Hello,

KDT 웹 개발자 양성 프로젝트

1기! 33th

with

과제 리뷰!


DATABASE

MongoDB 로 DB 적용하기!

```
router.delete('/title/:title', async (req, res) => {
  MongoClient.connect(uri, (err, db) => {
    const data = db.db('kdt1').collection('board');

    data.deleteOne({ title: req.params.title }, (err, result) => {
      if (err) {
        res.send('해당 제목의 글이 없습니다');
      } else {
        res.send('삭제 완료');
      }
    });
  });
});
```

Async / Await를 적용하여 모듈화

```
router.get('/', async (req, res) => {
  const client = await mongoClient.connect();
  const cursor = client.db('kdt1').collection('board');
  const ARTICLE = await cursor.find({}).toArray();

  const articleLen = ARTICLE.length;
  res.render('board', { ARTICLE, articleCounts: articleLen });
});
```

서버 배포!

GitHub

NAVER CLOUD PLATFORM

aws

سحاب

aws

키페어 권한 설정하기

- 먼저 터미널 또는 Git-bash 를 이용해서 키페어 파일을 저장한 폴더로 이동
- 키페어 권한 설정
- chmod 400 키페어이름.pem

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업 무 /KDT/_pem
$ chmod 400 xenosign.pem
```

SSH 를 이용 서버 접속

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT/_pem
$ ssh -i xenosign.pem ec2-user@43.200.173.233
The authenticity of host '43.200.173.233 (43.200.173.233)' can't be established.
ED25519 key fingerprint is SHA256:CM4MUSstYESY2+94kLTNCq89pn+veo5AF/E7XFErMrk.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '43.200.173.233' (ED25519) to the list of known hosts
.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
3 package(s) needed for security, out of 8 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-46-71 ~]$
```

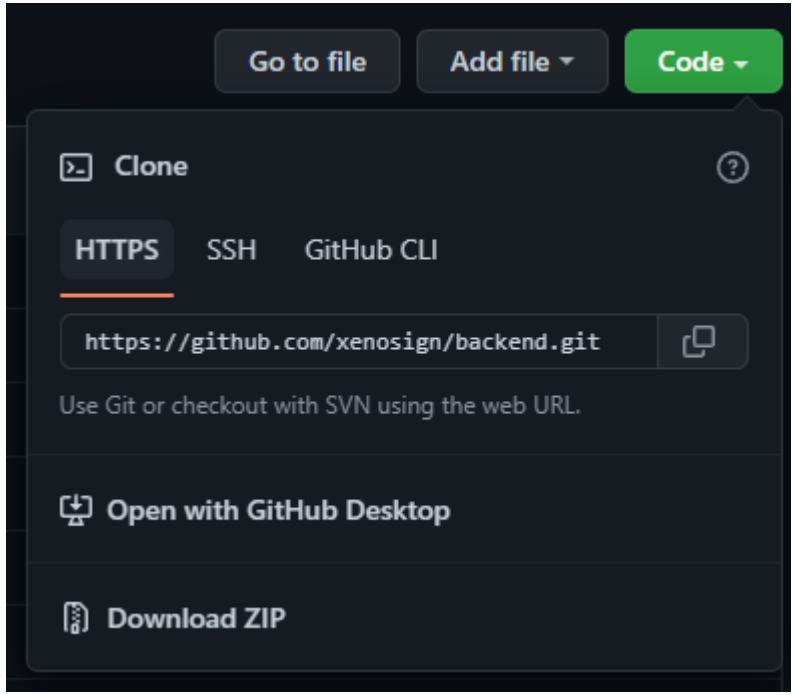
- ssh -i 키페어이름.pem ec2-user@퍼블릭IP주소
- 다음 질문에서 Yes 입력

EC2에 Node.js 설치

- https://docs.aws.amazon.com/ko_kr/sdk-for-javascript/v2/developer-guide/setting-up-node-on-ec2-instance.html
- Nvm 을 설치
  - curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
- Nvm 활성화
  - . ~/.nvm/nvm.sh

배포할 git, clone 주소 복사

- 자신이 배포하려는 깃허브 Repo 에 가서 Clone 주소 복사



PM2(Process Manager 2)

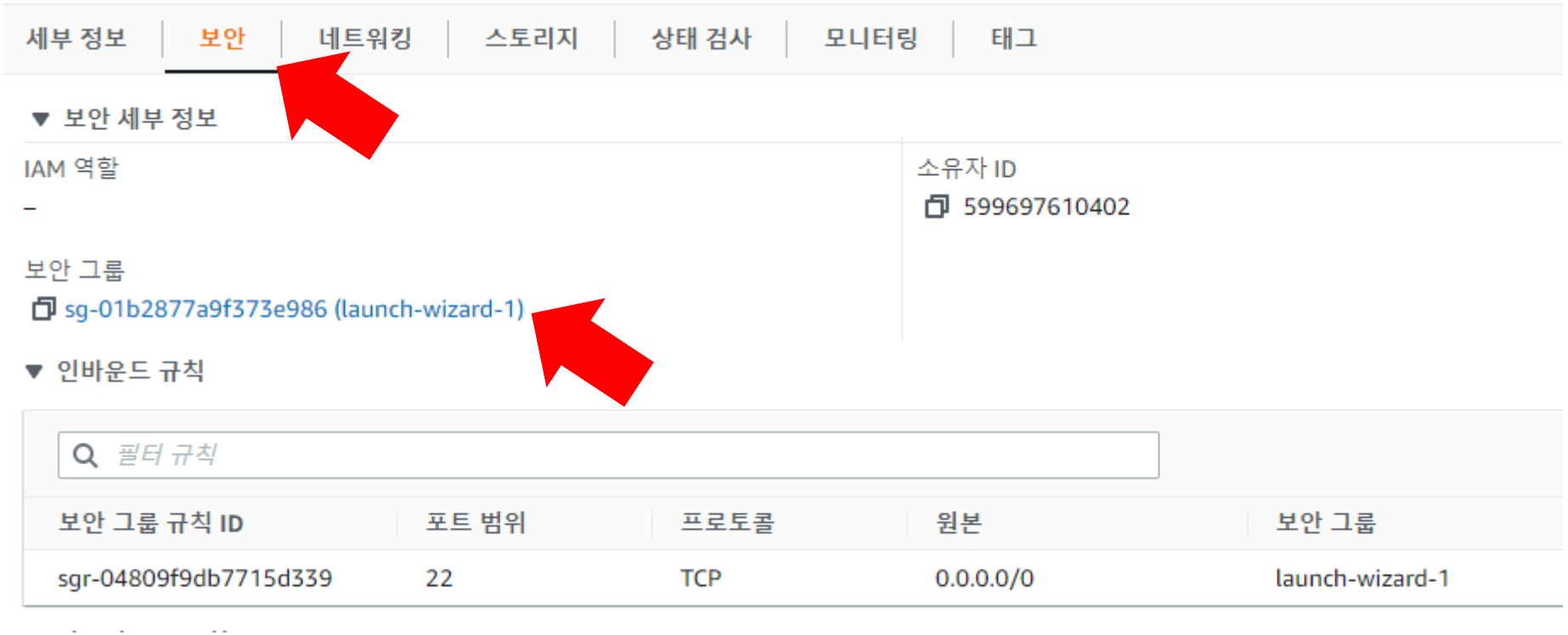
- PM2로 프로세스 실행하기
  - pm2 start 앱파일이름
- PM2로 프로세스 중단하기
  - pm2 stop 앱파일이름

```
[PM2] Spawning PM2 daemon with pm2_home=/home/ec2-user/.pm2
[PM2] PM2 Successfully daemonized
[PM2] Starting /home/ec2-user/app/backend/app.js in fork_mode (1 instance)
[PM2] Done.
```

id	name	mode	↺	status	cpu	memory
0	app	fork	0	online	0%	32.2mb

Port 설정이 필요합니다!

- 지금은 기본 설정이라서 port 번호 22번만 열려 있습니다!
- 직접 작성한 Port 번호를 보안 그룹에 추가하여 접속이 가능하게 만들어 봅시다!



MongoDB, IP 설정

효석'S ORG - 2022-09-06 > PROJECT 0

Network Access

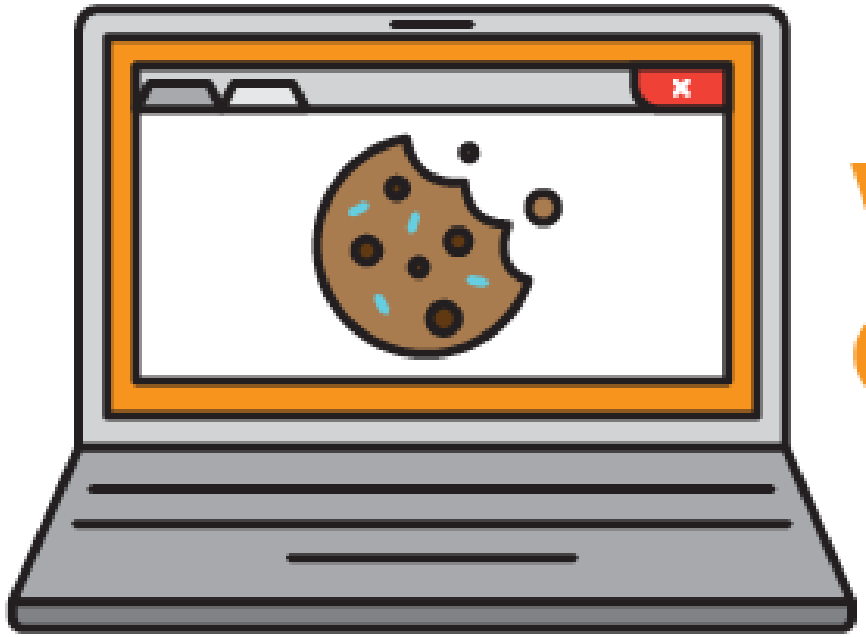
IP Access List | Peering | Private Endpoint

+ ADD IP ADDRESS

You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
115.136.110.37/32 (includes your current IP address)		Active	EDIT DELETE

Cookie!?



