**\*서버와 클라이언트 역할\***

**- 클라이언트 -**

* **사용자 인터페이스 제공** (웹 브라우저를 통해 사용자와 상호작용)
* **서버와 통신** (서버에 HTTP 요청을 보내고 응답을 받음)

🡪 필요한 데이터를 서버로부터 가져오거나, 서버에 데이터를 보냄

* **데이터 처리** (서버로부터 받은 데이터를 처리하고 이를 사용자에게 표시)

🡪 데이터를 가공하거나 필터링하여 적절한 형식으로 보여줌

* **로컬 저장소 관리** (쿠키, 로컬 스토리지, 세션 스토리지를 사용하여 데이터를 로컬에 저장하고 관리)

🡪 사용자 세션을 유지하거나, 사용자 설정을 저장

**- 서버 -**

* **HTTP 요청 처리** (클라이언트로부터 들어오는 HTTP 요청을 처리)

🡪 요청은 GET, POST, PUT, DELETE 등의 메서드를 사용할 수 있다.

* **데이터베이스 관리** (데이터베이스와 상호작용하여 데이터를 저장, 조회, 수정, 삭제함)

🡪 데이터베이스 연결을 관리하고, 데이터베이스 쿼리를 실행한다.

* **비즈니스 로직 실행** (클라이언트의 요청을 처리하기 위해 비즈니스 로직을 실행한다)

🡪 이 로직은 데이터 검증, 계산, 특정 조건에 따른 처리 등을 포함한다.

* **응답 생성** (클라이언트에게 보낼 적절한 응답을 생성)

🡪 응답은 HTML 페이지, JSON 데이터, 파일 다운로드 등의 형태가 될 수 있다.

* **보안관리** (사용자 인증 및 권한 부여를 처리하고, 데이터를 보호)

🡪 클라이언트가 권한이 없는 데이터에 접근하는 것을 방지한다.

**[로그인 기능]**

- 클라이언트는 axios의 POST 요청을 한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**axios 라이브러리를 사용하여 비동기적으로 POST요청을 보내는 JavaScript 코드**

좀 더 자세히 코드를 설명하자면,

**‘response’**는 ‘axios.post’의 결과를 담은 변수로, ‘axios.post’가 ‘Promise’를 반환하고, ‘await’는 이 ‘Promise’가 해결될 때까지 기다린 후 결과를 ‘response’에 할당한다.

-> **‘Promise’**: JavaScript에서 비동기 작업의 완료 또는 실패를 처리할 수 있는 객체로, 미래에 어떤 값이 반환되거나 실패할 것을 약속한다는 의미에서 이름이 붙여졌다.

**‘await’** 키워드는 ‘axios.post가 완료될 때까지 기다리며(‘Promise’가 해결될 때까지 코드 실행을 멈추고), 비동기 작업이 끝나면(‘Promise’가 해결되면) 그 결과 값을 반환한다.

-> 비동기 함수(‘async’ 함수) 내에서만 사용할 수 있다.

-> ‘await’을 사용할 수 있는 것은 ‘Promise’를 반환하는 함수이기 때문이다.

**‘axios.post’**는 주어진 URL로 POST 요청을 보내며, ‘sendData’를 요청 본문으로 전송한다.

**‘"URL/api/signin"’**은 요청을 보낼 대상 서버의 URL로, 실제로는 이 URL을 실제 API 엔드 포인트로 대체해야한다.

-> **API 엔드포인트(API Endpoint)**: 클라이언트가 요청을 보낼 수 있는 특정 URL

--> 이는 서버가 API 요청을 받아들이고 처리할 수 있는 URL 경로를 뜻한다.

--> **엔드포인트**는 일반적으로 웹 애플리케이션이나 서비스의 특정 기능이나 리소스에 접근할 수 있게 한다.

**‘sendData’**는 서버로 전송할 데이터 객체로, 로그인 정보를 포함할 수 있다.

이 코드에서 사용된 문법은 ‘axios’라이브러리, 비동기 함수와 ‘await’, ‘axios.post’함수가 있다.

