**[회원가입 기능]**

- 클라이언트는 axios의 POST 요청을 한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

🡪 클라이언트로부터 회원가입 폼에 입력된 값들이 저장된 JSON 객체를 넘겨받아, 이메일이 중복됐는지 검사하고, 중복되지 않았으면 패스워드를 해시화한 뒤 DB 에 계정을 생성함

**<코드 분석>**

**‘axios.post’**

* **‘axios’ 라이브러리**

: 브라우저와 Node.js에서 사용할 수 있는 HTTP 클라이언트 라이브러리로 HTTP 요청을 쉽게 보낼 수 있도록 도와준다.

‘npm install axios’로 설치한다.

* **‘axios.post(url, data):’**

🡪 **‘post’ 메서드**: HTTP POST 요청을 보냄

🡪 **‘url’**: 요청을 보낼 서버의 URL. (위의 코드에서는 "URL/api/signup"이다.)

🡪 **‘data’**: 서버로 보낼 데이터. (위의 코드에서는 ‘sendData’라는 변수에 담긴 데이터를 보낸다.)

**‘await’**

* ‘await’ 키워드

🡪 ‘await’: 비동기 함수(‘async’ 함수) 내에서 사용된다.

🡪 ‘await’는 프로미스가 해결 때까지 함수 실행을 일시 정지함.

🡪 ‘axios.post’는 프로미스를 반환하므로, ‘await’를 사용하여 요청이 완료될 때까지 기다린다.

**‘sendData’**

* ‘sendData 변수

: 서버로 보낼 데이터를 담고 있다.

🡪 위의 코드에서는 회원가입 정보를 담고 있다.

ex)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**‘const response’**

* ‘**response' 변수**

: 요청이 완료되면 ‘**response**’ 변수는 서버로부터 응답을 받는다.

코드를 더 쉽게 이해하기 위해서 추가적인 설명을 하겠습니다.

**1. 비동기 프로그래밍**

: 로그인 기능에서도 계속 나왔던 내용으로, 비동기 작업을 처리하기 위해 ‘async’와 ‘await’를 사용해서 코드가 블로킹 되지 않고, 다른 작업을 계속할 수 있도록 한다.

**2. HTTP 요청**

: HTTP 프로토콜은 클라이언트와 서버가 통신하는 방식으로, POST 요청은 데이터를 서버에 보내기 위해 사용된다.

**3. HTTP 상태 코드**

: 로그인 기능에서도 나온 내용으로, 서버는 요청에 대한 응답으로 상태 코드를 반환한다.

🡪 200: 성공

🡪 401: 잘못된 요청

🡪 500: 서버 오류 등이 있다.

🡪 ‘response.status’를 통해 상태 코드를 확인할 수 있다.

**4. 응답 데이터**

: 서버로부터 받은 응답 데이터는 ‘response.data’에 저장된다.

ex) 서버가 JSON 형식으로 데이터를 반환하면, 이 데이터를 사용하여 클라이언트에서 추가 처리를 할 수 있다.

**<요약>**

코드 분석을 요약하자면,

* ‘axios.post’는 서버에 HTTP POST 요청을 보낸다,
* ‘await’는 요청이 완료될 때까지 기다린다.
* ‘sendData’는 서버로 보낼 데이터를 포함한다.
* ‘response’는 서버의 응답을 포함한다.
* \*비동기 프로그래밍과 HTTP 요청/응답의 기본 개념을 이해하는 것이 중요하다\*

- 서버는 express의 POST 응답을 한다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

이 코드는 **사용자가 회원가입을 요청할 때 서버에서 이를 처리하는 로직을 구현한 것**으로, Express.js, Mongoose, bcrypt 등을 사용하여 회원가입 요청을 처리한다

<코드 분석>

‘**app.post('/api/signup', async (req, res) => { ... });**’

* ‘**app.post**’

🡪 Express.js에서 POST 요청을 처리하기 위한 메서드

🡪 첫 번째 인수: 요청 경로 ('/api/signup')

🡪 두 번째 인수: 요청을 처리하는 비동기 함수.