WEBSITE COOKIES

Cookie

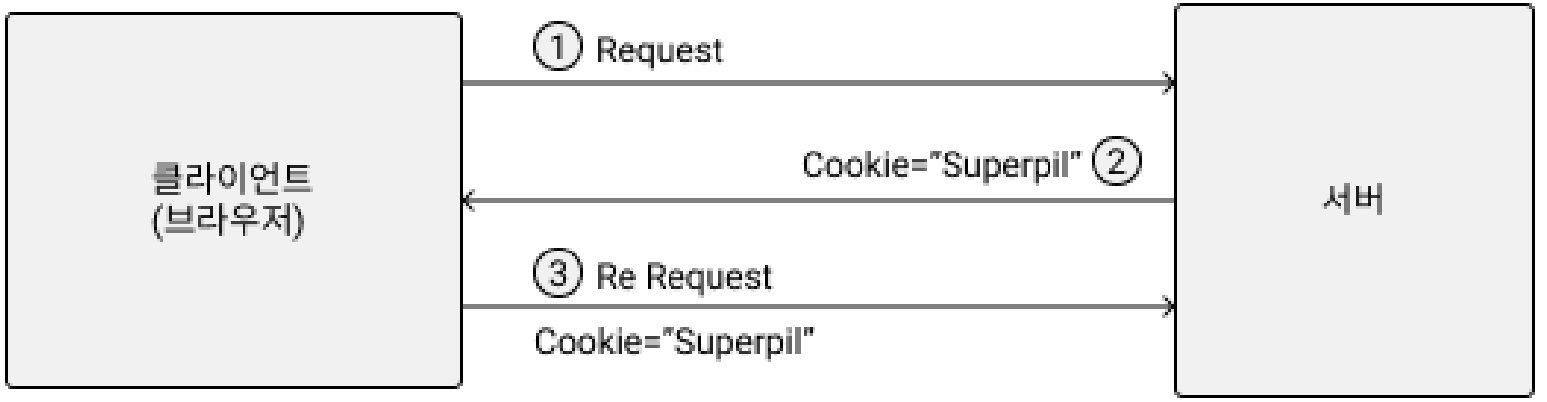
- HTTP 쿠키는 서버가 사용자의 웹 브라우저에 전송하여 저장하는 작은 데이터 저장소
- JS의 객체처럼 key 값과, data 가 들어 있습니다
- 브라우저는 쿠키로 데이터를 저장해 두었다가, 동일한 서버에 다시 요청을 보내게 될 때 쿠키에 저장된 정보를 함께 전송합니다
- 따라서, 서버에서는 쿠키를 통해 요청이 어떤 브라우저에 들어왔는지(혹은 요청이 동일한 브라우저에서 들어왔는지) 등을 파악할 수 있습니다

Cookie

- 서버에 저장 해야할 정보를 관리 → 로그인 정보, 장바구니, 팝업 하루 동안 안보기 등
- 중요한 정보는 아무래도 로컬에 남게 되므로 절대 저장해서는 안됩니다
- 중요 정보로는 이름, 데이터, 만료일, 경로 등이 저장 됩니다

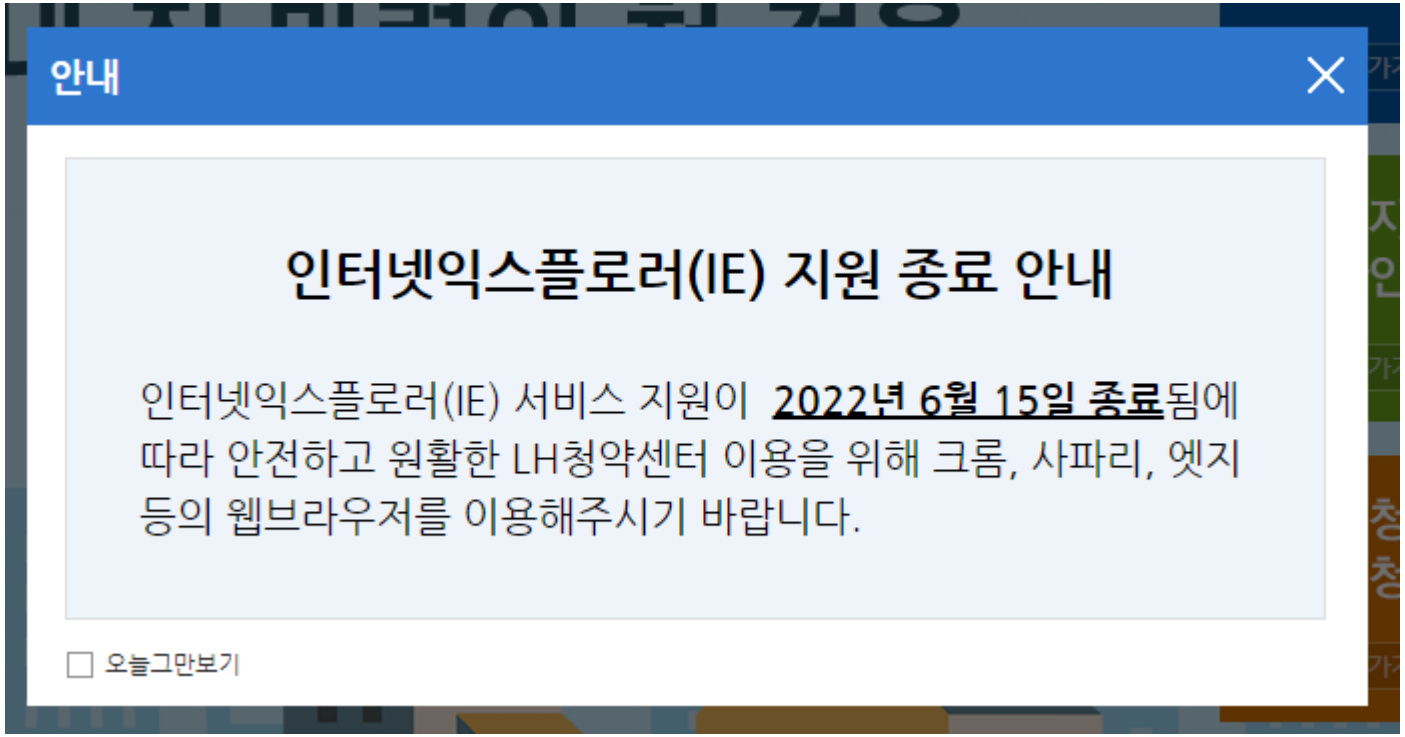
Cookie 의 동작 방식

쿠키의 동작 방식



1. 클라이언트가 페이지를 요청
2. 서버에서 쿠키를 생성
3. HTTP 헤더에 쿠키를 포함 시켜 응답
4. 브라우저가 종료되어도 쿠키 만료 기간이 있다면 클라이언트에서 보관하고 있음
5. 같은 요청을 할 경우 HTTP 헤더에 쿠키를 함께 보냄
6. 서버에서 쿠키를 읽어 이전 상태 정보를 변경 할 필요가 있을 때 쿠키를 업데이트 하여 변경된 쿠키를 HTTP 헤더에 포함시켜 응답

Cookie 사용 예시들



프론트에서

쿠키 사용하기

프론트에서 쿠키 사용하기

- Document.cookie 로 바로 만들어서 사용하면 됩니다

```
<script>
    console.log(document.cookie);
    document.cookie = "user=kdt; expires=19 Sep 2022 13:00:00 GMT; path=/";
    document.cookie = "test=test1; expires=19 Sep 2022 13:00:00 GMT; path=/";
    console.log(document.cookie);
  </script>
```

