API Documentation

#web_project

References from: 2024-01-09

API Name: Dating App

Version: 1.0

Date: 2024-01-09

Description:

This API provides endpoints for managing user accounts, matching users, and sending messages.

Tên API

- app.post('/login')
- app.post('/signup')
- app.get('/user')
- app.put('/addmatch')
- app.get('/users')
- app.get('/gendered-users')
- app.put('/user')
- app.get('/messages')
- app.post('/message')

Phương thức

- POST
- POST
- GET
- PUT
- GET
- GET

- PUT
- GET
- POST

Tham số

Tên	Kiểu dữ liệu	Bắt buộc	Mô tả
email	string	Có	Email của người dùng
password	string	Có	Mật khẩu của người dùng
userId	string	Có	ID của người dùng
name	string	Có	Tên của người dùng
gender_identity	string	Có	Giới tính của người dùng
location	object	Có	Vị trí của người dùng
userId	string	Có	ID của người dùng
correspondingUserId	string	Có	ID của người dùng đối diện
content	string	Có	Nội dung của tin nhắn

Đầu ra

Tên	Kiểu dữ liệu	Mô tả
token	string	Mã thông báo JWT
user	object	Thông tin của người dùng
matches	array	Danh sách người dùng đã được ghép đôi
users	array	Danh sách người dùng
gendered_users	array	Danh sách người dùng có giới tính tương ứng
user	object	Thông tin của người dùng đã được cập nhật
messages	array	Danh sách tin nhắn
message	object	Thông tin của tin nhắn đã được thêm

Mô tả

1. **app.post('/login')**
    - Endpoint POST “/login” để xử lí yêu cầu đăng nhập của người dùng. Nó tìm kiếm người dùng trong collection users dựa trên email được cung cấp, nếu tồn

tại thì kiểm tra mật khẩu bằng cách so sánh mật khẩu đã nhập và mật khẩu đã được mã hóa. Nếu thông tin đăng nhập hợp lệ, mã sẽ tạo một mã thông báo JWT để xác thực người dùng đã đăng nhập thành công. Mã thông báo này sẽ hết hạn sau 1 ngày.

- Code:

```
// Log in to the Database
app.post('/login', async (req, res) => {
        const client = new MongoClient(uri)
        const {email, password} = req.body

        try {
                await client.connect()
                const database = client.db('appdata')
                const users = database.collection('users')

                const user = await users.findOne({email})

                const correctPassword = await bcrypt.compare(password,
user.hashed_password)

                if (user && correctPassword) {
                        const token = jwt.sign(user, email, {
                                expiresIn: 60 * 24
                        })
                        res.status(201).json({token, userId:
user.user_id})
                }

                res.status(400).json('Invalid Credentials')

        } catch (err) {
                console.log(err)
        } finally {
                await client.close()
        }
})
```

2. **app.post('/signup')**
    - Endpoint POST “/signup” để xử lý yêu cầu đăng ký người dùng mới khi được cung cấp email và password. Khi email chưa được đăng kí sẽ tạo ra 1 `user_id` ngẫu nhiên và mật khẩu được mã hóa bằng `bcrypt`. Sau đó insert vào collection 'users' trong cơ sở dữ liệu. Một mã thông báo JWT được tạo để xác thực người dùng đã đăng ký thành công.
    - Code:

```
// Sign up to the Database
app.post('/signup', async (req, res) => {
        const client = new MongoClient(uri)
        const {email, password} = req.body

        const generatedUserId = uuidv4()
        const hashedPassword = await bcrypt.hash(password, 10)

        try {
                await client.connect()
                const database = client.db('appdata')
                const users = database.collection('users')

                const existingUser = await users.findOne({email})

                if (existingUser) {
                        return res.status(409).send('User already
exists. Please login')
                }

                const sanitizedEmail = email.toLowerCase()

                const data = {
                        user_id: generatedUserId,
                        email: sanitizedEmail,
                        hashed_password: hashedPassword
                }

                const insertedUser = await users.insertOne(data)
```

```
                const token = jwt.sign(insertedUser, sanitizedEmail, {
                        expiresIn: 60 * 24
                })
                res.status(201).json({token, userId: generatedUserId})

        } catch (err) {
                console.log(err)
        } finally {
                await client.close()
        }
})
```