‘**const { name, gender, email, password } = req.body;**

: 요청 본문(‘req.body’)에서 name, gender, email, password를 추출한다.

🡪 클라이언트가 전송한 회원가입 데이터를 받는다.

‘**try { ... } catch (error) { ... }**

: ‘try’ 블록 내에서 발생하는 에러를 ‘catch’ 블록에서 처리한다.

🡪 서버에서 발생할 수 있는 예외 상황을 처리하여 안정적인 응답을 보장한다.

‘**const user = await Users.findOne({ email });**’

* ‘**Users.findOne**’

🡪 MongoDB에서 특정 이메일을 가진 사용자를 찾기 위해 Mongoose의 findOne 메서드를 사용한다.

🡪 이메일이 일치하는 사용자를 데이터베이스에서 찾는다.

🡪 ‘await’ 키워드는 이 비동기 작업이 완료될 때까지 기다린다.

‘**if (user) { return res.status(409).json({status:409, message: '중복된 이메일'}); }**’

: 사용자가 이미 존재하면 **409 상태 코드**와 함께 응답을 반환한다. (사용자 중복 여부 확인)

🡪 409 상태 코드: Conflict (충돌)를 의미.

🡪 json 메서드를 사용하여 JSON 형식의 응답을 보낸다.

‘**const hashedPassword = await bcrypt.hash(password, 10);**’

* ‘**bcrypt.hash**’

: 비밀번호를 해시하기 위해 ‘bcrypt’ 라이브러리의 ‘hash’ 메서드를 사용

🡪 ‘10’은 해시 작업의 강도를 나타내는 ‘saltRounds’이다.

🡪 ‘awailt’ 키워드는 이 비동기 작업이 완료될 때까지 기다린다.

‘**await Users.create({ name, gender, email, password: hashedPassword });**’

* ‘**Users.create**’

: 새로운 사용자를 데이터베이스에 생성하기 위해 Mongoose의 create 메서드를 사용한다.

🡪 ‘name’, ‘gender’, ‘email’, ‘password’: ‘hashedPassword’ 등의 데이터를 저장한다.

🡪 ‘await’ 키워드는 이 비동기 작업이 완료될 때까지 기다린다.

‘**res.status(201).send('계정 생성 성공');**’

: 계정 생성이 성공하면 **201 상태 코드**와 함께 응답을 반환한다.

🡪 201 상태 코드: Created (생성됨)를 의미.

🡪 ‘send’ 메서드를 사용하여 성공 메시지를 보낸다.

‘**catch (error) { res.status(500).send('계정 생성 중 오류 발생'); }**’

: ‘try’ 블록 내에서 발생한 모든 에러를 처리하여 **500 상태 코드**와 함께 응답을 반환한다. (서버 에러 처리)

🡪 500 상태 코드: “Internal Server Error” (서버 내부 오류)를 의미한다.

🡪 ‘send’ 메서드를 사용하여 오류 메시지를 보낸다.

이러한 코드를 이해하기 위해 필요한 지식은 Expres.js, 비동기프로그래밍, MongoDB와 Mongoose, bcrypt 라이브러리, HTTP 상태 코드로 이에 관해 설명하겠습니다.

**- Express.js**

: Express.js는 Node.js에서 서버를 설정하고 HTTP 요청을 처리하기 위한 웹 애플리케이션 프레임워크이다.

🡪 이 프레임워크를 사용하면 라우팅, 미들웨어 설정, 응답 처리 등을 쉽게 할 수 있다.

(설치)



(기본 설정)

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

**- 비동기 프로그래밍: ‘async’와 ‘await’**

: 이 내용은 로그인 기능에서부터 계속해서 정리했기에 간단히 정리하겠습니다.

: 비동기 프로그래밍은 특히 I/O 작업에서 중요한데, 이는 작업이 완료될 때까지 다른 작업을 차단하지 않도록 한다. async 함수는 항상 Promise를 반환하며, await 키워드는 Promise가 해결될 때까지 함수의 실행을 일시 중지한다.