- Expires 값과 브라우저의 시간을 비교해서 쿠키가 해당 시간이 되면 자동으로 삭제 처리

프론트에서 쿠키 사용하기

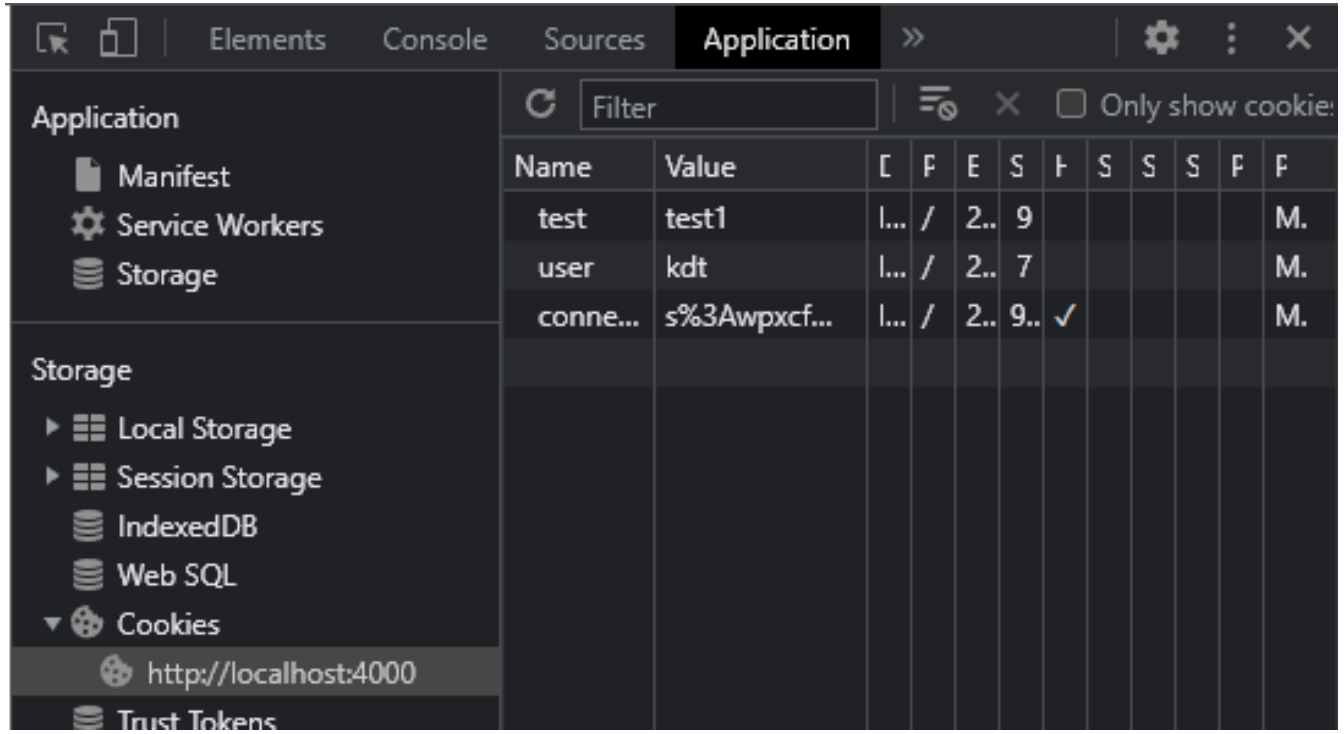
- 쿠키 여부를 체크해서 이런 방식으로도 사용이 가능합니다

```
console.log(document.cookie);
document.cookie = "user=kdt; expires=19 Sep 2022 13:00:00 GMT; path=/";
document.cookie = "test=test1; expires=19 Sep 2022 13:00:00 GMT; path=/";
console.log(document.cookie);

window.onload = () => {
  if (document.cookie) alert(`쿠키 팔아요! ${document.cookie}`);
};
```

- 프론트의 JS 코드를 통해 특정 상황에서 페이지가 새로 고침해도 저장이 필요한 정보를 쿠키에 저장하여 DB처럼 활용도 가능합니다
- 다만, 딱히 쓸 일이 많지는 않습니다! ☺

브라우저에서 쿠키 확인하기

- 개발자 도구 → Application → Storage → Cookies 에 가면 쿠키 정보 확인이 가능합니다



백엔드에서

쿠키 사용하기

백엔드(express)에서 쿠키 사용하기

- 백엔드에서는 cookie-parser 라는 모듈을 사용합니다!
- 일단 설치 → npm i cookie-parser -s
- 메인 서버에 쿠키 모듈 호출 및 미들 웨어 등록!

```
const cookieParser = require('cookie-parser');
app.use(cookieParser());
```

백엔드(express)에서 쿠키 사용하기

- 이제 쿠키를 사용할 곳에서 res.cookie('쿠키 이름', '데이터', '옵션 객체'); 를 써서 사용하면 됩니다!

```
res.cookie('alert', true, {
    expires: new Date(Date.now() + 1000 * 60),
    httpOnly: true,
});
```

- 옵션의 의미
  - Expires: 쿠키가 자동으로 삭제되는 일자를 지정
  - httpOnly: 해당 쿠키는 서버와의 http 통신에서만 읽을 수 있음을 표시, 프론트에서 처럼 JS로 해당 쿠키를 읽으려 하면 웹브라우저가 이를 차단

실제로 쿠키가 잘 작동하는지 확인하기

- Index.ejs 에서 cookie가 서버에서 잘 전송 되었는지 확인해 봅시다!
- 먼저 프론트에서 쿠키를 발행하는 코드는 주석 처리 + 기존에 발행 된 쿠키 도 삭제 처리

```
window.onload = () => {
    if (document.cookie) alert(`쿠키 팔아요! ${document.cookie}`);
};
```

실제로 쿠키가 잘 작동하는지 확인하기

- 엥? 안되는데요?
- 안되는 이유는 httpOnly 옵션 때문에 document.cookie 의 값을 읽으려고 하는 것을 브라우저가 차단하기 때문입니다.
- httpOnly 옵션을 false 로 만들고 다시 해봅시다!

쿠키의 값을 전달

- 쿠키를 생성하면서 쿠키의 값을 res.render 에 담아 같이 전달해서 사용도 가능합니다!
- 생성된 쿠키에 대한 접근은 req.cookies.쿠키이름 으로 하면 됩니다

```
res.cookie('alert', true, {
    expires: new Date(Date.now() + 1000 * 60),
    httpOnly: true,
  });
res.render('index', { alert: req.cookies.alert });
```

쿠키의 값을 전달

```
<script>
    window.onload = () => {
      if ('<%= alert %>' === 'true') alert(`쿠키 팔아요! ${document.cookie}`);
    };
</script>
```

- httpOnly 옵션이 켜져 있음에도 값을 전달 하였기 때문에 alert 창이 잘 뜨는 것을 볼 수 있습니다!
- 단, 쿠키 내용은 안뜹니다!

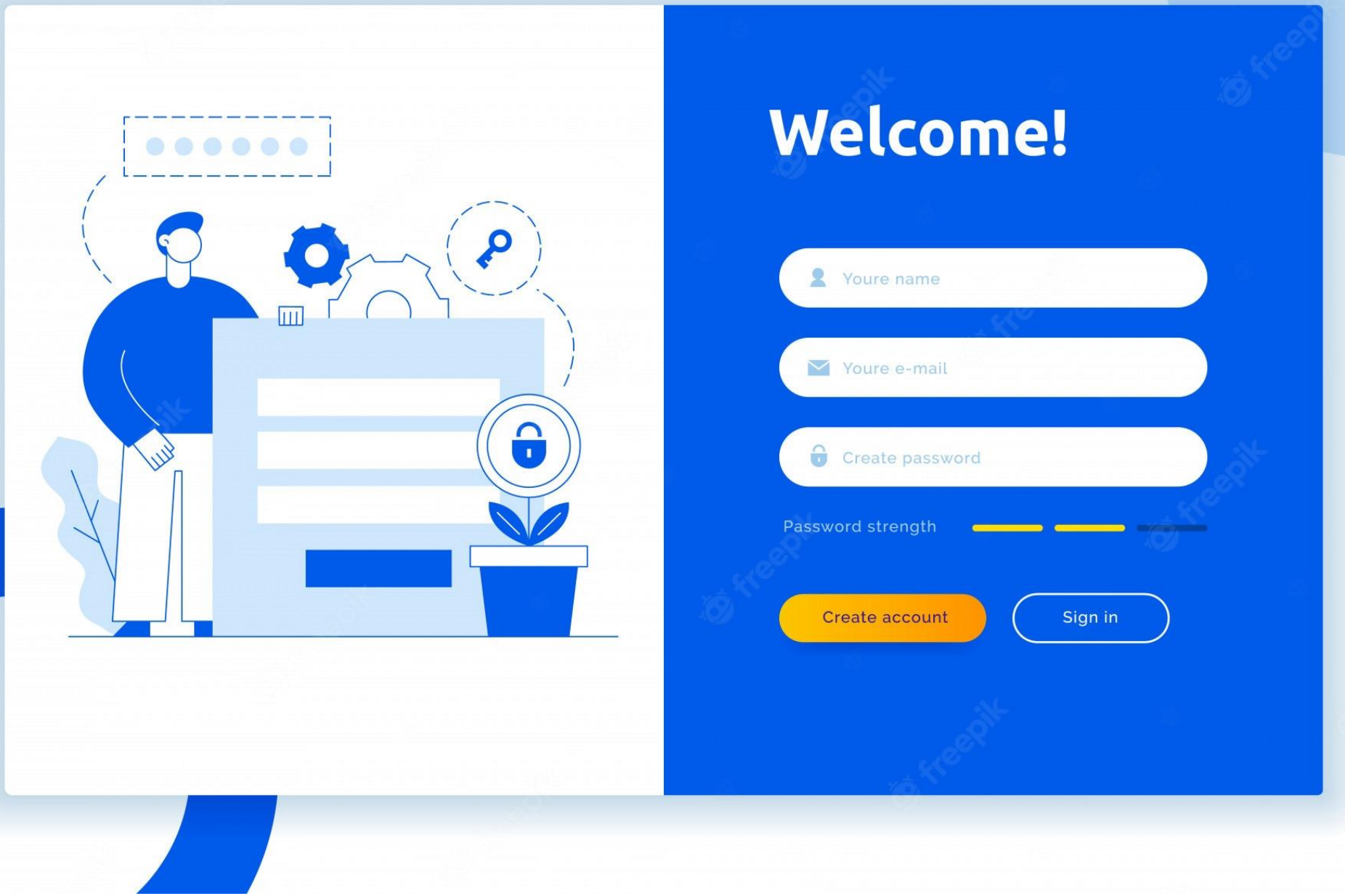
실습, 팝업창 오늘 하루동안 닫기 구현

- https://github.com/xenosign/backend/blob/main/views/index%20cookie.ejs
- 위의 코드를 index.ejs 파일에 덮어 씌워 주세요!
- 여러분이 직접 index.ejs 파일에 구현한 부분이 있다면 아래의 코드만 붙여 넣어 주세요!
  - <head> 에서 부트 스트랩 불러오는 코드 2줄
  - <!-- Modal --> 코드
  - 하단의 <script> 코드