**- ‘axios’ 라이브러리**

: ‘npm install axios’로 설치할 수 있으며, 브라우저와 Node.js에서 모두 작동하는 HTTP 클라이언트 라이브러리로, 간단하게 HTTP 요청을 보낼 수 있다.

또한 ‘Promise’기반의 HTTP 클라이언트로, 모든 HTTP 요청 메서드(‘get’, ‘post’, ‘put’, ‘delete’ 등)는 ‘Promise’를 반환한다.

**- 비동기 함수와 ‘await’**

: ‘async’/’await’는 비동기 코드를 동기 코드처럼 작성할 수 있게 해주는 문법이다.

‘async’ 키워드를 함수 앞에 붙여서 그 함수를 비동기로 만들 수 있다.

‘**await**’ 키워드는 ‘Promise’가 완료될 때까지 기다리며. ‘Promise’가 해결되면 결과 값을 반환한다.

* **비동기 함수**: 함수 호출이 완료될 때까지 기다리지 않고 다음 코드를 실행한다. 비동기 작업이 완료되면, 그 작업의 결과는 나중에 처리된다.

비동기 작업을 처리하는 방법에는 ‘콜백 함수’, ‘프라미스’(Promise), ‘async/await’ 등이 있다.

* 동기 함수: 함수 호출이 완료될 때까지 코드의 실행이 멈추는 방식이다. 즉, 함수가 반환될 때까지 프로그램의 다른 부분은 실행되지 않는다.

**- ‘axios.post’ 함수**

: ‘axios.post(url, data)’는 주어진 ‘url’로 POST 요청을 보내고, ‘data’를 요청 본문으로 전송한다.

* ‘url’: 요청을 보낼 대상 URL
* ‘data’: 서버로 전송할 데이터

이 코드를 이해하기 위해서 필요한 지식은 아래와 같습니다.

1. **JavaScript 기본 문법**: 변수 선언, 함수 정의 등.

2. **비동기 프로그래밍**: async/await, Promise의 개념.

3. **HTTP 프로토콜**: GET, POST 등의 HTTP 메서드와 상태 코드.

4. **API 통신**: 서버와의 데이터 교환 방법.

5. **Node.js 환경**: 서버 측에서 JavaScript를 실행할 수 있는 환경 (또는 브라우저 환경).

6. **에러 처리**: try/catch 블록을 사용하여 비동기 작업의 에러를 처리하는 방법.

- 서버는 express의 POST 응답을 한다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**- Express.js를 사용한 Node.js 애플리케이션에서 로그인 기능을 구현**

**- 사용자의 로그인 요청을 처리하는 서버 측 코드로, 사용자가 입력한 이메일과 비밀번호를 데이터베이스의 정보와 비교하여, 일치하면 성공 메시지를, 일치하지 않으면 에러 메시지를 반환한다.**

좀 더 자세히 설명하자면,

‘**app.post('/api/signin', async (req, res) => { ... });**’에서는 ‘app.post’와 비동기 함수로 나눠서 볼 수 있다.

- ‘**app.post**’: ‘Express.js’에서 POST 요청을 처리하는 메서드이다.

* 첫 번째 인수: ‘**'/api/signin'**’은 경로(엔드포인트)이다.
* 두 번째 인수: 요청을 처리하는 비동기 콜백 함수.

- **비동기 함수**: ‘**async**’ 키워드는 함수가 비동기임을 나타낸다. 비동기 함수 내에서는 ‘await’ 키워드를 사용할 수 있다.

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

: 클라이언트가 보내는 로그인 데이터를 받는다.

🡪 요청 본문(‘req.body’)에서 ‘email’과 ‘password’를 추출한다.

‘**try { ... } catch (err) { ... }**’

: 서버에서 발생할 수 있는 예외 상황을 처리한다.

🡪 ‘try’ 블록 안의 코드를 실행하다가 에러가 발생하면, ‘catch’ 블록이 실행된다.

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

: 용자가 존재하지 않으면, 401 상태 코드와 함께 에러 메시지를 반환한다.

🡪 **401** 상태 코드: **"Unauthorized"(인증 실패)**를 의미한다.

‘const isMatch = await bcrypt.compare(password, user.password);’에서 ‘**bcrypt.compare**’

: 비밀번호를 해시된 비밀번호와 비교하기 위해 ‘bcrypt’ 라이브러리의 compare 메서드를 사용.

