Vue.js Стартовий

Axios. Робота з API

Vue.js Стартовий

Introduction

Кінаш Станіслав

Front-end dev

stanislav.kinash

stanislav.kinash

MCID: 9210561

Тема уроку

Axios. Робота з API.

План уроку

1. Що таке Axios?
2. Підключення Axios.
3. Робота з бібліотекою.

Що таке Axios?

API (Application Programming Interface)

Загалом, **API** - це набір протоколів, процедур та інструментів, які дозволяють різним програмним програмам спілкуватися та взаємодіяти один з одним.

API забезпечують структурований спосіб програмного забезпечення для обміну даними та функціональними можливостями. Вони діють як міст між різними програмними системами, що дозволяє їм спілкуватися та працювати разом. **API** можна використовувати для широкого спектру цілей, включаючи обмін даними, управління ресурсами та інтеграцію додатків.

Що таке Axios?

API in Vue

Конкретно у **Vue API** - це набір правил та протоколів, які регулюють те, як розробники можуть взаємодіяти та використовувати основні функції та функціональність Vue

Ці API забезпечують структурований спосіб розробникам маніпулювати та контролювати поведінку компонентів Vue, директив та інших елементів фреймворку.

Що таке Axios?

Існує кілька популярних бібліотек JavaScript для виконання **HTTP-запитів** із веббраузерів або Node.js.

Ось чотири бібліотеки, які найчастіше використовуються:

1. Axios
2. Fetch
3. Superagent
4. Request

Що таке Axios?

Axios

Axios — це легка та проста у використанні бібліотека для створення HTTP-запитів. Він має простий та інтуїтивно зрозумілий API та надає такі функції, як перехоплювачі для обробки даних запитів і відповідей, а також скасування запитів.

Ось приклад використання **Axios** для створення запиту **GET:**

```
axios.get('https://jsonplaceholder.typicode.com/posts/1')
.then(response => {
  console.log(response.data)
})
.catch(error => {
  console.log(error)
})
```

Що таке Axios?

Fetch

Fetch — це рідний (нативний) JavaScript API для виконання HTTP-запитів. Він має простий API, який простий у використанні та розумінні, і забезпечує інтерфейс на основі обіцянок для обробки відповідей.

Ось приклад використання **Fetch** для виконання запиту **GET:**

```
fetch('https://jsonplaceholder.typicode.com/posts/1')
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.log(error))
```

Що таке Axios?

Superagent

Superagent — це легка та гнучка бібліотека для створення HTTP-запитів. Він підтримує багато розширених функцій, як-от потокові відповіді, відстеження прогресу та автоматичні повтори.

Ось приклад використання **Superagent** для створення запиту **GET:**

```
superagent.get('https://jsonplaceholder.typicode.com/posts/1')
.end((error, response) => {
  if (error) {
    console.log(error)
  } else {
    console.log(response.body)
  }
})
```

Що таке Axios?

Request

Request — популярна бібліотека для створення HTTP-запитів у Node.js. Він надає багато розширених функцій, як-от обробка файлів cookie, автентифікація OAuth і завантаження файлів із кількома частинами.

Ось приклад використання **Request** для створення запиту **GET:**

```
const request = require('request')

request('https://jsonplaceholder.typicode.com/posts/1', (error, response, body) => {
  if (error) {
    console.log(error)
  } else {
    console.log(JSON.parse(body))
  }
})
```

Що таке Axios?

Це лише кілька прикладів із багатьох бібліотек, доступних для виконання **HTTP-запитів** у JavaScript.

Вибір бібліотеки залежить від конкретних потреб програми та вподобань розробника.

Підключення Axios.

У Vue 3 **Axios** зазвичай використовується для надсилання та отримання даних із сервера, наприклад, для отримання даних з API або надсилання форми до серверної кінцевої точки.

Ось приклад використання Axios у Vue 3 для отримання даних з API:

```
<script>
import axios from 'axios'

export default {
data() {
  return {
    users: []
  }
},
created() {
  axios.get('https://jsonplaceholder.typicode.com/users')
    .then(response => {
      this.users = response.data
    })
    .catch(error => {
      console.log(error)
    })
}
}
</script>
```

Робота з бібліотекою.

У цьому прикладі ми імпортуємо **Axios** за допомогою оператора **import** у верхній частині нашого компонента. Потім ми визначаємо об'єкт даних із порожнім масивом користувачів.