실습, 팝업창 오늘 하루동안 닫기 구현

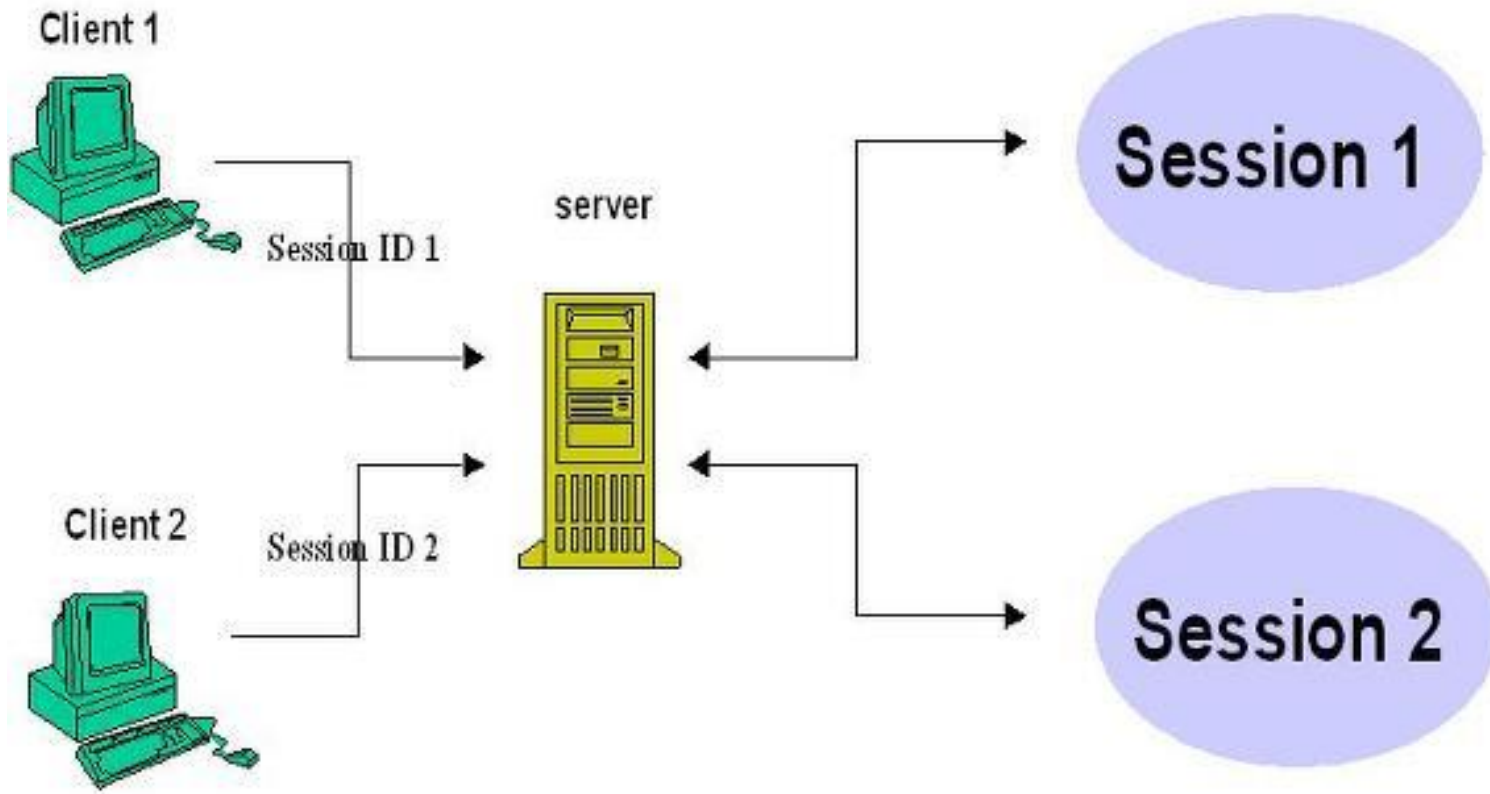
- 팝업이 뜬 상황에서 오늘 하루 안보기를 체크하고 닫기를 누르면 쿠키를 발행하고, 해당 쿠키에 의해 팝업이 하루동안 안보이도록 구현!
- <script>의 if 문 내부의 조건 채우기 부분 + modalClose() 함수의 //하루동안 안보기 기능 구현하기 부분을 구현!
- Index.js 라우터에서 /cookie 라는 주소를 POST 방식으로 받고 팝업창을 하루동안 안띄워주는 쿠키를 발행
- Index 페이지를 render 할 때, 쿠키의 값을 전달해서 처리


LOGIN


Welcome!
Youre name
Youre e-mail
Create password
Password strength
Create account
Sign in

Session

(서버의 쿠키)


Client 1
Session ID 1
server
Session 1
Client 2
Session ID 2
Session 2

HTTP Session 이란?

- 브라우저가 아닌 서버에 저장되는 쿠키
- 사용자가 서버에 접속한 시점부터 연결을 끝내는 시점을 하나의 상태로 보고 유지하는 기능을 함 → 로그인 유지
- 서버는 각 사용자에 대한 세션을 발행하고 서버로 접근(Request)한 사용자를 식별하는 도구로 사용
- 쿠키와 달리 저장 데이터에 제한이 없음
- 만료 기간 설정이 가능하지만, 브라우저가 종료되면 바로 삭제

HTTP Session 의 동작 방식

- 사용자가 최초로 서버 연결을 하면 하나의 Session-ID(임의의 긴 문자열)가 발행 됩니다
- 발행 된 Session-ID 는 서버와 브라우저의 메모리에 쿠키 형태로 저장이 됩니다
- 서버는 사용자가 서버에 접근 시, 쿠키에 저장 된 Session-ID를 통해서 서버는 사용자를 구분하고 요청에 대한 응답을 합니다

Cookie vs Session

- 하는 역할은 비슷
- 쿠키는 로컬에 저장 되므로 보안 이슈가 발생 가능
- 세션은 로컬에 session-id 만 저장하고, 데이터는 서버에서 처리하므로 보안이 더 좋음
- 단, 쿠키는 데이터를 바로 저장하고 있으므로 속도가 빠름.
- 세션은 쿠키에서 session-id 를 읽어서 서버에서 데이터를 받아야 하므로 속도는 더 느림

Session 으로 로그인 구현하기

먼저 회원 가입, 로그인 페이지 만들기!

- 먼저 회원 가입, 로그인 페이지 부터 만들겠습니다
- 이 참에 간단하게 부트 스트랩의 사용법에 대해서 알아보시죠!
- 부트 스트랩은 css 를 class 에서 적용이 가능해서 편리
- 또한, container 를 지원해서 반응형을 쉽게 구현이 가능합니다!
- https://getbootstrap.com/
- https://uncovered-nutmeg-b8e.notion.site/06-Bootstrap-cec3422080c7462b87894b9438ee8f0f

회원 가입 페이지

```
<body>
    <div class="row">
      <div class="col-8 col-md-4 col-lg-10 bg-primary">col-1</div>
      <div class="col-4 col-md-8 col-lg-2 bg-warning">col-2</div>
    </div>

  <div class="container-lg d-flex flex-column align-items-center bg-light w-25 mt-5 p-5 rounded-3">
    <h2>회원가입</h2>
    <form action="/register" method="POST" class="w-75">
      <div class="mt-3 id">
        <label class="mb-2" for="input-email">아이디</label>
        <input type="text" name="id" class="form-control" id="input-email" placeholder="아이디를 입력
하세요" required />
      </div>
      <div class="mt-3 password">
        <label class="mb-2" for="input-pw">비밀번호</label>
        <input type="password" name="password" class="form-control" id="input=pw" placeholder="비밀번
호를 입력하세요" required />
      </div>
      <div class="d-flex justify-content-center mt-4 password">
        <button type="submit" class="btn btn-primary">회원가입</button>
      </div>
    </form>
  </div>
</body>
```

로그인 페이지

```
<body>
    <div class="container-lg d-flex flex-column align-items-center bg-light w-25 mt-5 p-5 rounded-3">
        <h2>로그인</h2>
        <form action="/login" method="POST" class="w-75">
            <div class="mt-3 id">
                <label class="mb-2" for="input-email">아이디</label>
                <input type="text" name="id" class="form-control" id="input-email" placeholder="아이디
를 입력하세요" required />
            </div>
            <div class="mt-3 password">
                <label class="mb-2" for="input-pw">비밀번호</label>
                <input type="password" name="password" class="form-control" id="input=pw"
placeholder="비밀번호를 입력하세요"
                    required />
            </div>
            <div class="d-flex justify-content-center mt-4 password">
                <button type="submit" class="btn btn-primary">로그인</button>
            </div>
        </form>
    </div>
</body>
```

랜딩 페이지 변경

랜딩 페이지를 변경

- 세션 테스트를 위해 회원가입 / 로그인 / 게시판 서비스로만 이동이 가능 하도록 랜딩 페이지 변경

```
<div class="container mt-5">
    <h1 class="text-center">Hello, Express Service</h1>
    <h2 class="mt-5 text-center"><a href="/login">로그인 바로가기</a></h2>
    <h2 class="mt-5 text-center"><a href="/register">회원가입 바로가기</a></h2>
    <h2 class="mt-5 text-center"><a href="/board">게시판 바로가기</a></h2>
  </div>
```

Session

모듈 추가하기

세션 모듈 추가하기

- 먼저 express-session 모듈 부터 설치 합시다
  - npm i express-session -s
- 모듈 추가 및 미들웨어 연결

```
const session = require('express-session’);
const app = express();
app.use(
  session({
    secret: 'tetz',
    resave: false,
    saveUninitialized: true,
    cookie: {
      maxAge: 1000 * 60 * 60,
    },
  })
);
```

세션 모듈 옵션 설명

- secret: 세션을 발급할 때 사용되는 키 값(아무거나 입력 가능)
- resave: 모든 request 마다 기존에 있던 session에 아무런 변경사항이 없어도 session 을 다시 저장하는 옵션
- saveUninitialized: 세션에 저장할 내역이 없더라도 처음부터 세션을 생성할지 설정
- secure → https 에서만 세션을 주고받을 수 있습니다. http 에서는 세션을 주고받는 것이 불가능
- cookie : 세션 쿠키 설정 (세션 관리 시 클라이언트에 보내는 쿠키)
  - maxAge: 쿠키의 생명 기간이고 단위는 ms입니다.
  - httpOnly → 자바스크립트를 통해서 세션을 사용할 수 없도록 강제

resister.js

구현하기

회원 가입 용 라우터 구현

- register.js 를 만들어서 회원가입 기능을 모듈화

```
const express = require('express');
const mongoClient = require('./mongo');
const router = express.Router();

router.get('/', (req, res) => {
  res.render('register');
});

module.exports = router;
```

- 서버에 라우터 등록

```
const registerRouter = require('./routes/register');
app.use('/register', registerRouter);
```

Register.ejs 에서 데이터 받기

- POST 방식 /register 주소로 회원 가입을 요청하므로 처리
- ‘uesrs’ 라는 컬렉션을 만들어서 회원 데이터를 관리
- Req.body 에 담겨있는 ID 가 DB 에 존재하는지 확인하고, 존재할 경우 id 중복 안내 → 회원 가입 페이지로 이동 안내
- ID가 중복되지 않을 경우, DB에 새로운 회원을 등록하고 회원 가입 성공 메시지를 출력 → 로그인 페이지로 이동 안내

```
router.post('/', async (req, res) => {
  const client = await mongoClient.connect();
  const userCursor = client.db('kdt1').collection('users');
  const duplicated = await userCursor.findOne({
    id: req.body.id,
  });

  if (duplicated === null) {
    const result = await userCursor.insertOne({
      id: req.body.id,
      password: req.body.password,
    });
    if (result.acknowledged) {
      res.send('회원 가입 성공!<br><a href="/login">로그인 페이지로 이동</a>');
    } else {
      res.status(404);
      res.send(
        '회원 가입 문제 발생.<br><a href="/register">회원가입 페이지로 이동</a>'
      );
    }
  } else {
    res.status(404);
    res.send(
      '중복된 id 가 존재합니다.<br><a href="/register">회원가입 페이지로 이동</a>'
    );
  }
});
```

login.js

구현하기

로그인 용 라우터 구현

- login.js 를 만들어서 회원가입 기능을 모듈화

```
const express = require('express');
const mongoClient = require('./mongo');
const router = express.Router();

router.get('/', async (req, res) => {
  res.render('login');
});

module.exports = router;
```

- 서버에 라우터 등록

```
const loginRouter = require('./routes/login');
app.use('/login', loginRouter);
```

로그인 구현

로그인 처리

- POST 방식 /login 주소로 로그인을 요청하므로 처리
- Req.body 에 담겨있는 ID 가 DB 에 존재하는지 확인하고, 존재 하지 않을 경우 해당 id 가 없다고 안내 → 로그인 페이지로 이동 안내
- ID 가 있을 경우, ID와 비밀번호까지 일치하는지 확인하고 둘 다 일치하는 경우 로그인 성공 → req.session 을 사용하여 로그인 여부 / 로그인된 ID 를 session 에 저장 → 게시판 서비스로 이동
- 비밀번호가 틀리면 비밀 번호가 불일치 안내 → 로그인 페이지로 이동 안내

```
router.post('/', async (req, res) => {
  const client = await mongoClient.connect();
  const userCursor = client.db('kdt1').collection('users');
  const idResult = await userCursor.findOne({
    id: req.body.id,
  });

  if (idResult !== null) {
    const result = await userCursor.findOne({
      id: req.body.id,
      password: req.body.password,
    });
    if (result !== null) {
      req.session.login = true;
      req.session.userId = req.body.id;
      res.redirect('/board');
    } else {
      res.status(404);
      res.send(
        '비밀번호가 틀렸습니다.<br><a href="/login">로그인 페이지로 이동</a>'
      );
    }
  } else {
    res.status(404);
    res.send(
      '해당 id 가 없습니다.<br><a href="/login">로그인 페이지로 이동</a>'
    );
```

로그 아웃 구현

로그 아웃 버튼 만들기

- Board.ejs 파일에 로그아웃 버튼 추가
- GET 방식 /login/logout 주소로 로그아웃 요청

```
<div class="board_write">
        <span>현재 등록 글 :   <%= articleCounts %></span>
        <a class="btn red" href="/board/write">글쓰기</a>
        <a class="btn orange" href="/login/logout">로그아웃</a>
      </div>
```

로그 아웃 처리

- 로그 아웃 요청이 들어오면 생성 된 req.session 을 삭제 처리 → 최초 화면으로 이동

```
router.get('/logout', async (req, res) => {
  req.session.destroy((err) => {
    res.redirect('/');
  });
});
```

로그인 여부에 따른 게시판 서비스 변경

로그인 여부에 따른 상황 처리

- 로그인이 안되어 있으면 게시판에 접속이 불가능 하도록 수정
- Req.session 은 어느 라우터에서나 불러서 쓸 수 있다!
- Req.session.login 의 값을 확인해서 게시판 서비스로 이동 할지, 로그인 페이지 이동을 안내할지 결정
- Board.ejs 파일에 req.session.id 에 저장된 회원 id 정보도 같이 전달!

```
router.get('/', async (req, res) => {
  if (req.session.login) {
    const client = await mongoClient.connect();
    const cursor = client.db('kdt1').collection('board');
    const ARTICLE = await cursor.find({}).toArray();

    const articleLen = ARTICLE.length;
    res.render('board', {
      ARTICLE,
      articleCounts: articleLen,
      userId: req.session.userId,
    });
  } else {
    res.status(404);
    res.send('로그인 해주세요.<br><a href="/login">로그인 페이지로 이동</a>');
  }
});
```

로그인 확인용 함수를 이용

- 미들웨어의 2번째 매개 변수로 로그인 확인용 함수를 넣어서 처리하는 방법도 있습니다!
- 로그인 여부를 req.session 값을 통해 판별하고 로그인이 되어 있으면 next() 를 이용 뒤의 익명 함수를 수행하는 구조를 가집니다
- If 문을 덜 사용하고 편리하게 사용이 가능합니다

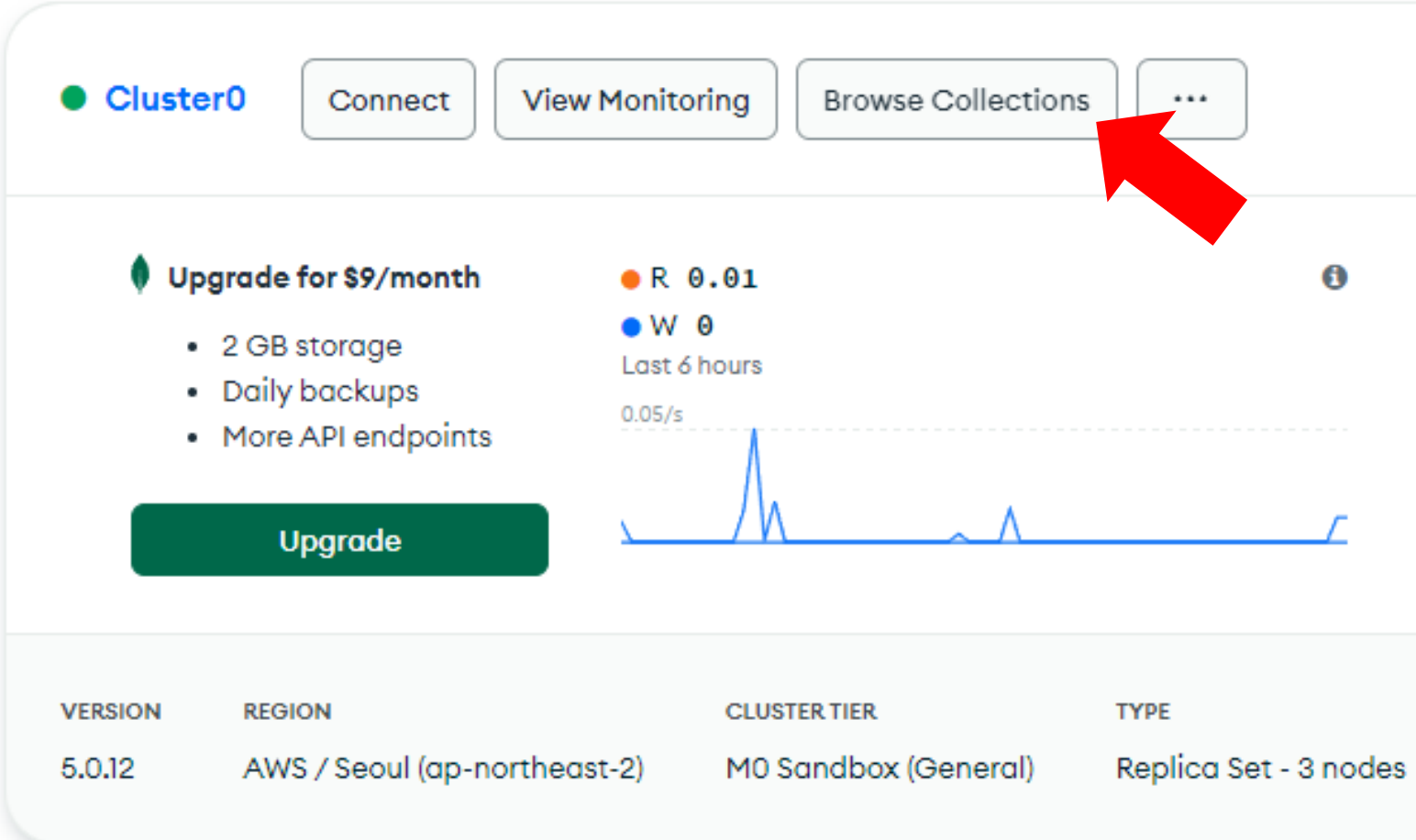
```
function isLogin(req, res, next) {
  if (req.session.login) {
    next();
  } else {
    res.send('로그인 해주세요.<br><a href="/login">로그인 페이지로 이동</a>');
  }
}

router.get('/', isLogin, async (req, res) => {
  const client = await mongoClient.connect();
  const cursor = client.db('kdt1').collection('board');
  const ARTICLE = await cursor.find({}).toArray();

  const articleLen = ARTICLE.length;
  res.render('board', {
    ARTICLE,
    articleCounts: articleLen,
    userId: req.session.userId,
  });
});
```

게시글에

작성자 id 정보 추가

게시글에 작성자 id 정보 추가

- 각각의 게시글의 작성자가 누구인지 알려주는 기능을 추가
- 자신이 작성한 글은 수정 및 삭제 버튼이 보이도록 기능을 추가
- 위의 기능 추가를 위해서는 게시글 DB에 작성자의 id 정보가 있어야함!
- 몽고 DB에 가서 일단 추가 해봅시다!


Cluster0
Connect
View Monitoring
Browse Collections
...
Upgrade for $9/month
2 GB storage
Daily backups
More API endpoints
Upgrade
R 0.01
W 0
Last 6 hours
0.05/s
VERSION
5.0.12
REGION
AWS / Seoul (ap-northeast-2)
CLUSTER TIER
M0 Sandbox (General)
TYPE
Replica Set - 3 nodes

+ Create Database

Search Namespaces

- kdt1
  - board
  - users

kdt1.board

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 162B TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 36KB

Find Indexes Schema Anti-Patterns 0 Aggregation Search Indexes

INSERT DOCUMENT

FILTER { field: 'value' } OPTIONS Apply Reset

QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('6322cd8b67425cd460843faa')
id: "1234"
title: "1234dd"
content: "1234
                dd"
```

```
_id: ObjectId('6322ee9a75b30d3e3b8acc41')
id: "13"
title: "132"
content: "13
                2"
```

```
_id: ObjectId('6322cd8b67425cd460843faa')                    ObjectId
title: "1234dd"                                              String
                                                             String
```

Add field after _id

```
1  _id: ObjectId('6322cd8b67425cd460843faa')                 ObjectId
   id: "13"                                                  String
3  title: "1234dd"                                           String
4  content: "1234                                            String
```

Document modified.

CANCEL UPDATE

Board.ejs

파일 수정

작성자 표시

- 제목 위에 작성자의 id 를 보여주는 부분 추가

```
<li>
    <div class="author">
        작성자 : <%= ARTICLE[i].id %>
    </div>
    <div class="title">
        <%= ARTICLE[i].title %>
    </div>
```

자신이 작성한 글에만 수정 삭제 버튼 표시

- 게시글의 작성자 id 와 로그인한 유저의 id 를 비교해서 수정 및 삭제 버튼 표시

```
<div class="foot">
    <% if (ARTICLE[i].id === userId) { %>
    <a class="btn orange" href="board/modify/<%= ARTICLE[i].title %>">수정</a>
    <a class="btn blue" href="#" onclick="deleteArticle('<%= ARTICLE[i].title %>')">삭제</a>
    <% } %>
</div>
```

게시글 추가 기능 수정

게시글 추가 페이지로 이동

- 로그인 상태가 아니면 해당 페이지로 이동이 안되도록 설정

```
router.get('/write', isLogin, (req, res) => {
  res.render('board_write');
});
```

게시글 추가 기능

- 글을 추가 할 때에도 로그인 여부 판별
- 새로운 게시글을 추가할 때, title, content 이외에 id 값으로 로그인한 유저의 id 값을 받아서 글을 추가

```
router.post('/', isLogin, async (req, res) => {
  if (req.body) {
    if (req.body.title && req.body.content) {
      const newArticle = {
        id: req.session.userId,
        title: req.body.title,
        content: req.body.content,
      };
```

```
        const client = await mongoClient.connect();
        const cursor = client.db('kdt1').collection('board');
        await cursor.insertOne(newArticle);
        res.redirect('/board');
      } else {
        const err = new Error('요청 이상');
        err.statusCode = 404;
        throw err;
      }
    } else {
      const err = new Error('요청에 데이터가 없습니다');
      err.statusCode = 404;
      throw err;
    }
} else {
    res.send('로그인 해주세요.<br><a href="/login">로그인 페이지로 이동</a>');
}
```

게시글 수정 기능

수정

게시글 수정 페이지로 이동

- 로그인 상태가 아니면 해당 페이지로 이동이 안되도록 설정

```
router.get('/modify/:title', isLogin, async (req, res) => {
  const client = await mongoClient.connect();
  const cursor = client.db('kdt1').collection('board');
  const ARTICLE = await cursor.find({}).toArray();
  const arrIndex = ARTICLE.findIndex(
    (_article) => _article.title === req.params.title
  );
  const selectedArticle = ARTICLE[arrIndex];
  res.render('board_modify', { selectedArticle });
});
```

게시글 수정 기능

- 로그인이 안되어 있으면 게시글 수정 요청이 안되도록 설정 그외의 부분은 동일하므로 건들 필요가 없음!

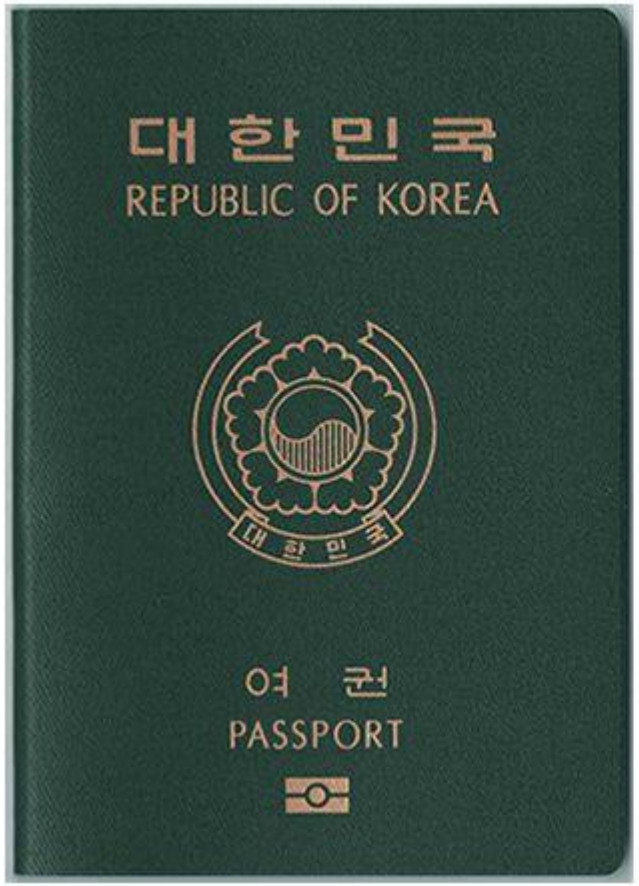
```
router.post('/title/:title', isLogin, async (req, res) => {
  // 기존 코드
});
```

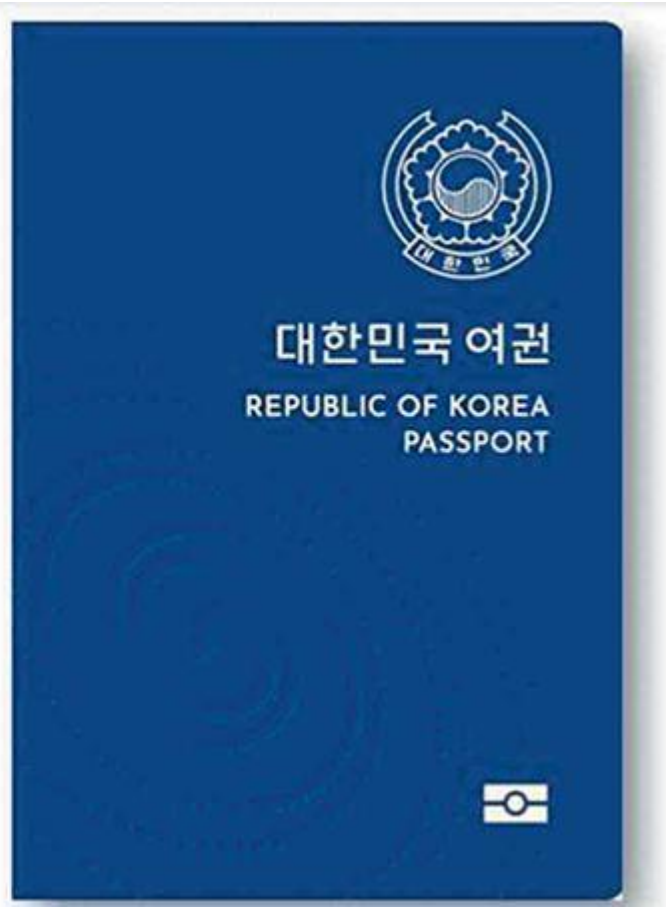
게시글 삭제 기능 수정

게시글 삭제 기능

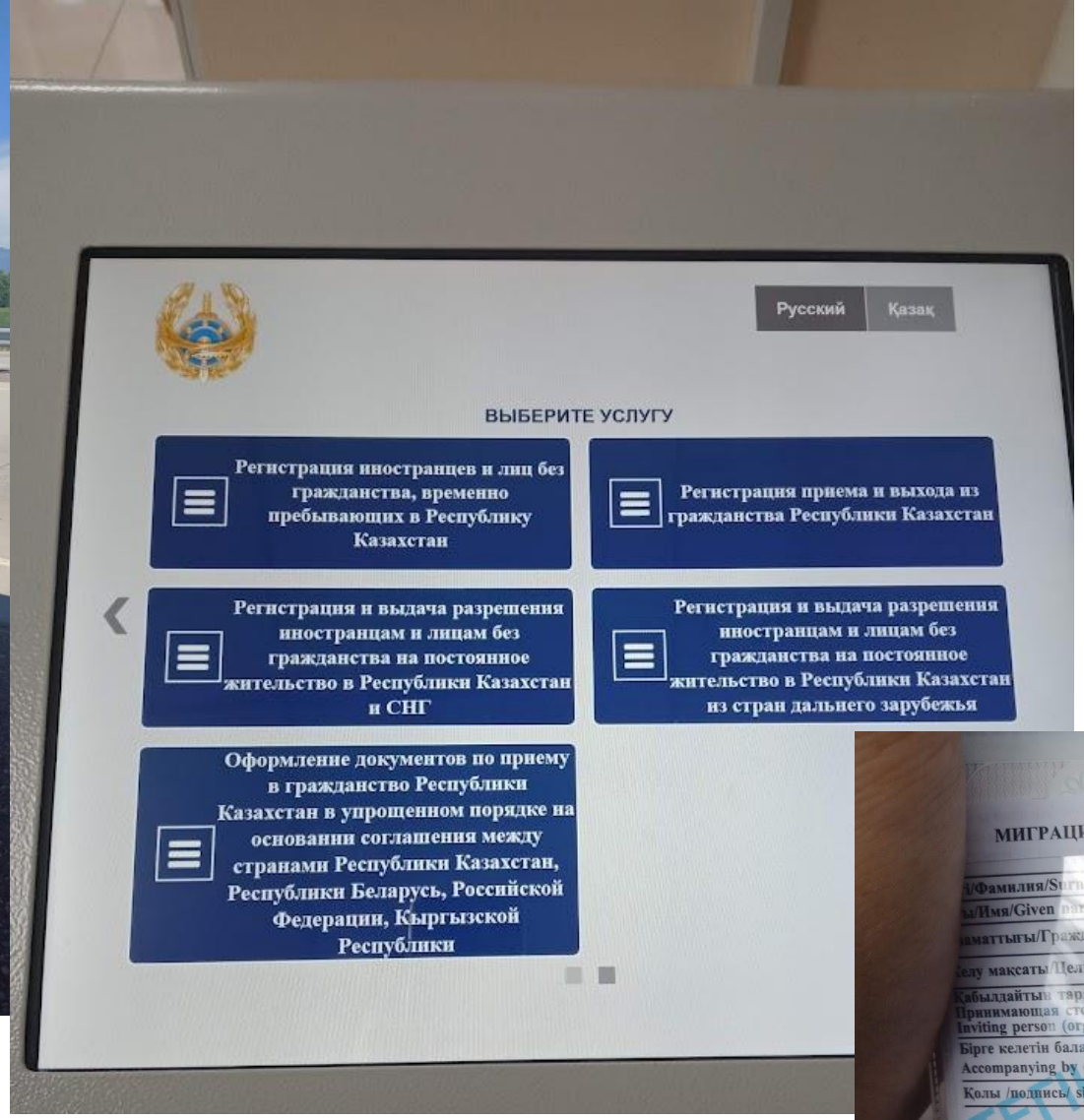
- 로그인이 안되어 있으면 게시글 삭제 요청이 안되도록 설정 그외의 부분은 동일하므로 건들 필요가 없음!

```
router.delete('/title/:title', isLogin, async (req, res) => {
  // 기존 코드
});
```


대 한 민 국
REPUBLIC OF KOREA
대 한 민 국
여 권
PASSPORT


대한민국 여권
REPUBLIC OF KOREA
PASSPORT

PASSPORT


Русский
Қазақ
ВЫБЕРИТЕ УСЛУГУ
Регистрация иностранцев и лиц без гражданства, временно пребывающих в Республику Казахстан
Регистрация приема и выхода из гражданства Республики Казахстан
Регистрация и выдача разрешения иностранцам и лицам без гражданства на постоянное жительство в Республики Казахстан и СНГ
Регистрация и выдача разрешения иностранцам и лицам без гражданства на постоянное жительство в Республики Казахстан из стран дальнего зарубежья
Оформление документов по приему в гражданство Республики Казахстан в упрощенном порядке на основании соглашения между странами Республики Казахстан, Республики Беларусь, Российской Федерации, Кыргызской Республики


ТӨЛҚҰЖАТҚА КӨШІ-ҚОН КАРТОЧКАСЫ
МИГРАЦИОННАЯ КАРТОЧКА к паспорту _____ №
MIGRATIONAL CARD to the passport
Кіру
Фамилия/Surname LEE
Имя/Given names HYOSEOK
Азаматтығы/Гражданство/Nationality KOR
Келу мақсаты/Цель въезда/Purpose of visit TOUR
Қабылдайтын тарап
Принимающая сторона
Inviting person (organization)
Бірге келетін балалар / Совместно следуют дети
Accompanying by children NO
Қолы /подпись/ signature
15 06 18
Шығу
Тіркеу № _____
«___» _____ 200 __ж. «___» _____ 200 __ж. дейін
Тіркеу жасалған органы _____
Лауазымды тұлғаның қолы _____
м/о

Passport?

- 로그인 기능을 다방면으로 도와주는 모듈입니다
- 사실 방금 전에 Session 다 배워 놓고 왜 지금 가르치냐 하실지도 모르겠지만... Session 을 알고 요걸 배우는게 중요합니다
- 그리고 (개인적으로) 이게 더 어려워요 ☺
- 대신, 요 친구는 다음 시간에 배울 Google, Facebook, Github 등 소셜 로그인 구현을 위한 기능을 제공하기 때문에 배울 필요가 있습니다
- 이때는 이게 훨 편합니다 ☺

Passport!

- http://www.passportjs.org/
- 우리는 Local 방식 → 전통적인 아이디, 비밀번호를 입력하는 방식을 이용하므로 passport-local 도 같이 설치해 줍니다
- 설치부터 합시다
- Npm i passport
- Npm i passport-local

Passport!

- 서버 코드에 모듈 추가하기(passport 는 세션을 사용합니다!)

```
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;

// Session
app.use(
  session({
    secret: 'tetz',
    resave: false,
    saveUninitialized: true,
    cookie: {
      maxAge: 1000 * 60 * 60,
    },
  })
);
// Passport
app.use(passport.initialize());
app.use(passport.session());
```

Passport, Local 전략 설정

- Passport 는 기본적으로 로그인을 어떤 방식으로 처리할 지 미리 정해놓습니다.
- 그리고 실제적 로그인 처리는 passport.authenticate 메소드를 이용하여 처리합니다.

Passport, Local 전략 설정

- 아이디와 비밀번호를 어떤 키 값으로 받을지 설정합니다
- 그리고 인증 절차를 수행하는 함수를 설정합니다
- 인증 절차 수행의 결과는 Callback 함수에 담아서 처리를 합니다.
- 복잡한 편이니 코드로 설명을 드릴게요!

```
new LocalStrategy(
      {
        // id, pw 로 사용할 키 값 설정
        usernameField: 'id',
        passwordField: 'password',
      },
```

```
// 실제 로그인 기능 구현 부분
      async (id, password, cb) => {
        // 서버에 유저 정보를 전달하여 id 가 있는지 확인
        const client = await mongoClient.connect();
        const userCursor = client.db('kdt1').collection('users');
        const idResult = await userCursor.findOne({ id });
        // id가 존재하면 비밀번호 까지 있는지 확인하기
        if (idResult !== null) {
          const result = await userCursor.findOne({
            id,
            password,
          });
          // 비밀번호 까지 일치하면 콜백 함수에 찾은 유저 정보를 전달
          if (result !== null) {
            cb(null, result);
          } else {
            // 각각 상황에 맞는 에러 메세지 전달
            cb(null, false, { message: '비밀번호가 다릅니다.' });
          }
        } else {
          cb(null, false, { message: '해당 id 가 없습니다.' });
        }
      }
```

```
passport.use(
  new LocalStrategy(
    {
      usernameField: 'id',
      passwordField: 'password',
    },
    async (id, password, cb) => {
      const client = await mongoClient.connect();
      const userCursor = client.db('kdt1').collection('users');
      const idResult = await userCursor.findOne({ id });
      if (idResult !== null) {
        const result = await userCursor.findOne({
          id,
          password,
        });
        if (result !== null) {
          cb(null, result);
        } else {
          cb(null, false, { message: '비밀번호가 다릅니다.' });
        }
      } else {
        cb(null, false, { message: '해당 id 가 없습니다.' });
      }
    }
  )
);
```

Passport, serialzeUser

- Passport 도 session 을 사용하여 로그인 여부를 판단합니다
- 위에서 Callback 함수에 담긴 내용은 serialzeUser 메소드가 받아서 세션을 만들어 전달 받은 유저 정보를 세션에 저장합니다
- 해당 세션은 req.user 로 접근하셔 정보를 확인 할 수 있습니다
- 콜백 함수에 의해 user.id 부분이 세션에 저장 되는 코드!

```
passport.serializeUser((user, cb) => {
  cb(null, user.id);
});
```

Passport, deserializeUser

- Passport 는 더욱 까다롭게 로그인 여부를 판단하기 때문에 로그인한 사용자가 다른 페이지에 접속하려고 할 때에도 계속 인증을 요구 합니다
- 해당 기능을 deserializeUser 가 수행
- ssession 에 저장된 사용자 정보를 통해서 DB에 사용자가 실제로 있는지 꾸준히 확인하는 과정을 거칩니다

```
passport.deserializeUser(async (id, cb) => {
  const client = await mongoClient.connect();
  const userCursor = client.db('kdt1').collection('users');
  const result = await userCursor.findOne({ id });
  if (result) cb(null, result);
});
```

PASSPORT

로그인 구현

Passport 로그인

- Passport 로그인은 authenticate 메소드가 처리해 줍니다!

```
passport.authenticate('local', (err, user, info) => {
```

- 에러가 발생하면 err 매개변수에 에러 값이 담깁니다
- 로그인이 성공하면 user 에 전달한 user 정보가 담겨서 넘어옵니다
- 로그인이 실패하면 user 는 null 로 리턴되며, 전달한 메시지는 info 매개변수의 객체에 담겨서 전달이 됩니다

```
router.post('/', (req, res, next) => {
  passport.authenticate('local', (err, user, info) => {
    if (err) next(err);
    if (!user) {
      return res.send(
        `${info.message}<br><a href="/login">로그인 페이지로 이동</a>`
      );
    }
    req.logIn(user, (err) => {
      if (err) next(err);
      res.redirect('/board');
    });
  })(req, res, next);
});
```

PASSPORT

로그아웃 구현

Passport 로그 아웃

- Passport 로그 아웃은 req.logout 메소드를 사용합니다

```
router.get('/logout', (req, res, next) => {
  req.logout((err) => {
    if (err) {
      return next(err);
    }
    return res.redirect('/');
  });
});
```

로그인 여부에 따른 게시판 서비스 변경

Passport 로 로그인 했으므로 기능을 변경

- Passport 에 의해 Login 된 유저 Session 은 req.user 에 담기게 되므로 isLogin 함수에 조건을 추가해 줍니다!

```
function isLogin(req, res, next) {
  if (req.session.login || req.user) {
    next();
  } else {
    res.send('로그인 해주세요.<br><a href="/login">로그인 페이지로 이동</a>');
  }
}
```

Passport 로 로그인 했으므로 기능을 변경

- 그리고 이제 회원 아이디 값이 req.session.userId 가 아니라 req.user.id 에 있으므로 해당 부분도 수정해 줍니다
- 이 정보가 들어가는 곳은 게시판을 처음 렌더링 할 때 사용자 정보를 받는 부분, 글을 쓸 때 작성자 정보를 받는 부분 2곳이 있습니다.

게시판 렌더링 부분 수정

```
router.get('/', isLogin, async (req, res) => {
  const client = await mongoClient.connect();
  const cursor = client.db('kdt1').collection('board');
  const ARTICLE = await cursor.find({}).toArray();

  const articleLen = ARTICLE.length;
  res.render('board', {
    ARTICLE,
    articleCounts: articleLen,
    userId: req.session.userId ? req.session.userId : req.user.id,
  });
});
```

게시판 글쓰기 부분 수정

```
router.post('/', isLogin, async (req, res) => {
  if (req.body) {
    if (req.body.title && req.body.content) {
      const newArticle = {
        id: req.session.userId ? req.session.userId : req.user.id,
        title: req.body.title,
        content: req.body.content,
      };

      const client = await mongoClient.connect();
      const cursor = client.db('kdt1').collection('board');
      await cursor.insertOne(newArticle);
      res.redirect('/board');
    } else {
      const err = new Error('요청 이상');
      err.statusCode = 404;
      throw err;
    }
  } else {
    const err = new Error('요청에 데이터가 없습니다');
    err.statusCode = 404;
    throw err;
  }
```

서버 코드 정리

서버 코드 정리

- 메인이 되는 서버는 여러 작업자가 같이 수정하는 경우가 많으므로 LocalStrategy 같은 코드가 밖에 있으면 서로 협업이 어려워 집니다
- LocalStrategy 도 모듈화 시켜봅시다
- localStrategy.js 파일을 생성합니다!

서버 코드 정리

- localStrategy.js 파일에 passport, LocalStrategy 모듈을 불러 옵니다.
- 그리고 module.exports 를 하나의 익명 함수로 묶어서 서버 코드에 있던 LocalStrategy 를 묶어서 처리해 줍니다
- serialzeUser, deserialzeUser 도 같이 묶어서 처리해 줍니다.

```
const passport = require('passport');
const LocalStrategy = require('passport-
local').Strategy;

const mongoClient = require('./mongo');
```

```
module.exports = () => {
  passport.use(
    new LocalStrategy(
      {
        usernameField: 'id',
        passwordField: 'password',
      },
      async (id, password, cb) => {
        const client = await mongoClient.connect();
        const userCursor = client.db('kdt1').collection('users');
        const idResult = await userCursor.findOne({ id });
        if (idResult !== null) {
          const result = await userCursor.findOne({
            id,
            password,
          });
          if (result !== null) {
            cb(null, result);
          } else {
            cb(null, false, { message: '비밀번호가 다릅니다.' });
          }
        } else {
          cb(null, false, { message: '해당 id 가 없습니다.' });
        }
      }
    )
  );
```

```
passport.serializeUser((user, cb) => {
    cb(null, user.id);
  });

  passport.deserializeUser(async (id, cb) => {
    const client = await mongoClient.connect();
    const userCursor = client.db('kdt1').collection('users');
    const result = await userCursor.findOne({ id });
    if (result) cb(null, result);
  });
};
```

서버 코드 수정

- localStrategy.js 파일을 모듈로 호출!
- locaStrategy(); 를 수행하여 모듈 코드를 서버 코드에 적용

```
const localStrategy = require('./routes/localStrategy’);
localStrategy();
```

DOTENV

.ENV

DOTENV, 중요 정보를 관리하는 모듈

- DOTENV 는 중요한 정보(서버 접속 정보 등등)를 외부 코드에서 확인이 불가능 하도록 도와주는 모듈입니다!
- 일단 설치 합시다
- Npm i dotenv -s
- 모듈 호출하기

```
require('dotenv').config();
```

DOTENV, 중요 정보를 관리하는 모듈

- .env 파일을 최상단 폴더에 만들기
- 중요한 정보를 .env 파일에 저장

```
PORT = 4000
DB_URI =
mongodb+srv://tetz:qwer1234@cluster0.sdiakr0.mongodb.net/?retryWrites=true&w=majority
```

- 해당 정보가 필요한 곳에서 process.env.저장명 으로 사용

```
const uri = process.env.DB_URI;
```

DOTENV, 중요 정보를 관리하는 모듈

- 정말 중요한 정보만 저장이 되는 파일이므로 github 에 올리면 안되겠죠?
- .gitignore 에 추가해 줍니다!

```
node_modules/
.env
```

- 따라서 해당 파일은 직접 업로드 하면서 사용하시면 됩니다!


수고하셨습니다!