CT449: PHÁT TRIỂN ỨNG DỤNG WEB

Ứng dụng Contactbook - Backend - Phần 2

Chúng ta sẽ xây dựng ứng dụng Quản lý danh bạ theo mô hình Ứng dụng trang đơn (SPA) sử dụng công nghệ Node, Express, MongoDb phía backend (API server) và Vue phía frontend (GUI). Trong buổi thực hành 1 và 2, chúng ta sẽ xây dựng backend cho ứng dụng.

Server của ứng dụng sẽ phải hỗ trợ các yêu cầu sau:

- *POST /api/contacts* : tạo một liên hệ (contact) mới
- *GET /api/contacts* : trả về tất cả các liên hệ trong CSDL
- *DELETE /api/contacts* : xóa tất cả các liên hệ
- *GET /api/contacts/favorite* : trả về các liên hệ được yêu thích
- *GET /api/contacts/<contact-id>* : trả về thông tin một liên hệ dựa trên id
- *PUT /api/contacts/<contact-id>* : cập nhật một liên hệ dựa trên id
- *DELETE /api/contacts/<contact-id>* : xóa một liên hệ dựa trên id
- Yêu cầu đến URL không được định nghĩa : thông báo lỗi 404 "Resource not found"

Một liên hệ (contact) gồm các trường thông tin sau: name (chuỗi), email (chuỗi), address (chuỗi), phone (chuỗi), favorite (true/false). **Định dạng chung cho dữ liệu trao đổi giữa client và server là định dạng JSON**. Mã nguồn được quản lý bởi git và upload lên GitHub.

Hướng dẫn dưới đây sẽ cài đặt các yêu cầu nêu trên. Sinh viên có thể không cần cài đặt giống hướng dẫn, chỉ cần thực hiện đúng các yêu cầu đặt ra ở trên.

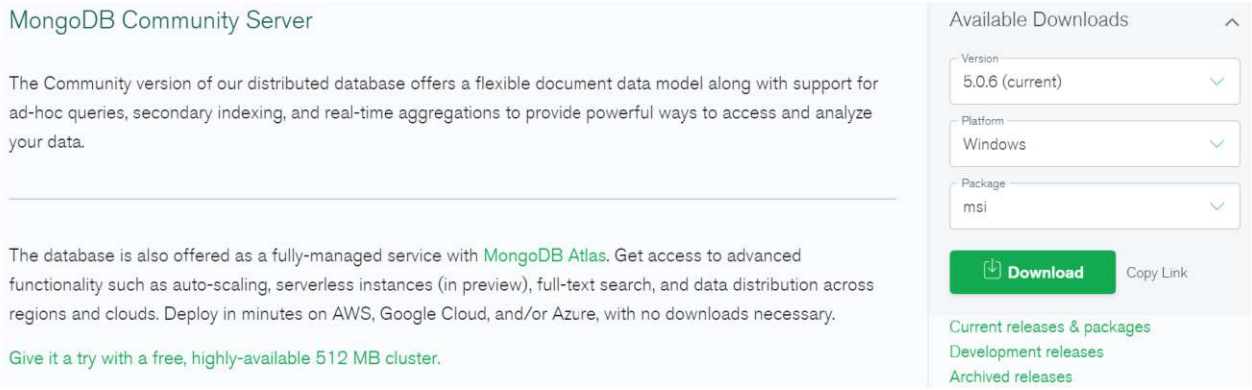
Yêu cầu cho báo cáo thực hành:

- Trên GitHub tạo một Repository để chứa mã nguồn của dự án, tên của Repository theo quy cách sau: MSSV_Hoten_BACKEND_1 (ví dụ: B1234567_Nguyen_Van_A_BACKEND_2)
- Tạo file báo cáo cần nộp là tập tin PDF hoặc tập tin Word theo quy định sau:
    - Tên tập tin theo quy cách: MSSV_Hoten_BACKEND_2 (ví dụ: B1234567_Nguyen_Van_A_BACKEND_2)
    - Nội dung file báo cáo là trình bày các bước thực hiện (chụp hình code và kết quả thực hiện bằng trình duyệt và Postman)

(Tiếp tục từ kết quả thực hành ở Phần 1)

Bước 0: Cài đặt MongoDB

Tải và cài đặt MongoDB Community Server (https://www.mongodb.com/try/download/community):



Bước 1: Cài đặt thư viện mongodb, định nghĩa hàm trợ giúp kết nối và lớp dịch vụ truy xuất cơ sở dữ liệu (CSDL)