🡪 입력된 비밀번호와 데이터베이스에 저장된 해시된 비밀번호를 비교한다.

🡪 ‘await’ 키워드는 이 비동기 작업이 완료될 때까지 기다린다. (전체 코드에서)

‘**if (!isMatch) { ... }**’

: 비밀번호 일치 여부 확인

🡪 비밀번호가 일치하지 않으면, **401 상태 코드**와 함께 에러 메시지를 반환한다.

‘**return res.status(200).json({ status: 200, message: '로그인 성공' });**’

: 성공 응답

🡪 비밀번호가 일치하면, **200 상태 코드**와 함께 성공 메시지를 반환한다.

🡪 **200** 상태 코드: **"OK"(성공)**를 의미한다.

‘**catch (err) { return res.status(500).json({ status: 500, message: 'Server error'}); }**’

: 서버 에러 처리

🡪 ‘try’ 블록 내에서 발생한 모든 에러를 처리하여, **500 상태 코드**와 함께 서버 에러 메시지를 반환한다.

🡪 **500** 상태 코드: **"Internal Server Error"(서버 내부 오류)**를 의미

이 코드를 이해하기 위해 필요한 지식은 아래와 같습니다.

**1. Express.js**

(Node.js를 위한 웹 애플리케이션 프레임워크로, 서버를 쉽게 설정하고 요청을 처리할 수 있다.)

🡪 위의 코드에서는 app.post를 사용하여 특정 경로로 오는 POST 요청을 처리하는 역할을 한다.

**2. 비동기 프로그래밍**

(‘async’/’await’ 키워드를 사용하여 비동기 작업을 동기 코드처럼 작성할 수 있다.)

(Promise를 반환하며, await 키워드는 Promise가 해결될 때까지 기다린다)

**3. MongoDB와 Mongoose**

(MongoDB는 NoSQL 데이터베이스로, Mongoose는 MongoDB를 위한 ODM(Object Data Modeling) 라이브러리이다.)

🡪 위의 코드에서 ‘User.findOne’은 MongoDB 컬렉션에서 조건에 맞는 하나의 문서를 찾는 Mongoose 메서드이다.

4. bcrypt

(비밀번호를 해시하고 비교하기 위한 라이브러리이다.)

🡪 ‘bcrypt.compare’는 평문 비밀번호를 해시된 비밀번호와 비교한다.

**5. HTTP 상태 코드**

🡪 200: 요청이 성공적으로 처리되었음을 나타낸다.

🡪 401: 인증되지 않은 요청임을 나타낸다.

🡪 500: 서버 내부 오류가 발생했음을 나타낸다.

**[로그인 처리]**

위에서 "URL/api/signin"로 받은 정보를 처리하는 API 엔드 포인트 코드를 구현한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이 코드는 사용자가 로그인 요청을 보내면, 서버에서 사용자의 이메일과 비밀번호를 확인하여 로그인 처리 결과를 반환하는 API 엔드포인트이다.

이를 구현하기 위해 Express.js, Mongoose, bcrypt 등의 라이브러리를 사용한다.

위에서 이미 다룬 코드와 동일한 내용의 코드로, ‘res.status().send();’에서 상태 코드 각각에 맞게 실행된다.

* **‘res.status(401).send();’**

: 클라이언트에게 인증되지 않았음을 알리는 응답으로 사용자가 제공한 자격 증명이 유효하지 않을 때 사용.

🡪 “**인증되지 않음**”을 나타냄

* **‘res.status(200).send();’**

: 클라이언트에게 요청이 성공적으로 처리되었음을 알리는 응답으로 요청이 성공적으로 완료되었을 때 사용.

🡪 “**OK”(성공**)을 나타냄.

* **‘res.status(500).send();’**

: 클라이언트에게 서버 내부 오류가 발생했음을 알리는 응답으로 서버에서 예기치 않은 오류가 발생했을 때 사용.

🡪 “**Internal Server Error**”(서버 내부 오류)를 나타냄.

🡪 클라이언트에게 인증되지 않았음을 명확하게 알리는 방법이다.