3. **app.get('/user')**
    - Endpoint GET “/user” dùng để lấy thông tin của một người dùng dựa trên `user_id` từ collection users. Sau khi thiết lập kết nối và truy vấn cơ sở dữ liệu, nó tìm kiếm người dùng theo `user_id` và trả về thông tin người dùng đó trong phản hồi.
    - Code:

```
// Get individual user
app.get('/user', async (req, res) => {
    const client = new MongoClient(uri)
    const userId = req.query.userId

    try {
        await client.connect()
        const database = client.db('appdata')
        const users = database.collection('users')

        const query = {user_id: userId}
        const user = await users.findOne(query)
        res.send(user)

    } finally {
        await client.close()
    }
})
```

4. **app.put('/addmatch')**
    - Một endpoint PUT “/addmatch” để cập nhật thông tin của người dùng sau khi người dùng đó được ghép đôi với một người dùng khác. API này sẽ thêm một bản ghi vào mảng matches của người dùng. Bản ghi này sẽ bao gồm thông tin về người dùng được ghép đôi.
    - Code:

```
// Update User with a match
app.put('/addmatch', async (req, res) => {
    const client = new MongoClient(uri)
    const {userId, matchedUserId} = req.body

    try {
        await client.connect()
        const database = client.db('appdata')
        const users = database.collection('users')

        const query = {user_id: userId}
        const updateDocument = {
            $push: {matches: {user_id: matchedUserId}}
        }
        const user = await users.updateOne(query, updateDocument)
        res.send(user)
    } finally {
        await client.close()
    }
})
```

5. **app.get('/users')**
    - Một endpoint GET “/users” lấy danh sách `user_id` người dùng từ tham số userIds trong yêu cầu và trả về thông tin của tất cả người dùng có `user_id` trong danh sách đó.
    - Code:

```
// Get all Users by userIds in the Database
app.get('/users', async (req, res) => {
    const client = new MongoClient(uri)
```

```
    const userIds = JSON.parse(req.query.userIds)

    try {
        await client.connect()
        const database = client.db('appdata')
        const users = database.collection('users')

        const pipeline =
            [
                {
                    '$match': {
                        'user_id': {
                            '$in': userIds
                        }
                    }
                }
            ]

        const foundUsers = await users.aggregate(pipeline).toArray()

        res.json(foundUsers)

    } finally {
        await client.close()
    }
})
```

6. **app.get('/gendered-users')**
    - Một endpoint GET “/gendered-users” để trả về danh sách tất cả người dùng có giới tính tương ứng với tham số gender. Đầu tiên, endpoint lấy tham số gender từ yêu cầu. Sau đó, nó sử dụng tham số này để xây dựng một truy vấn tìm kiếm. Truy vấn này sẽ tìm tất cả các tài liệu trong collection users có trường gender_identity trùng khớp với giá trị gender. Kết quả truy vấn được trả về dưới dạng một mảng. Mảng này được chuyển đổi thành một đối tượng JSON và trả về trong phản hồi của yêu cầu.
    - Code:

```
// Get all the Gendered Users in the Database
app.get('/gendered-users', async (req, res) => {
    const client = new MongoClient(uri)
    const gender = req.query.gender
    const smoking = req.query.smoking
    const drinking = req.query.drinking

    try {
        await client.connect()
        const database = client.db('appdata')
        const users = database.collection('users')
        const query = {
            gender_identity: {$eq: gender},
            smoking_identity: {$eq: smoking},
            drinking_identity: {$eq: drinking}
        }
        const foundUsers = await users.find(query).toArray()
        res.json(foundUsers)

    } finally {
        await client.close()
    }
})
```