Cài đặt thư viện `mongodb` vào dự án: `npm install mongodb`

Trong thư mục *app/config*, hiệu chỉnh (thêm đoạn code) tập tin *index.js*:

```
const config = {
    ...
    db: {
        uri: process.env.MONGODB_URI || "mongodb://127.0.0.1:27017/contactbook"
    }
}
```

Định nghĩa lớp trợ giúp kết nối đến MongoDB: *app/utils/mongodb.util.js*:

```
const { MongoClient } = require("mongodb");

class MongoDB {
    static connect = async (uri) => {
        if (this.client) return this.client;
        this.client = await MongoClient.connect(uri);
        return this.client;
    };
}

module.exports = MongoDB;
```

Thực hiện kết nối đến CSDL MongoDB khi chạy server, thay toàn bộ nội dung tập tin *server.js* bằng đoạn mã dưới đây:

```
const app = require("./app");
const config = require("./app/config");
const MongoDB = require("./app/utils/mongodb.util");

async function startServer() {
    try {
        await MongoDB.connect(config.db.uri);
        console.log("Connected to the database!");

        const PORT = config.app.port;
        app.listen(PORT, () => {
            console.log(`Server is running on port ${PORT}`);
        });
    } catch (error) {
        console.log("Cannot connect to the database!", error);
        process.exit();
    }
}

startServer();
```

Định nghĩa lớp dịch vụ *ContactService* (trong tập tin *app/services/contact.service.js*) chứa các API của thư viện mongodb để thực hiện các thao tác với CSDL MongoDB:

```
const { ObjectId } = require("mongodb");

class ContactService {
 constructor(client) {
  this.Contact = client.db().collection("contacts");
 }
    // Định nghĩa các phương thức truy xuất CSDL sử dụng mongodb API
}

module.exports = ContactService;
```

Bước 2: Cài đặt các handler

Cài đặt handler create

Hiệu chỉnh tập tin *app/controllers/contact.controller.js*:

```
// Create and Save a new Contact
exports.create = async (req, res, next) => {
    if (!req.body?.name) {
        return next(new ApiError(400, "Name can not be empty"));
    }

    try {
        const contactService = new ContactService(MongoDB.client);
        const document = await contactService.create(req.body);
        return res.send(document);
    } catch (error) {
        return next(
            new ApiError(500, "An error occurred while creating the contact")
        );
    }
};
```

Lời gọi *contactService.create()* lưu thông tin đối tượng contact xuống CSDL. Phương thức *create()* được định nghĩa trong lớp ContactService (*app/services/contact.service.js*) như sau:

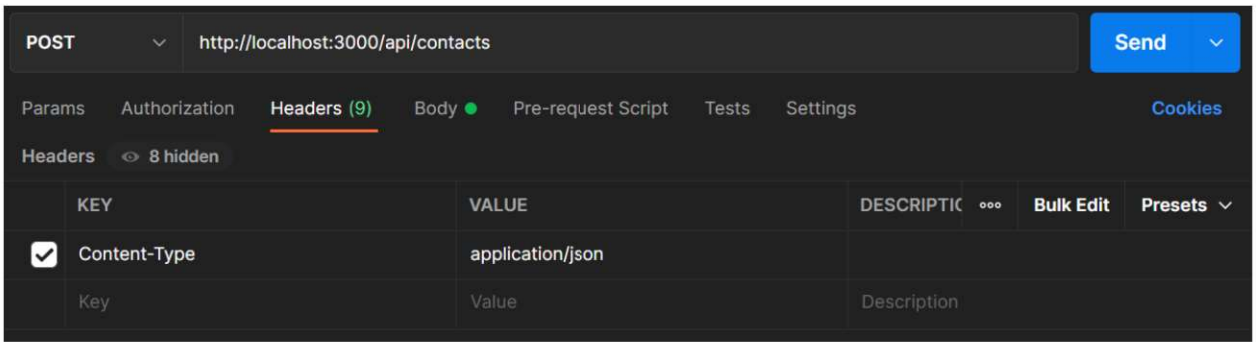
```
...
class ContactService {
 ...
    // Định nghĩa các phương thức truy xuất CSDL sử dụng mongodb API
    extractConactData(payload) {
  const contact = {
   name: payload.name,
   email: payload.email,
   address: payload.address,
   phone: payload.phone,
   favorite: payload.favorite,
  };
  // Remove undefined fields
  Object.keys(contact).forEach(
   (key) => contact[key] === undefined && delete contact[key]
  );
  return contact;
 }

 async create(payload) {
  const contact = this.extractConactData(payload);
  const result = await this.Contact.findOneAndUpdate(
   contact,
   { $set: { favorite: contact.favorite === true } },
   { returnDocument: "after", upsert: true }
  );
  return result;
 }
}
...
```

Nếu có lỗi xảy ra sẽ chuyển cho middleware xử lý lỗi đã định nghĩa trong *app.js* (thông qua lời gọi *next(error)*). Chú ý require các hàm/lớp được sử dụng ở đầu tập tin *app/controllers/contact.controller.js*.