**- MongoDB와 Mongoose**

* **MongoDB**: 문서 지향 데이터베이스
* **Mongoose**: MongoDB와 상호 작용하기 위한 ODM(Object Data Modeling) 라이브러리로, 스키마를 정의하고 데이터를 검증하는 등의 기능을 제공한다.

🡪 **스키마**: MongoDB 컬렉션의 문서 구조를 정의하는 객체.

스키마를 통해 문서가 가져야 할 필드와 그 필드의 데이터 타입, 유효성 검사,

기본값, 인덱스 등의 세부 사항을 지정할 수 있다.

🡪 데이터의 일관성을 유지하고, 데이터베이스와 상호작용할 때 코드의 안정

성을 높일 수 있다.

(설치)



(Mongoose 설정)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

(사용자 스키마 정의)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**- bcrypt 라이브러리**

: bcrypt는 비밀번호 해싱과 검증을 위한 라이브러리이다.

🡪 안전한 비밀번호 저장을 위해 많이 사용됩니다. 해싱된 비밀번호는 원래 비밀번호로 되돌릴 수 없다.

(설치)



(비밀번호 해싱)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

(비밀번호 검증)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**- HTTP 상태 코드**

: 클라이언트와 서버 간의 통신에서 요청이 어떻게 처리되었는지를 나타낸다.

* 2xx: 성공

🡪 200 OK: 요청이 성공적으로 처리됨

🡪 201 Created: 요청이 성공적으로 처리되었으며, 새로운 리소스가 생성됨.

* 4xx: 클라이언트 오류

🡪 400 Bad Request: 클라이언트의 요청이 잘못되었음.

🡪 401 Unauthorized: 인증이 필요함.

🡪 403 Forbidden: 서버가 요청을 이해했지만 승인을 거부함.

🡪 404 Not Found: 요청한 리소스를 찾을 수 없음.

🡪 409 Conflict: 요청이 서버의 현재 상태와 충돌함.

* 5xx: 서버 오류

🡪 500 Internal Server Error; 서버에서 예기치 않은 오류가 발생함.

🡪 503 Service Unavailable: 서버가 일시적으로 요청을 처리할 수 없음.

**<요약>**

이 코드는 회원가입 요청을 처리하여, 데이터베이스에서 중복 이메일을 확인하고, 비밀번호를 해시하여 저장하며, 새 사용자를 생성한다.

각 단계에서 적절한 예외 처리를 통해 안정적인 응답을 제공하고, 이를 통해 사용자 데이터를 안전하게 처리하고 서버의 상태를 클라이언트에게 명확히 전달할 수 있다.

**- 라우트 정의**

: ‘app.post('/api/signup', async (req, res) => { ... })’를 사용하여 ‘/api/signup’ 경로에 대한 POST 요청을 처리한다.

**- 요청 본문 파싱**

: ‘const { name, gender, email, password } = req.body;’를 통해 클라이언트가 보낸 데이터를 추출한다.

**- 사용자 중복 확인**

* ‘const user = await Users.findOne({ email });’를 사용하여 데이터베이스에 이미 해당 이메일이 있는지 확인한다.
* 이메일이 이미 존재하면 ‘res.status(409).json({status: 409, message: '중복된 이메일'});’로 응답한다.

**- 비밀번호 해싱**

: ‘const hashedPassword = await bcrypt.hash(password, 10);’를 사용하여 비밀번호를 해시합니다.

**- 새 사용자 생성**

: ‘await Users.create({ name, gender, email, password: hashedPassword });’를 사용하여 데이터베이스에 새 사용자를 생성한다.

**- 성공 응답**

: ‘res.status(201).send('계정 생성 성공');’로 사용자 생성 성공을 응답한다.

**- 오류 처리**

: ‘catch’ 블록에서 발생한 예외를 처리하여 ‘res.status(500).send('계정 생성 중 오류 발생');’로 응답한다.