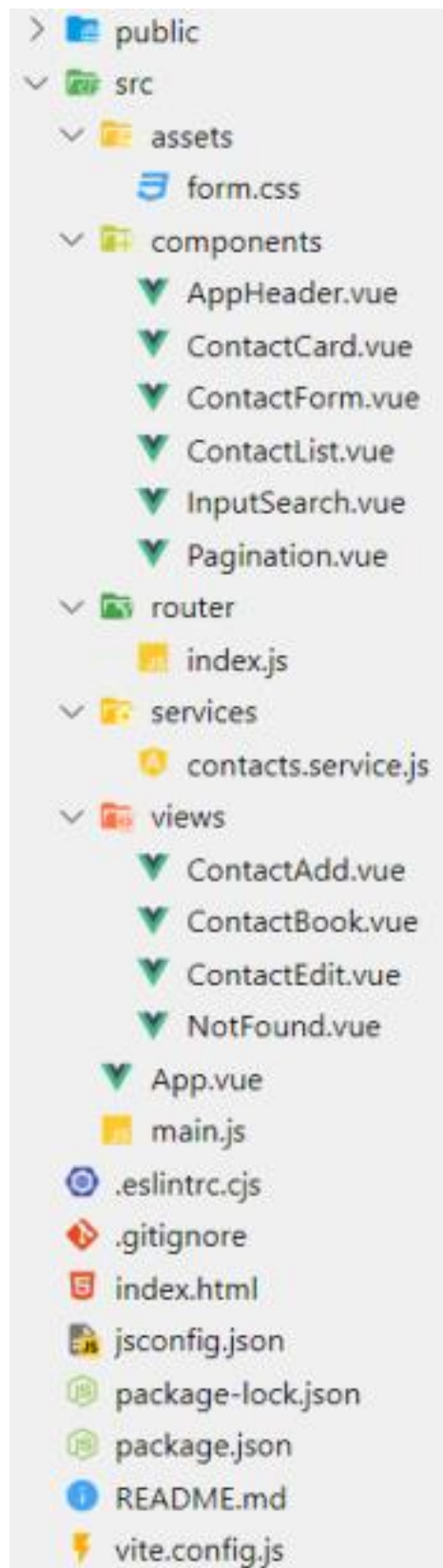
Địa chỉ

Điện thoại

☐ Liên hệ yêu thích

Lưu

The directory structure of the project is as follows:



Step 9: Manage server state with @tanstack/vue-query

Instead of directly call server APIs inside components, update the project to use [@tanstack/vue-query](https://tanstack.com/vue-query) to fetch and modify server state (e.g., contact resources).