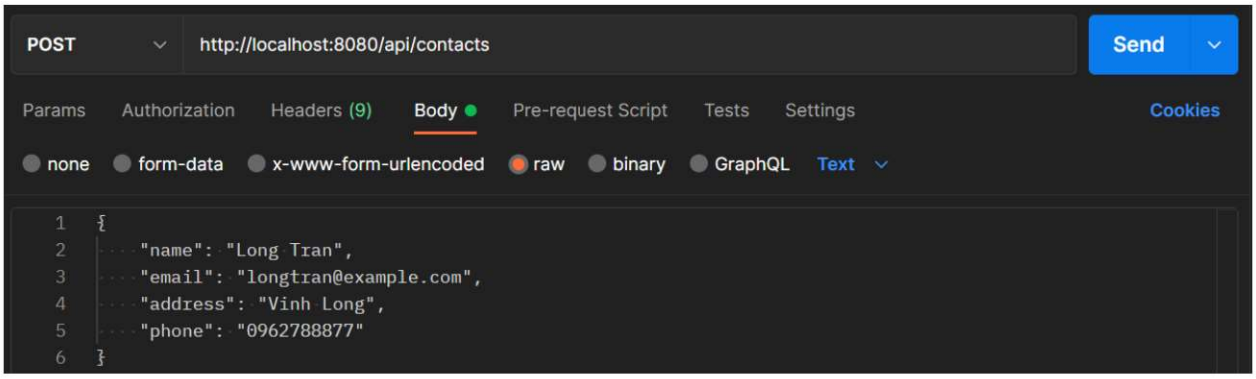
```
const ContactService = require("../services/contact.service");
const MongoDB = require("../utils/mongodb.util");
const ApiError = require("../api-error");
```

Dùng Postman hoặc curl kiểm tra handler hoạt động đúng.

Để gửi dữ liệu JSON về server với Postman hoặc curl, cần đặt Header "Content-Type: application/json" và đặt dữ liệu JSON trong phần Body của yêu cầu, ví dụ:



Hiệu chỉnh Header "Content-Type: application/json"



Đặt dữ liệu JSON trong phần Body

Cài đặt handler findAll trong tập tin *app/controllers/contact.controller.js* như sau:

```
// Retrieve all contacts of a user from the database
exports.findAll = async (req, res, next) => {
    let documents = [];

    try {
        const contactService = new ContactService(MongoDB.client);
        const { name } = req.query;
        if (name) {
            documents = await contactService.findByName(name);
        } else {
            documents = await contactService.find({});
        }
    } catch (error) {
        return next(
            new ApiError(500, "An error occurred while retrieving contacts")
        );
    }

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

contactService.find(condition) và *contactService.findByName(name)* lần lượt tìm kiếm các tài liệu thỏa điều kiện chỉ định trong đối tượng *condition*, theo tên name. Hai phương thức này có thể được định nghĩa như sau trong tập tin *app/services/contact.service.js*:

```
...
class ContactService {
 ...
    async find(filter) {
  const cursor = await this.Contact.find(filter);
  return await cursor.toArray();
 }

 async findByName(name) {
  return await this.find({
      name: { $regex: new RegExp(new RegExp(name)), $options: "i" },
  });
 }
}
...
```

Trong đoạn code của phương thức findByName(name), ta dùng biểu thức chính quy không phân biệt hoa thường (*$option: "i"*) để so khớp tên contact cần tìm kiếm.

Dùng Postman hoặc curl kiểm tra handler hoạt động đúng.

Cài đặt handler findOne trong tập tin *app/controllers/contact.controller.js* như sau:

```
// Find a single contact with an id
exports.findOne = async (req, res, next) => {
    try {
        const contactService = new ContactService(MongoDB.client);
        const document = await contactService.findById(req.params.id);
        if (!document) {
            return next(new ApiError(404, "Contact not found"));
        }
        return res.send(document);
    } catch (error) {
        return next(
            new ApiError(
                500,
                `Error retrieving contact with id=${req.params.id}`
            )
        );
    }
};
```

contactService.findById(id) tìm kiếm tài liệu theo Id. Phương thức *findById(id)* có thể được định nghĩa như sau trong tập tin *app/services/contact.service.js*:

```
...
class ContactService {
 ...
    async findById(id) {
  return await this.Contact.findOne({
   _id: ObjectId.isValid(id) ? new ObjectId(id) : null,
  });
 }
}
...
```

Dùng Postman hoặc curl kiểm tra handler hoạt động đúng.

Cài đặt handler update trong tập tin *app/controllers/contact.controller.js* như sau:

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

    try {
        const contactService = new ContactService(MongoDB.client);
        const document = await contactService.update(req.params.id, req.body);
        if (!document) {
            return next(new ApiError(404, "Contact not found"));
        }
        return res.send({ message: "Contact was updated successfully" });
    } catch (error) {
        return next(
            new ApiError(500, `Error updating contact with id=${req.params.id}`)
        );
    }
};
```

contactService.update(id, document) tìm kiếm tài liệu theo Id và cập nhật tài liệu này với dữ liệu trong đối tượng *document*. Phương thức *update(id, document)* có thể được định nghĩa như sau trong tập tin *app/services/contact.service.js*:

```
...
class ContactService {
 ...
 async update(id, payload) {
  const filter = {
   _id: ObjectId.isValid(id) ? new ObjectId(id) : null,
  };
  const update = this.extractConactData(payload);
  const result = await this.Contact.findOneAndUpdate(
   filter,
   { $set: update },
   { returnDocument: "after" }
  );
  return result.value; //return result;
 }
}
...
```

Dùng Postman hoặc curl kiểm tra handler hoạt động đúng.

Cài đặt handler delete trong tập tin *app/controllers/contact.controller.js* như sau:

```
// Delete a contact with the specified id in the request
exports.delete = async (req, res, next) => {
    try {
        const contactService = new ContactService(MongoDB.client);
        const document = await contactService.delete(req.params.id);
        if (!document) {
            return next(new ApiError(404, "Contact not found"));
        }
        return res.send({ message: "Contact was deleted successfully" });
    } catch (error) {
        return next(
            new ApiError(
                500,
                `Could not delete contact with id=${req.params.id}`
            )
        );
    }
};
```

contactService.delete(id) tìm kiếm tài liệu theo Id và xóa tài liệu này. Phương thức *delete(id)* có thể được định nghĩa như sau trong tập tin *app/services/contact.service.js*:

```
...
class ContactService {
 ...
 async delete(id) {
  const result = await this.Contact.findOneAndDelete({
   _id: ObjectId.isValid(id) ? new ObjectId(id) : null,
  });
  return result;
 }
}
...
```

Dùng Postman hoặc curl kiểm tra handler hoạt động đúng.

Cài đặt handler findAllFavorite trong tập tin *app/controllers/contact.controller.js* như sau:

```
// Find all favorite contacts of a user
exports.findAllFavorite = async (_req, res, next) => {
    try {
        const contactService = new ContactService(MongoDB.client);
        const documents = await contactService.findFavorite();
        return res.send(documents);
    } catch (error) {
        return next(
            new ApiError(
                500,
                "An error occurred while retrieving favorite contacts"
            )
        );
    }
};
```

Phương thức *findFavorite()* trong lớp *ContactService* có thể được định nghĩa như sau có thể được định nghĩa như sau trong tập tin *app/services/contact.service.js*::

```
...
class ContactService {
 ...
 async findFavorite() {
  return await this.find({ favorite: true });
 }
}
...
```

Dùng Postman hoặc curl kiểm tra handler hoạt động đúng.

Cài đặt handler deleteAll:

```
// Delete all contacts of a user from the database
exports.deleteAll = async (_req, res, next) => {
    try {
        const contactService = new ContactService(MongoDB.client);
        const deletedCount = await contactService.deleteAll();
        return res.send({
            message: `${deletedCount} contacts were deleted successfully`,
        });
    } catch (error) {
        return next(
            new ApiError(500, "An error occurred while removing all contacts")
        );
    }
};
```

contactService.deleteMany() xóa tất các các đối tượng trong collection. Phương thức *deleteAll()* có thể được định nghĩa như sau:

```
...
class ContactService {
 ...
 async deleteAll() {
  const result = await this.Contact.deleteMany({});
  return result.deletedCount;
 }
}
...
```

Dùng Postman hoặc curl kiểm tra handler hoạt động đúng.

Sau khi kiểm tra các handler hoạt động đúng, chúng ta có thể lưu các thay đổi vào git:

```
git add -u
git add app/utils app/services
git commit -m "Cai dat cac handler truy xuat CSDL"
```

Đẩy các thay đổi lên GitHub: `git push origin master`.

Cấu trúc thư mục dự án đến thời điểm này sẽ như sau: