

CT449: PHÁT TRIỂN ỨNG DỤNG WEB

Ứng dụng Contactbook - Frontend

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 3 và 4 này chúng ta sẽ xây dựng frontend cho ứng dụng.

Ứng dụng được xây dựng **theo dạng SPA với Vue** bao gồm các chức năng sau:

- Trang liệt kê danh sách các contact, hỗ trợ chức năng tìm kiếm theo tất cả các thông tin (tên, email, điện thoại, địa chỉ).
- Trang hiệu chỉnh thông tin một contact.
- Trang thêm một contact mới.
- Chức năng xóa một contact, xóa tất cả các contact.
- Hiện thị trang lỗi 404 khi truy cập vào các URL không hợp lệ.

Ứng dụng sử dụng HTTP API đã được định nghĩa trong buổi thực hành 1 và 2. 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: File báo cáo cần nộp là file PDF trong đó có ghi thông tin mã sinh viên, họ tên, lớp học phần cùng với hình minh họa tại các bước kiểm tra kết quả thực thi (không cần chụp hình mã nguồn). Cuối file báo cáo ghi đường link đến GitHub mã nguồn của dự án mà bạn tạo.

Buổi 3 và 4 làm chung vào một file báo cáo.

Bước 0: Chuẩn bị môi trường làm việc

- Có thể sử dụng bất kỳ trình soạn thảo code nào. Tuy nhiên, khuyến nghị dùng [Visual Studio Code](#) và cài đặt thêm phần mở rộng [Vue Language Features \(Volar\)](#).
- Để dễ debug lỗi cho ứng dụng viết bằng Vue có thể cài đặt thêm phần mở rộng [Vue.js devtools](#) cho trình duyệt.
- HTTP API server đã phát triển trong buổi 1 và 2 hoạt động đúng.

Bước 1: Tạo ứng dụng Vue

Tạo dự án tên *contactbook-frontend*:

```
npm init vue@3
```

Chọn "No" cho tất cả các tùy chọn:

Vue.js – The Progressive JavaScript Framework

```
✓ Project name: ... contactbook-frontend
✓ Add TypeScript? ... No / Yes
✓ Add JSX Support? ... No / Yes
✓ Add Vue Router for Single Page Application development? ... No / Yes
✓ Add Pinia for state management? ... No / Yes
✓ Add Vitest for Unit Testing? ... No / Yes
✓ Add Cypress for both Unit and End-to-End testing? ... No / Yes
✓ Add ESLint for code quality? ... No / Yes
```

Vào thư mục dự án, XÓA bỏ nội dung thư mục `src/components` và hiệu chỉnh `App.vue` như sau:

```
<script>
export default {

}
</script>

                                npm run dev

<template>
  <h1>Hello, Vue.js!</h1>
</template>

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

Hiệu chỉnh `vite.config.js` để cấu hình port cho ứng dụng:

```
...
export default defineConfig({
  ...
  server: {
    port: 3001,
  },
});
```

Cài đặt các phụ thuộc và chạy kiểm tra:

```
cd contactbook-frontend
npm install
npm run dev
```

Mở trình duyệt bất kỳ, truy cập <http://localhost:3001/> để xem kết quả.

Bước 2: Quản lý mã nguồn dự án với git và GitHub

Khởi tạo git cho dự án:

```
cd contactbook-frontend
git init
git add -A
git commit -m "Khoi tao du an"
```

Tạo dự án trên GitHub và thêm remote vào dự án cục bộ:

```
git remote add origin <github-url>
git push origin master
```

Bước 3: Tạo lớp dịch vụ lấy dữ liệu từ server

Cài đặt gói `axios`: `npm i axios`

Tạo tập tin `src/services/api.service.js` với nội dung sau:

```
import axios from "axios";

const commonConfig = {
  headers: {
    "Content-Type": "application/json",
    Accept: "application/json",
  },
};

export default (baseUrl) => {
  return axios.create({
    baseUrl,
    ...commonConfig,
  });
};
```

Tạo tập tin `src/services/contact.service.js` với nội dung sau:

```
import createApiClient from "../api.service";

class ContactService {
  constructor(baseUrl = "/api/contacts") {
    this.api = createApiClient(baseUrl);
  }
  async getAll() {
    return (await this.api.get("/")).data;
  }
  async create(data) {
    return (await this.api.post("/", data)).data;
  }
  async deleteAll() {
    return (await this.api.delete("/")).data;
  }
  async get(id) {
    return (await this.api.get(`/${id}`)).data;
  }
  async update(id, data) {
    return (await this.api.put(`/${id}`, data)).data;
  }
}
```

```

    }
    async delete(id) {
      return (await this.api.delete(`/${id}`)).data;
    }
  }
}

export default new ContactService();

```

Lớp *ContactService* định nghĩa các phương thức tương tác với dữ liệu phía API server (đã phát triển trong buổi thực hành 1 và 2) bằng cách gọi các lời gọi HTTP tương ứng. Trường hợp các bạn có thay đổi API phía server trong buổi thực hành 1 và 2 khác so với hướng dẫn thì hiệu chỉnh lại phía client (*api.service.js* và *contact.service.js*) cho phù hợp.

Để không cần phải gán cứng hostname/IP của API server trong dự án, chúng ta hiệu chỉnh *vite.config.js*, cấu hình proxy chuyển các yêu cầu có URL chứa */api* xuất phát từ ứng dụng Vue (đang host ở <http://localhost:3001/>) sang địa chỉ của API server (<http://localhost:3000/>):

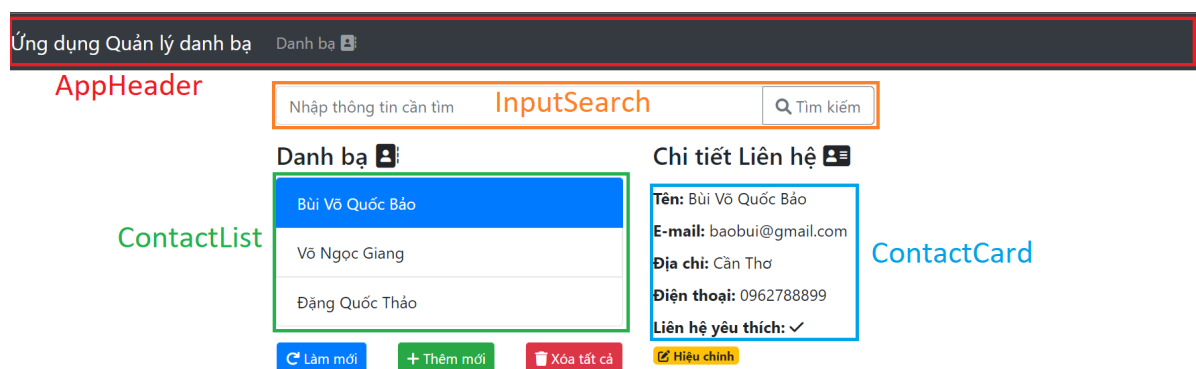
```

export default defineConfig({
  ...
  server: {
    port: 3001,
    proxy: {
      "/api": {
        target: "http://localhost:3000/",
        changeOrigin: true,
      },
    },
  },
});

```

Bước 4: Cài đặt trang hiển thị danh sách các liên hệ

Giao diện được trình bày trong hướng dẫn này có dạng như sau:



Cài đặt thư viện Bootstrap 4 và Font Awesome vào dự án:

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

Hiệu chỉnh tập tin *src/main.js* để import Bootstrap và Font Awesome:

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

Cài đặt `vue-router`: `npm i vue-router@4`, và tạo tập tin `src/router/index.js` với nội dung như sau:

```
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;
```

Đối tượng `import.meta.env` chứa các biến môi trường cho ứng dụng quản lý bởi Vite. `env.BASE_URL` trả về URL cơ sở của ứng dụng trên máy chủ web. Giá trị này được xác định bởi tùy chọn "base" trong file `vite.config.js` (mặc định là "/" - ứng dụng được triển khai ngay tại document root của máy chủ web).

Mở `src/main.js` để thêm router vào ứng dụng:

```
...
import router from "./router";

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

Như thể hiện trên hình, trang `ContactBook` sử dụng 3 component là: `InputSearch`, `ContactList` và `ContactCard`. Lần lượt tạo các thành phần này:

1. Thành phần `InputSearch` (`src/components/InputSearch.vue`):

```
<script>
export default {
  props: {
    modelValue: { type: String, default: "" },
  },
  emits: ["submit", "update:modelValue"],
  methods: {
    updateModelValue(e) {
      this.$emit("update:modelValue", e.target.value);
    },
  },
}
```

```

        submit() {
            this.$emit("submit");
        },
    },
};
</script>

<template>
    <div class="input-group">
        <input
            type="text"
            class="form-control"
            placeholder="Nhập thông tin cần tìm"
            :value="modelValue"
            @input="updateModelValue"
            @keyup.enter="submit"
        />
        <div class="input-group-append">
            <button
                class="btn btn-outline-secondary"
                type="button"
                @click="submit"
            >
                <i class="fas fa-search"></i> Tìm kiếm
            </button>
        </div>
    </div>
</template>

```

Với việc định nghĩa thuộc tính modelValue, liên kết value của input với modelValue và phát sinh sự kiện update:modelValue khi value của input thay đổi cho phép ta sử dụng v-model tạo liên kết hai chiều với *InputSearch*, ví dụ: `<InputSearch v-model="..." />`

2. Thành phần *ContactList* (`src/components/ContactList.vue`):

```

<script>
export default {
    props: {
        contacts: { type: Array, default: [] },
        activeIndex: { type: Number, default: -1 },
    },
    emits: ["update:activeIndex"],
    methods: {
        updateActiveIndex(index) {
            this.$emit("update:activeIndex", index);
        }
    }
};
</script>

<template>
    <ul class="list-group">
        <li
            class="list-group-item"
            v-for="(contact, index) in contacts"

```

```

      :key="contact._id"
      :class="{ active: index === activeIndex }"
      @click="updateActiveIndex(index)"
    >
      {{ contact.name }}
    </li>
  </ul>
</template>

```

Với việc định nghĩa thuộc tính *activeIndex* và phát sinh sự kiện *update:activeIndex* khi một phần tử trong danh sách được nhấp chọn cho phép ta sử dụng v-model tạo liên kết hai chiều với *ContactList*, ví dụ: `<ContactList v-model:activeIndex="..." />`

3. Thành phần *ContactCard* (*src/components/ContactCard.vue*):

```

<script>
export default {
  props: {
    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>

```

Tạo tập tin *src/views/ContactBook.vue* định nghĩa trang hiển thị danh sách các liên hệ:

```

<template>
  <div class="page row">
    <div class="col-md-10">

```

```

        <InputSearch v-model="searchText" />
    </div>
    <div class="mt-3 col-md-6">
        <h4>
            Danh bạ
            <i class="fas fa-address-book"></i>
        </h4>
        <ContactList
            v-if="filteredContactsCount > 0"
            :contacts="filteredContacts"
            v-model:activeIndex="activeIndex"
        />
        <p v-else>Không có liên hệ nào.</p>

        <div class="mt-3 row justify-content-around align-items-center">
            <button class="btn btn-sm btn-primary" @click="refreshList()">
                <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="removeAllContacts"
            >
                <i class="fas fa-trash"></i> Xóa tất cả
            </button>
        </div>
    </div>
    <div class="mt-3 col-md-6">
        <div v-if="activeContact">
            <h4>
                Chi tiết Liên hệ
                <i class="fas fa-address-card"></i>
            </h4>
            <ContactCard :contact="activeContact" />
        </div>
    </div>
</div>
</template>

<script>
import ContactCard from "@/components/ContactCard.vue";
import InputSearch from "@/components/InputSearch.vue";
import ContactList from "@/components/ContactList.vue";
import ContactService from "@/services/contact.service";

export default {
    components: {
        ContactCard,
        InputSearch,
        ContactList,
    },

```



```

    // Đoạn mã xử lý đầy đủ sẽ trình bày bên dưới
  };
</script>

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

```

Trang *ContactBook* cần có các biến dữ liệu sau:

- *contacts*: lưu danh sách các liên hệ để hiển thị. Danh sách này sẽ được lấy về từ server khi *ContactBook* được khởi tạo.
- *activeIndex*: chỉ mục của đối tượng contact được chọn bởi người dùng. *activeIndex* sẽ xác định đối tượng contact được đưa vào *ContactCard* để hiển thị.
- *searchText*: chứa dữ liệu nhập vào từ thanh tìm kiếm.

Trang *ContactBook* dựa vào *ContactService* để lấy và cập nhật dữ liệu về server. Đoạn mã JavaScript xử lý chính của của trang *ContactBook* như sau:

```

// src/views/ContactBook.vue
...
<script>
...
export default {
  ...
  data() {
    return {
      contacts: [],
      activeIndex: -1,
      searchText: "",
    };
  },
  watch: {
    // Giám sát các thay đổi của biến searchText.
    // Bỏ chọn phần tử đang được chọn trong danh sách.
    searchText() {
      this.activeIndex = -1;
    },
  },
  computed: {
    // Chuyển các đối tượng contact thành chuỗi để tiện cho tìm kiếm.
    contactStrings() {
      return this.contacts.map((contact) => {
        const { name, email, address, phone } = contact;
        return [name, email, address, phone].join("");
      });
    },
    // Trả về các contact có chứa thông tin cần tìm kiếm.
    filteredContacts() {
      if (!this.searchText) return this.contacts;
      return this.contacts.filter((_contact, index) =>
        this.contactStrings[index].includes(this.searchText)
      );
    },
  },
};

```

```

    );
  },
  activeContact() {
    if (this.activeIndex < 0) return null;
    return this.filteredContacts[this.activeIndex];
  },
  filteredContactsCount() {
    return this.filteredContacts.length;
  },
},
methods: {
  async retrieveContacts() {
    try {
      this.contacts = await ContactService.getAll();
    } catch (error) {
      console.log(error);
    }
  },

  refreshList() {
    this.retrieveContacts();
    this.activeIndex = -1;
  },

  async removeAllContacts() {
    if (confirm("Bạn muốn xóa tất cả Liên hệ?")) {
      try {
        await ContactService.deleteAll();
        this.refreshList();
      } catch (error) {
        console.log(error);
      }
    }
  },

  goToAddContact() {
    this.$router.push({ name: "contact.add" });
  },
},
mounted() {
  this.refreshList();
},
};
</script>
...

```

Tập tin `src/App.vue` chứa định nghĩa cho component gốc của ứng dụng:

```

<script>
import AppHeader from "@/components/AppHeader.vue";

export default {
  components: {
    AppHeader,
  },

```

```

};
</script>

<template>
  <div id="app">
    <AppHeader />

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

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

```

Thành phần *AppHeader* (*src/components/AppHeader.vue*) định nghĩa thanh điều hướng toàn cục của ứng dụng:

```

<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>

```

Chạy API server (bước thực hành 1 và 2) tại cổng 3000 và sau đó chạy ứng dụng Vue bằng lệnh: `npm run dev` (nếu chưa chạy). Truy cập vào <http://localhost:3001/> để kiểm tra việc hiển thị danh sách các liên hệ, hiển thị thông tin một liên hệ (chọn một liên hệ trong danh sách) và chức năng tìm kiếm liên hệ. Nếu trong CSDL chưa có dữ liệu thì thêm vào vài liên hệ (dùng Postman hoặc curl gửi yêu cầu đến API server).

Sau khi kiểm tra xong thì tiến hành lưu các thay đổi vào git và GitHub:

```

git add -u
git add src/components/ src/router/ src/services/ src/views/
git commit -m "Hiện thị danh sach cac contact lay ve tu server"
git push origin master

```

Bước 5: Tạo trang lỗi 404 không tìm thấy trang

Thêm định nghĩa route khớp với tất cả các URL (*src/router/index.js*):

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

Tạo trang NotFound trong tập tin *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>
```

Mở trình duyệt, truy cập đến một URL không tồn tại để kiểm tra kết quả.

Sau khi kiểm tra xong thì tiến hành lưu các thay đổi vào git và GitHub:

```
git add src/router/index.js src/views/NotFound.vue
git commit -m "Cài đặt trang lỗi 404"
git push origin master
```

Bước 6: Tạo form để thêm hoặc cập nhật liên hệ

Hai trang cập nhật và thêm mới liên hệ cần form nhập liệu. Chúng ta sẽ tạo một component là *ContactForm*. *ContactForm* nhận vào một đối tượng contact làm thuộc tính. Nếu đối tượng contact là tồn tại trên server (thuộc tính id có giá trị) thì *ContactForm* sẽ ở chế độ edit. Ngược lại, nó sẽ ở chế độ add. Chỉ khi ở chế độ edit, nút xóa liên hệ sẽ hiển thị.

Khi làm việc với form trong Vue, chúng ta có thể sử dụng gói `vee-validate` (<https://vee-validate.logaretm.com/v4/>) và `yup` (<https://github.com/jquense/yup>) để kiểm tra tính hợp lệ của các dữ liệu nhập liệu trên form. `vee-validate` cung cấp các control và form tùy biến hỗ trợ cho việc kiểm tra dữ liệu theo các luật (rule). `yup` là thư viện giúp tạo các luật ràng buộc trên dữ liệu.

CHÚ Ý: Việc sử dụng thư viện `vee-validate` và `yup` là không bắt buộc, các bạn hoàn toàn có thể tạo form HTML thông thường.

Cài đặt gói `vee-validate` và `yup`: `npm i vee-validate yup`.

Sao chép tập tin *form.css* đã cho vào thư mục *src/assets/* sau đó tạo tập tin *ContactForm.vue* trong thư mục *src/components/* với nội dung sau:

```
<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="contactLocal.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="contactLocal.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="contactLocal.address"
      />
      <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="contactLocal.phone"
      />
      <ErrorMessage name="phone" class="error-feedback" />
    </div>

    <div class="form-group form-check">
      <input
        name="favorite"
        type="checkbox"
        class="form-check-input"
        v-model="contactLocal.favorite"
      />
      <label for="favorite" class="form-check-label">
        <strong>Liên hệ yêu thích</strong>
      </label>
    </div>
  </Form>
</template>
```

```

    </label>
  </div>

  <div class="form-group">
    <button class="btn btn-primary">Lưu</button>
    <button
      v-if="contactLocal._id"
      type="button"
      class="m1-2 btn btn-danger"
      @click="deleteContact"
    >
      Xóa
    </button>
  </div>
</Form>
</template>

<script>
import * as yup from "yup";
import { Form, Field, ErrorMessage } from "vee-validate";

export default {
  components: {
    Form,
    Field,
    ErrorMessage,
  },
  emits: ["submit:contact", "delete:contact"],
  props: {
    contact: { type: Object, required: true }
  },
  data() {
    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(
          /((09|03|07|08|05)+([0-9]{8})\b)/g,
          "Số điện thoại không hợp lệ."
        ),
    });
    return {
      // Chúng ta sẽ không muốn hiệu chỉnh props, nên tạo biến cục bộ
      // contactLocal để liên kết với các input trên form
      contactLocal: this.contact,
      contactFormSchema,
    };
  },
};

```

```

    };
  },
  methods: {
    submitContact() {
      this.$emit("submit:contact", this.contactLocal);
    },
    deleteContact() {
      this.$emit("delete:contact", this.contactLocal.id);
    },
  },
},
};
</script>

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

```

Trong phương thức `data()` ở trên, chúng ta định nghĩa lược đồ thể hiện các luật ràng buộc cho các trường dữ liệu và đưa các luật này vào form (`:validation-schema="contactFormSchema"`). Cũng chú ý rằng `ContactForm` trên đây có thể phát sinh ra hai sự kiện: `submit:contact` và `delete:contact`.

Bước 7: Tạo trang cập nhật liên hệ

Trang cập nhật liên hệ có dạng sau:

Ứng dụng Quản lý danh bạ
Danh bạ

Hiệu chỉnh Liên hệ
ContactForm

Tên

Bùi Võ Quốc Bảo

E-mail

baobui@gmail.com

Địa chỉ

Cần Thơ

Điện thoại

0962788899

☒ Liên hệ yêu thích

Lưu

Xóa

Tạo tập tin `src/views/ContactEdit.vue` như sau:

```

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

<script>
import ContactForm from "@/components/ContactForm.vue";

```

```

import ContactService from "@/services/contact.service";

export default {
  components: {
    ContactForm,
  },
  props: {
    id: { type: String, required: true },
  },
  data() {
    return {
      contact: null,
      message: "",
    };
  },
  methods: {
    async getContact(id) {
      try {
        this.contact = await ContactService.get(id);
      } catch (error) {
        console.log(error);
        // Chuyển sang trang NotFound đồng thời giữ cho URL không đổi
        this.$router.push({
          name: "notfound",
          params: {
            pathMatch: this.$route.path.split("/").slice(1)
          },
          query: this.$route.query,
          hash: this.$route.hash,
        });
      }
    },

    async updateContact(data) {
      try {
        await ContactService.update(this.contact._id, data);
        this.message = "Liên hệ được cập nhật thành công.";
      } catch (error) {
        console.log(error);
      }
    },

    async deleteContact() {
      if (confirm("Bạn muốn xóa Liên hệ này?")) {
        try {
          await ContactService.delete(this.contact._id);
          this.$router.push({ name: "contactbook" });
        } catch (error) {
          console.log(error);
        }
      }
    },
  },
  created() {
    this.getContact(this.id);
  }
}

```



```

        this.message = "";
    },
};
</script>

```

Trang cập nhật liên hệ được truy cập đến với đường dẫn có dạng `/contacts/:id` với id là id của liên hệ cần cập nhật. Trước khi trang cập nhật được hiển thị, thuộc tính id được dùng để lấy dữ liệu về liên hệ từ server.

Thêm định nghĩa route đến *ContactEdit* vào tập tin *src/router/index.js*:

```

...
const routes = [
  ...
  {
    path: "/contacts/:id",
    name: "contact.edit",
    component: () => import("@/views/ContactEdit.vue"),
    props: true // Truyền các biến trong $route.params vào làm props
  },
];
...

```

Thêm đường liên kết đến trang hiệu chỉnh từ trang ContactBook (*src/views/ContactBook.vue*), ngay dưới thẻ ContactCard:

```

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

```

Kiểm tra chức năng cập nhật liên hệ vừa tạo. Sửa lỗi nếu có và lưu các thay đổi lên git và GitHub:

```

git add -u
git add src/assets/ src/components/ContactForm.vue src/views/ContactEdit.vue
git commit -m "Tao trang cap nhat lien he"
git push origin master

```

Bước 8: Tạo trang thêm mới một liên hệ

Sinh viên tự thực hiện tương tự trang cập nhật liên hệ:

Thêm Liên hệ

ContactForm


Tên

E-mail

Địa chỉ














Điện thoại

☐ Liên hệ yêu thích

 Lưu

Kiểm tra chức năng hoạt động đúng và lưu các thay đổi lên git và GitHub.

Cấu trúc thư mục dự án:

- >  public
- ✓  src
 - >  assets
 - ✓  components
 - ▼ AppHeader.vue
 - ▼ ContactCard.vue
 - ▼ ContactForm.vue
 - ▼ ContactList.vue
 - ▼ InputSearch.vue
 - ✓  router
 - JS index.js
 - ✓  services
 - JS api.service.js
 - JS contact.service.js
 - ✓  views
 - ▼ ContactAdd.vue
 - ▼ ContactBook.vue
 - ▼ ContactEdit.vue
 - ▼ NotFound.vue
 - ▼ App.vue
 - JS main.js
-  .gitignore
-  index.html
-  package-lock.json
-  package.json
-  README.md
-  vite.config.js