У створеному хуку нашого компонента ми використовуємо метод axios.get, щоб зробити запит **GET** до URL-адреси **"https://jsonplaceholder.typicode.com/users".**

```
<script>
import axios from 'axios'

export default {
data() {
  return {
    users: []
  }
},
created() {
  axios.get('https://jsonplaceholder.typicode.com/users')
    .then(response => {
      this.users = response.data
    })
    .catch(error => {
      console.log(error)
    })
}
}
</script>
```

Робота з бібліотекою.

Коли запит виконується успішно, викликається метод **then** із даними відповіді, які ми призначаємо масиву користувачів в об’єкті даних нашого компонента.

Якщо запит завершується невдачею, викликається метод **catch** з помилкою.

Робота з бібліотекою.

Потім ми можемо використовувати директиву **v-for** для повторення масиву користувачів і відображення імені кожного користувача в елементі **li.**

```
<template>
<div>
  <h1>Users</h1>
  <ul>
    <li v-for="user in users" :key="user.id">
      {{ user.name }}
    </li>
  </ul>
</div>
</template>
```

Робота з бібліотекою.

Це лише простий приклад, але **Axios** можна використовувати для багатьох інших типів HTTP-запитів, включаючи запити **POST**, **PUT** і **DELETE**.

Крім того, **Axios** надає багато варіантів для налаштування запитів, таких як додавання заголовків, надсилання даних у різних форматах і обробка автентифікації.

Робота з бібліотекою.

Запит HTTP **POST** для API курсу валюти з додатковими параметрами:

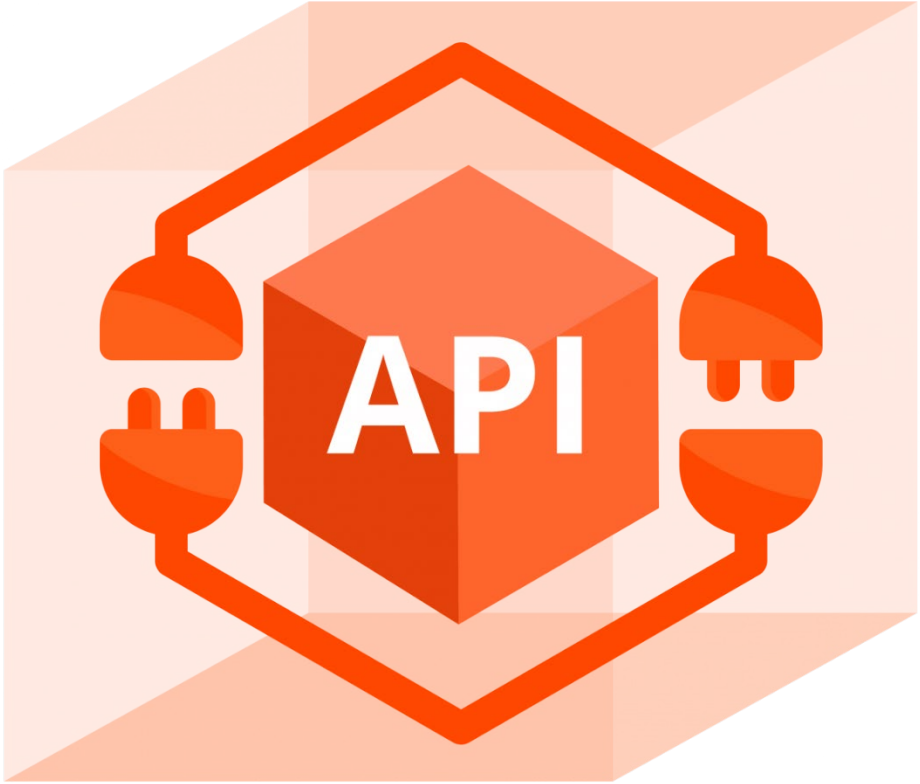
Axios можна використовувати для надсилання запитів **POST** на сервер. Ось приклад того, як зробити запит **POST** до API курсу валюти з деякими додатковими параметрами:

```
import axios from 'axios';

const API_URL = 'https://api.example.com/currency-rates';

const accessToken = 'YOUR_ACCESS_TOKEN';

const data = {
  currency: 'USD',
  rate: 1.2
};

const headers = {
  'Content-Type': 'application/json',
  'Authorization': `Bearer ${accessToken}`
};

axios.post(API_URL, data, { headers })
  .then(response => {
    console.log('Response:', response.data);
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

Робота з бібліотекою.

У цьому прикладі ми надсилаємо запит **POST** на URL-адресу https://api.example.com/currency-rates із корисним навантаженням JSON, що містить дані про валюту та курс.

Ми також включаємо заголовок **Content-Type**, щоб вказати, що ми надсилаємо дані JSON, і заголовок **Authorization** для автентифікації нашого запиту за допомогою маркера носія.



Робота з бібліотекою.

Запит HTTP **PUT** для ідентифікації землетрусів в Utah API з додатковими параметрами:

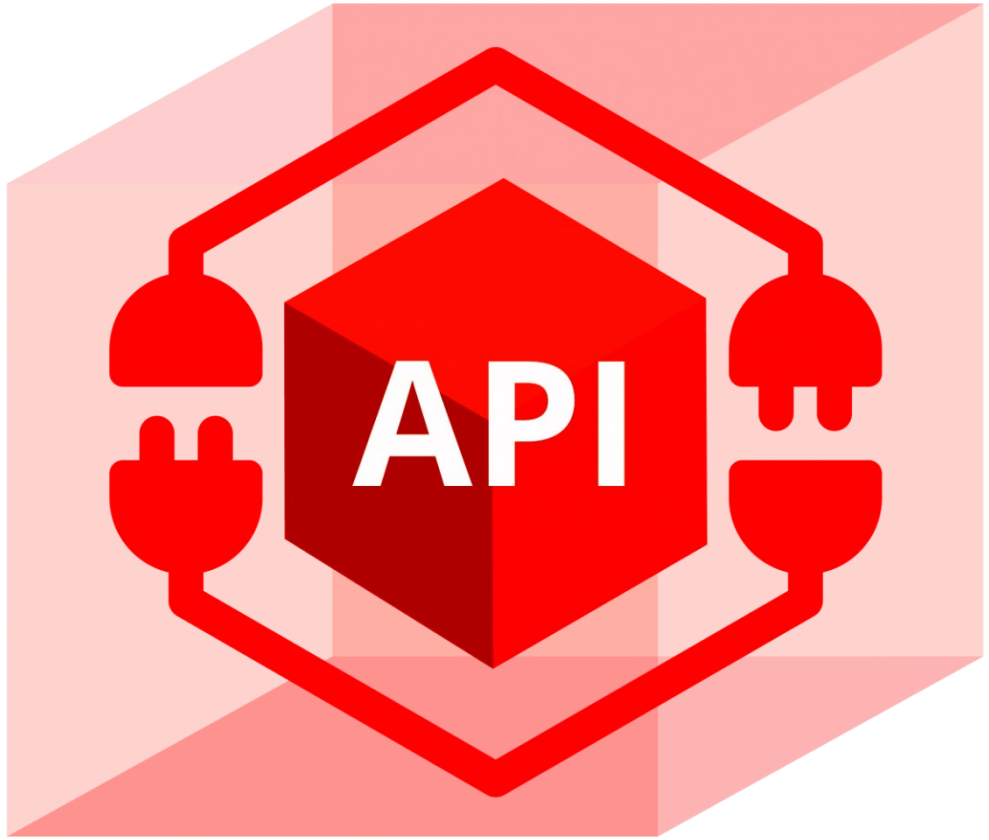
Axios також можна використовувати для надсилання запитів **PUT** на сервер. Ось приклад того, як зробити запит **PUT** для визначення землетрусів в API штату Юта з деякими додатковими параметрами:

```
import axios from 'axios';

const API_URL = 'https://api.example.com/earthquakes/utah';

const accessToken = 'YOUR_ACCESS_TOKEN';

const data = {
  magnitude: 5.0,
  location: 'Salt Lake City'
};

const headers = {
  'Content-Type': 'application/json',
  'Authorization': `Bearer ${accessToken}`
};

axios.put(API_URL, data, { headers })
  .then(response => {
    console.log('Response:', response.data);
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

Робота з бібліотекою.

У цьому прикладі ми надсилаємо запит **PUT** на URL-адресу https://api.example.com/earthquakes/utah із корисним навантаженням JSON, що містить дані про магнітуду та розташування.

Ми також включаємо заголовок **Content-Type**, щоб вказати, що ми надсилаємо дані **JSON**, і заголовок **Authorization** для автентифікації нашого запиту за допомогою маркера носія.



Робота з бібліотекою.

Запит HTTP **DELETE** для Google Maps API з додатковими параметрами:

Axios також можна використовувати для надсилання запитів **DELETE** на сервер. Ось приклад того, як зробити запит DELETE до API Карт Google із додатковими параметрами:

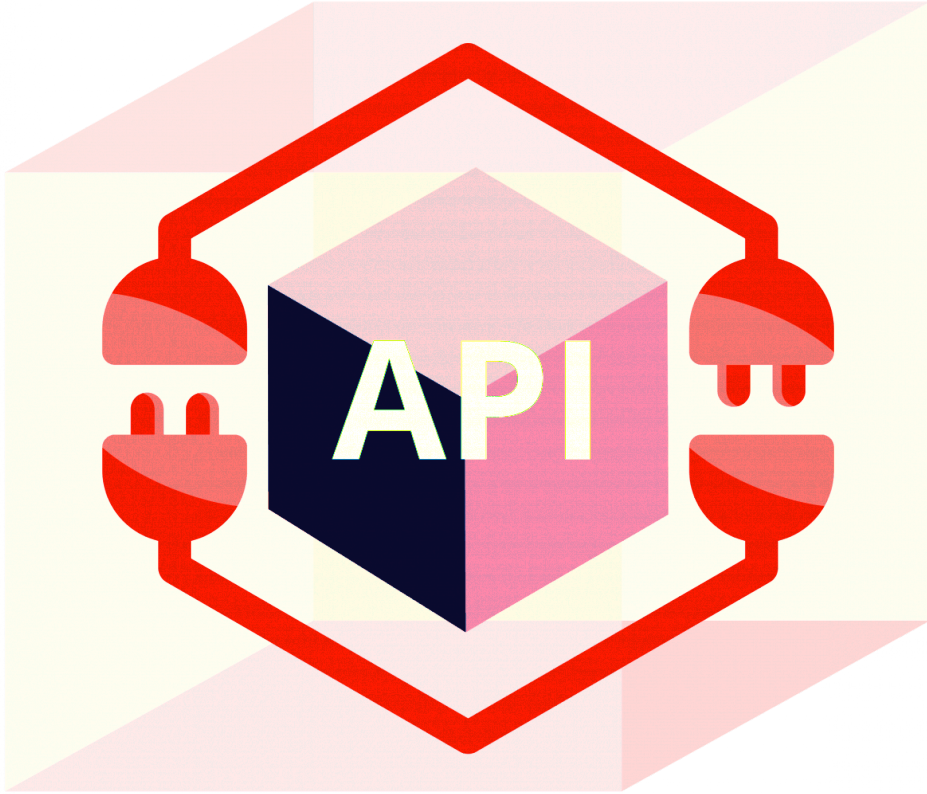
```
import axios from 'axios';

const accessToken = 'YOUR_ACCESS_TOKEN';

const API_URL = 'https://maps.googleapis.com/maps/api/place/delete/json';

const data = {
  place_id: 'ChIJrTLr-GyuEmsRBfy61i59si0'
};

const headers = {
  'Content-Type': 'application/json',
  'Authorization': `Bearer ${accessToken}`
};

axios.delete(API_URL, { data, headers })
  .then(response => {
    console.log('Response:', response.data);
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

Робота з бібліотекою.

У цьому прикладі ми надсилаємо запит **DELETE** на https://maps.googleapis.com/maps/api/place/delete/json URL-адресу з корисним навантаженням **JSON**, що містить дані place_id.

Ми також включаємо заголовок **Content-Type**, щоб вказати, що ми надсилаємо дані JSON, і заголовок **Authorization** для автентифікації нашого запиту за допомогою маркера носія.

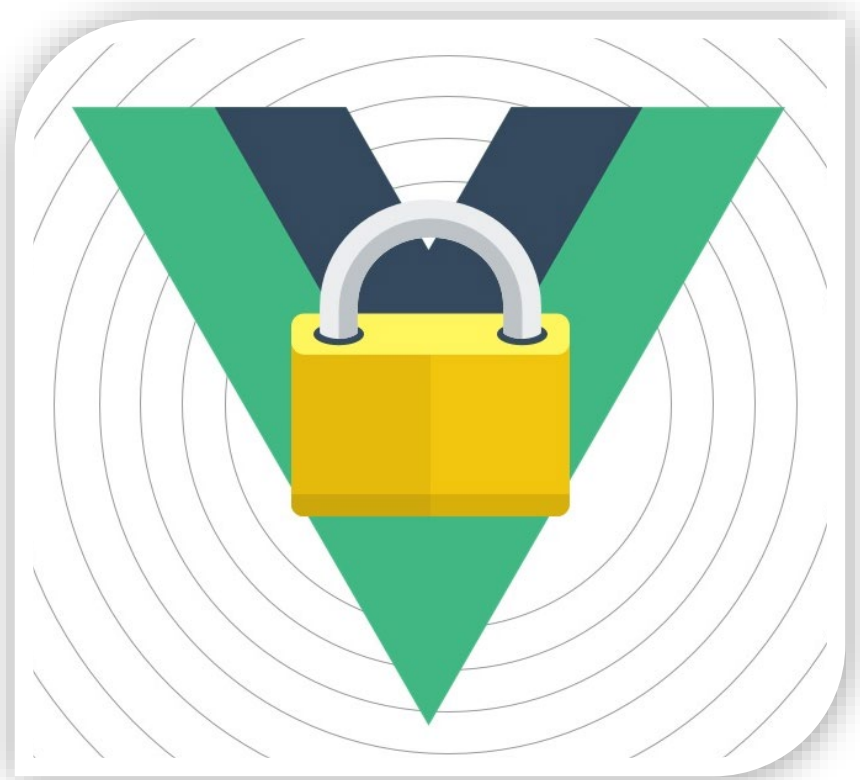
Зауважте, що корисне навантаження передається в запит за допомогою опції даних, яка є обов'язковою умовою для виконання запиту DELETE із корисним навантаженням за допомогою Axios.



Робота з бібліотекою.

У HTTP автентифікація носія **(Bearer authentication)** — це тип автентифікації на основі маркера доступу, у якому маркер доступу передається від клієнта до сервера в заголовку HTTP, зокрема в заголовку авторизації.

Маркери носія **(Bearer tokens)** часто використовуються для автентифікації API, і вони зазвичай видаються сервером авторизації в результаті процесу автентифікації користувача. Маркер доступу зазвичай має обмежений термін служби, і його потрібно періодично оновлювати.



Робота з бібліотекою.

Формат токена **Bearer** — це просто слово «Bearer», після якого йде пробіл, а потім маркер доступу.

Наприклад, Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4f6wkPOdMeJ

Робота з бібліотекою.

У Vue маркери носія часто використовуються для автентифікації запитів HTTP до кінцевих точок API, які потребують автентифікації. Під час надсилання запиту до кінцевої точки API, яка вимагає автентифікації, ви можете включити маркер носія в заголовок авторизації запиту, використовуючи формат, описаний вище. Це дозволяє серверу ідентифікувати користувача, пов'язаного з маркером, і авторизувати запитану дію.

Наприклад, припустімо, що у вас є кінцева точка API, яка вимагає автентифікації за допомогою токенів носія. Ви можете включити маркер у HTTP-запит таким чином:

```
import axios from 'axios'

const accessToken = 'your_access_token_here'

axios.post('https://api.example.com/create-user', {
  username: 'johndoe',
  password: 'password123'
}, {
  headers: {
    'Authorization': `Bearer ${accessToken}`,
    'Content-Type': 'application/json'
  }
})
```

Робота з бібліотекою.

У цьому прикладі метод axios.post використовується для надсилання запиту **POST** до кінцевої точки https://api.example.com/create-user.

Корисне навантаження запиту включає ім'я користувача та пароль для нового створюваного користувача. Об'єкт headers використовується для включення заголовка авторизації, який включає маркер Bearer, а також заголовка Content-Type, який вказує, що корисне навантаження є у форматі JSON. Це дозволить серверу автентифікувати запит і створити нового користувача в системі.

```
import axios from 'axios'

const accessToken = 'your_access_token_here'

axios.post('https://api.example.com/create-user', {
  username: 'johndoe',
  password: 'password123'
}, {
  headers: {
    'Authorization': `Bearer ${accessToken}`,
    'Content-Type': 'application/json'
  }
})
```

Робота з бібліотекою.

Давайте створимо форму з полями, додамо валідацію полів та надсилатимемо дані на сервер за допомогою Axios.

Давайте створимо новий компонент Vue під назвою ContactForm.vue і визначимо поля форми в його шаблоні:

Цей компонент визначає форму з чотирма полями введення та кнопкою надсилання. Поля форми прив’язані до об’єкта форми за допомогою директив v-model, а дані форми перевіряються за допомогою простих умов if.

```
<template>
<div>
  <form @submit.prevent="submitForm">
    <div>
      <label for="name">Name:</label>
      <input type="text" id="name" v-model.trim="form.name" required>
      <div v-if="formErrors.name" class="error">{{ formErrors.name }}</div>
    </div>
    <div>
      <label for="email">Email:</label>
      <input type="email" id="email" v-model.trim="form.email" required>
      <div v-if="formErrors.email" class="error">{{ formErrors.email }}</div>
    </div>
    <div>
      <label for="phone">Phone:</label>
      <input type="tel" id="phone" v-model.trim="form.phone" required>
      <div v-if="formErrors.phone" class="error">{{ formErrors.phone }}</div>
    </div>
    <div>
      <label for="message">Message:</label>
      <textarea id="message" v-model.trim="form.message" required></textarea>
      <div v-if="formErrors.message" class="error">{{ formErrors.message }}</div>
    </div>
    <div>
      <button type="submit">Submit</button>
    </div>
  </form>
</div>
</template>
```

Робота з бібліотекою.

Створимо об’єкт **form** з полями **name**, **email**, **phone** і **message**.

Також створимо додатковий об’єкт **formErrors** для відловлення помилок при заповненні форми - валідація.

```
<script>
import axios from 'axios';

export default {
  data() {
    return {
      form: {
        name: '',
        email: '',
        phone: '',
        message: ''
      },
      formErrors: {}
    };
  },
```

Робота з бібліотекою.

Створимо асинхронну функцію **submitForm()**
та напишемо умови-валідатори для наших полів форми.

Для перевірки коректності введеної електронної пошти та мобільного телефону використаємо потужний інструмент JavaScript – **Regex**.

```
methods: {
  async submitForm() {
    this.formErrors = {};

    const { name, email, phone, message } = this.form;

    // validate form fields
    if (!name) {
      this.formErrors.name = 'Name is required';
    }

    if (!email) {
      this.formErrors.email = 'Email is required';
    } else if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)) {
      this.formErrors.email = 'Invalid email format';
    }

    if (!phone) {
      this.formErrors.phone = 'Phone is required';
    } else if (!/^\d{10}$/.test(phone)) {
      this.formErrors.phone = 'Invalid phone format';
    }

    if (!message) {
      this.formErrors.message = 'Message is required';
    }
```

Робота з бібліотекою.

Якщо форма дійсна, запит HTTP POST надсилається на сервер за допомогою Axios.

Зауважте, що метод **submitForm** позначено як асинхронний, оскільки запит HTTP є асинхронною операцією. Блок **try/catch** використовується для обробки будь-яких помилок, які можуть виникнути під час запиту. Заголовок **Content-Type** має значення application/json, щоб вказати, що тіло запиту міститься у форматі JSON.

Добавимо також стилі для полів, що не пройшли валідацію.

```
      if (Object.keys(this.formErrors).length === 0) {
        try {
          // send form data to server
          const response = await axios.post('/api/contact', this.form, {
            headers: {
              'Content-Type': 'application/json'
            }
          });

          console.log(response);
        } catch (error) {
          console.error(error);
        }
      }
    }
  }
};
</script>

<style>
.error {
  color: red;
  font-size: 12px;
  margin-top: 4px;
}
</style>
```

TestProvider Pearson CyberBionic systematics Microsoft Partner Windows Azure EBA

Перевірка знань

TestProvider.com

TestProvider – це online-сервіс перевірки знань з інформаційних технологій. За його допомогою ви можете оцінити свій рівень та виявити слабкі місця. Він буде корисним як у процесі вивчення технології, так і для загальної оцінки знань IT-спеціаліста.

Перевірте, як ви засвоїли даний матеріал на TestProvider.com

Успішне проходження фінального тестування дозволить отримати відповідний Сертифікат.

Vue.js Стартовий

Дякую за увагу! До нових зустрічей!

Кінаш Станіслав
Front-end dev

MCID: 9210561