7. **app.put('/user')**
    - Endpoint PUT “/user” được thực hiện để cập nhật thông tin người dùng. Truy vấn để xác định người dùng cần cập nhật dựa trên `user_id` lấy từ formData. Hàm getLocationByIP() được gọi để lấy thông tin vị trí dựa trên địa chỉ IP của người dùng. Kết quả trả về chứa thông tin vị trí gồm kinh độ và vĩ độ. Một đối tượng updateDocument được tạo chứa thông tin cần cập nhật của người dùng. Sau đó, phương thức updateOne() được sử dụng để cập nhật thông tin người dùng tương ứng với truy vấn và đối tượng updateDocument. Kết quả trả về chứa thông tin về người dùng đã được cập nhật. Cuối cùng, thông tin người dùng đã được cập nhật được trả về dưới dạng một đối tượng JSON qua hàm res.json().
    - Code:

```
// Update a User in the Database
app.put('/user', async (req, res) => {
    const client = new MongoClient(uri)
    const formData = req.body.formData

    try {
        await client.connect()
        const database = client.db('appdata')
        const users = database.collection('users')

        const query = {user_id: formData.user_id}

        const ipInfoResponse = await getLocationByIP();
        const { loc } = ipInfoResponse;
        const [latitude, longitude] = loc.split(',');

        // Add location data to the updateDocument object
        const updateDocument = {
            $set: {
                first_name: formData.first_name,
                last_name: formData.last_name,
                dob_day: formData.dob_day,
                dob_month: formData.dob_month,
                dob_year: formData.dob_year,
                show_gender: formData.show_gender,
                gender_identity: formData.gender_identity,
                gender_interest: formData.gender_interest,
                smoking_identity: formData.smoking_identity,
                smoking_interest: formData.smoking_interest,
                url: formData.url,
                hobbies: formData.hobbies,
                about: formData.about,
                matches: formData.matches,
                latitude: parseFloat(latitude),
                longitude: parseFloat(longitude),
            },
        };
```

```
        const insertedUser = await users.updateOne(query,
updateDocument)

        res.json(insertedUser)

    } finally {
        await client.close()
    }
})

// Get users' location by IP Address
async function getLocationByIP() {
    return new Promise((resolve, reject) => {
        http.get('http://ipinfo.io/json', (response) => {
            let data = '';

            response.on('data', (chunk) => {
                data += chunk;
            });

            response.on('end', () => {
                try {
                    const result = JSON.parse(data);
                    resolve(result);
                } catch (error) {
                    reject(error);
                }
            });
        }).on('error', (error) => {
            reject(error);
        });
    });
}
```

8. **app.get('/messages')**
    - Endpoint GET “/message” nhận yêu cầu GET để lấy danh sách tin nhắn. Đầu tiên, endpoint lấy hai tham số `userId` và `correspondingUserId` từ yêu cầu. Sau đó, nó sử dụng hai tham số này để xây dựng một truy vấn tìm kiếm. Truy

vấn này tìm kiếm các tin nhắn trong collection messages có `from_userId` là `userId` và `to_userId` là `correspondingUserId`. Sau đó gửi danh sách tin nhắn đã tìm thấy trong phản hồi của yêu cầu.

- Code:

```
app.get('/messages', async (req, res) => {
    const {userId, correspondingUserId} = req.query
    const client = new MongoClient(uri)

    try {
        await client.connect()
        const database = client.db('appdata')
        const messages = database.collection('messages')

        const query = {
            from_userId: userId, to_userId: correspondingUserId
        }
        const foundMessages = await messages.find(query).toArray()
        res.send(foundMessages)
    } finally {
        await client.close()
    }
})
```

9. **app.post('/message')**
    - Endpoint POST “/message” để thêm một tin nhắn mới vào database dựa trên nội dung được gửi trong body của yêu cầu. Đầu tiên, endpoint lấy nội dung tin nhắn từ body của yêu cầu. Sau đó, nó sử dụng nội dung này để tạo một đối tượng tin nhắn. Đối tượng tin nhắn này được thêm vào collection messages trong database bằng phương thức insertOne(). Phương thức này trả về một đối tượng chứa thông tin về tin nhắn đã được thêm (bao gồm mã ID của tin nhắn) và thông tin này được gửi trong phản hồi của yêu cầu.
    - Code:

```
// Add a Message to our Database
app.post('/message', async (req, res) => {
    const client = new MongoClient(uri)
```

```
    const message = req.body.message

    try {
        await client.connect()
        const database = client.db('appdata')
        const messages = database.collection('messages')

        const insertedMessage = await messages.insertOne(message)
        res.send(insertedMessage)
    } finally {
        await client.close()
    }
})
```