

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 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:

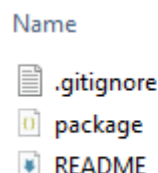
```
git clone <đường-link-đến-repo-GitHub-đã-nhận> contactbook-backend
```

(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
```

 <b>khoadangtran baitap</b>	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

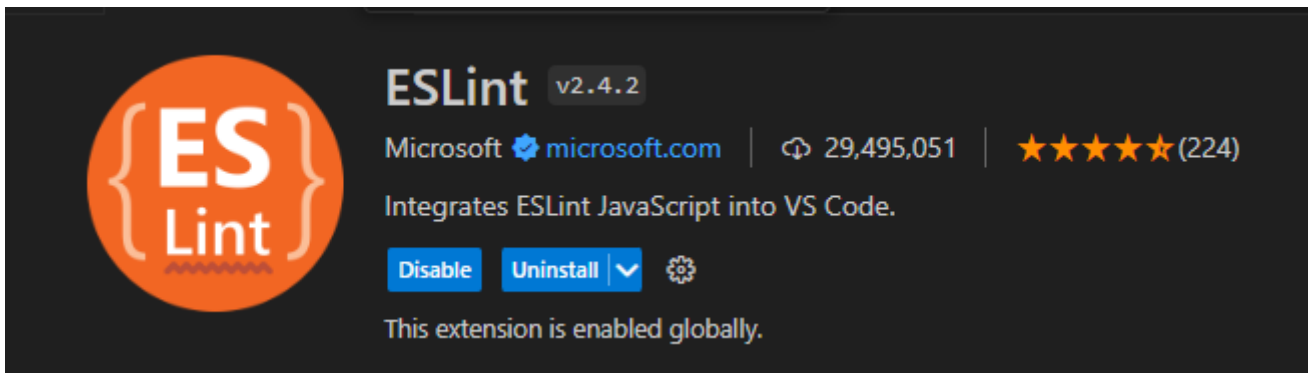
in which, `origin` is the default name given to the remote git repo on GitHub when the repo is cloned.

Verify that the files are uploaded to GitHub.

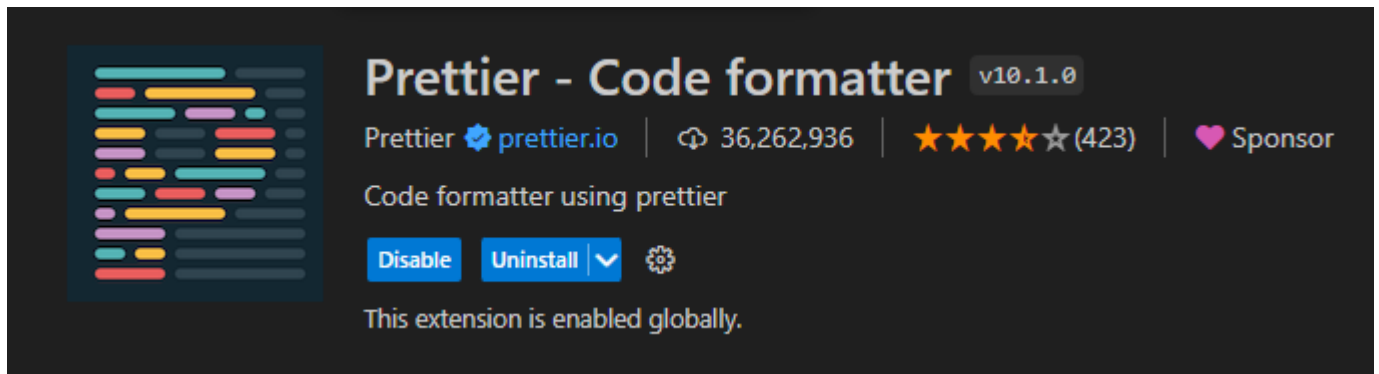
### Step 3: Setup Visual Studio Code and ESLint and Prettier

ESLint is a tool that statically analyzes JavaScript code and helps find and fix problems in the source code.

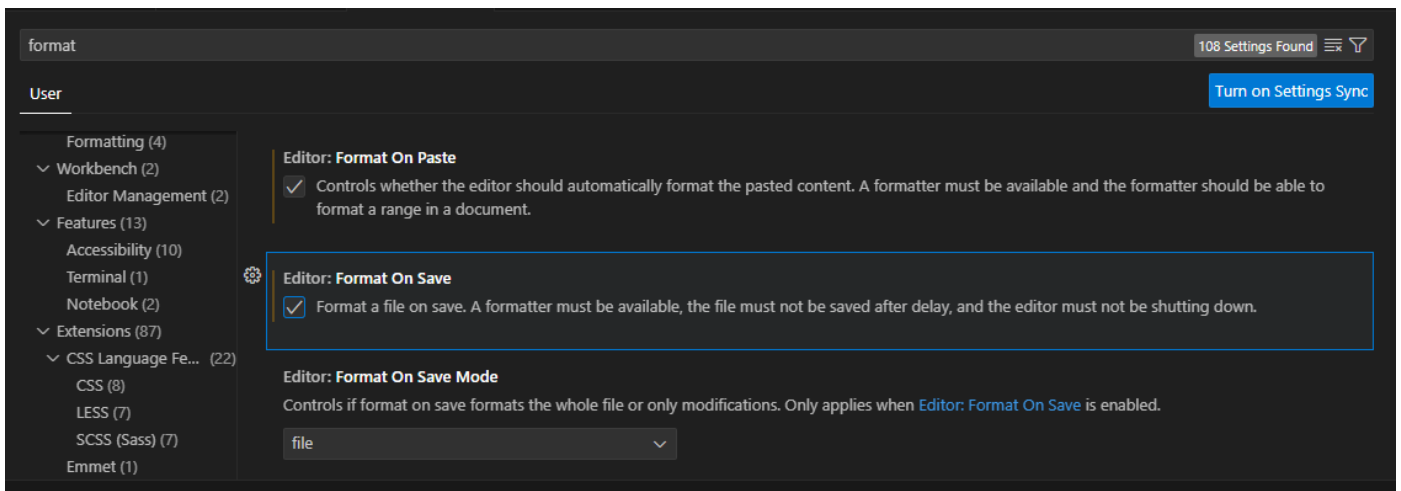
Install [ESLint extension](#) for Visual Studio Code.



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/](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
 .eslinttrc.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 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 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: `npx 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:

