Hello,

KDT 웹 개발자 양성 프로젝트

5기!

with

게시판에

DB 적용하기

게시판 서비스를 위한 TABLE 생성

```sql
CREATE TABLE board (
    `ID_PK` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    `TITLE` VARCHAR(100) NOT NULL,
    `CONTENT` VARCHAR(300) NOT NULL,
    `REGISTER_DATE` DATETIME DEFAULT CURRENT_TIMESTAMP,
    `UPDATE_DATE` DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

테스트를 위한 기본 데이터 삽입

```
 9 •    INSERT INTO board (TITLE, CONTENT) VALUES ('제목1', '테스트 컨텐츠 입니다!');
10 •    INSERT INTO board (TITLE, CONTENT) VALUES ('제목1', '테스트 컨텐츠 입니다!');
```

Result Grid | Filter Rows: | Edit: | Export/Import:

	ID_PK	TITLE	CONTENT	REGISTER_DATE	UPDATE_DATE
▶	1	제목1	테스트 컨텐츠 입니다!	2022-11-24 14:27:39	2022-11-24 14:27:39
	2	제목2	테스트 컨텐츠 입니다!	2022-11-24 14:27:41	2022-11-24 14:27:41
*	NULL	NULL	NULL	NULL	NULL

DB 버전

게시판 작업 시작!

게시판 용 컨트롤러 작성

- Controllers 폴더에 boardController.js 작성

```
const connection = require('./dbConnect');

const boardDB = {
  getAllArticles: (cb) => {
    connection.query('SELECT * FROM mydb.board;', (err, data) => {
      if (err) throw err;
      console.log(data);
      cb(data);
    });
  },
};

module.exports = boardDB;
```

Controllers/boardContoroller.js

기본 페이지 작업!

- MySQL 로 부터 데이터를 받아서 EJS 에 전달하는 라우터 작성

```
router.get('/', (req, res) => {
  boardDB.getAllArticles((data) => {
    const ARTICLE = data;
    const articleCounts = ARTICLE.length;
    res.render('db_board', { ARTICLE, articleCounts });
  });
});
```

routes/dbBoard.js

메인 서버에 라우터 등록

```
const mainRouter = require('./routes');
const userRouter = require('./routes/users');
const boardRouter = require('./routes/board');
const dbRouter = require('./routes/db');
const dbBoardRouter = require('./routes/dbBoard');

app.use('/', mainRouter);
app.use('/users', userRouter);
app.use('/board', boardRouter);
app.use('/db', dbRouter);
app.use('/dbBoard', dbBoardRouter);
```

dbBoard 용 EJS 파일 생성!

- 기존 board.ejs 파일을 카피해서 쓰기 → db_board.ejs 생성
- 글쓰기 모드로 이동 href 및 글 수정 모드 이동 href 도 수정!

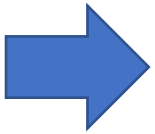
```
<span>현재 등록 글 :   <%= articleCounts %></span>
<a class="btn red" href="/dbBoard/write">글쓰기</a>
```

```
<div class="foot">
  <a class="btn orange" href="/dbBoard/modify/<%= ARTICLE[i].TITLE %>">수정</a>
  <a class="btn blue" href="#" onclick="deleteArticle('<%= ARTICLE[i].TITLE %>')">삭제</a>
</div>
```

EJS 파일 변경

- EJS 파일을 변경!

```
<div class="title">
    <%= ARTICLE[i].title %>
</div>
<div class="content">
    <%= ARTICLE[i].content %>
</div>
```



```
<div class="title">
    <%= ARTICLE[i].TITLE %>
</div>
<div class="content">
    <%= ARTICLE[i].CONTENT %>
</div>
```

글 쓰기 기능 구현

새로운 글을 DB에 등록하는 컨트롤러 만들기!

- 데이터는 잘 들어오므로 해당 데이터를 DB에 등록하는 컨트롤러 작업!

```
writeArticle: (newArticle, cb) => {
    connection.query(
      `INSERT INTO mydb.board (TITLE, CONTENT) VALUES ('${newArticle.title}', '${newArticle.content}')`,
      (err, data) => {
        if (err) throw err;
        cb(data);
      }
    );
  },
```

Controllers/boardContoroller.js

컨트롤러를 사용하도록 라우터 수정

```
router.post('/write', (req, res) => {
  if (req.body.title && req.body.content) {
    boardDB.writeArticle(req.body, (data) => {
      console.log(data);
      if (data.affectedRows >= 1) {
        res.redirect('/dbBoard');
      } else {
        const err = new Error('글 쓰기 실패');
        throw err;
      }
    });
  } else {
    const err = new Error('글 제목 또는 내용이 없습니다!');
    throw err;
  }
});
```

routes/dbBoard.js

글 수정하기 기능 구현

ID_PK 값으로 특정 글을 찾는 컨트롤러 만들기

```
getArticle: (id, cb) => {
  connection.query(
    `SELECT * FROM mydb.board WHERE ID_PK = ${id};`,
    (err, data) => {
      if (err) throw err;
      cb(data);
    }
  );
},
```

Controllers/boardContoroller.js

글 수정 모드로 이동 라우터 변경!

```
router.get('/modify/:id', (req, res) => {
  boardDB.getArticle(req.params.id, (data) => {
    if (data.length > 0) {
      res.render('db_board_modify', { selectedArticle: data[0] });
    }
  });
});
```

routes/dbBoard.js

게시글 목록 페이지에서 수정 모드로 이동

```
<div class="foot">
  <a class="btn orange" href="/dbBoard/modify/<%= ARTICLE[i].ID_PK %>">수정</a>
  <a class="btn blue" href="#" onclick="deleteArticle('<%= ARTICLE[i].TITLE %>')">삭제</a>
</div>
```

- 게시글은 이제 ID_PK 값으로 찾으므로 전달하는 값을 제목에서 ID_PK 로 수정!
- 데이터도 title, content 소문자에서 대문자로 변경!

Write Mode

제목

제목2

내용

테스트 컨텐츠 입니다!

글 수정하기

글 수정 페이지에서 전달 된 값으로 DB 수정!

- ID_PK 값을 받아서 해당 ID를 가지는 DB를 UPDATE 해주면 됩니다!
- 컨트롤러 작업!

```
modifyArticle: (id, modifyArticle, cb) => {
    connection.query(
      `UPDATE mydb.board SET TITLE = '${modifyArticle.title}', CONTENT =
'${modifyArticle.content}' WHERE ID_PK = ${id};`,
      (err, data) => {
        if (err) throw err;
        cb(data);
      }
    );
  },
```

Controllers/boardContoroller.js

글을 수정하는 라우터 생성!

```
router.post('/modify/:id', (req, res) => {
  if (req.body.title && req.body.content) {
    boardDB.modifyArticle(req.params.id, req.body, (data) => {
      if (data.affectedRows >= 1) {
        res.redirect('/dbBoard');
      } else {
        const err = new Error('글 수정 실패');
        throw err;
      }
    });
  } else {
    const err = new Error('글 제목 또는 내용이 없습니다!');
    throw err;
  }
});
```

routes/dbBoard.js

글 삭제하기 기능 구현

컨트롤러

```
  deleteArticle: (id, cb) => {
    connection.query(
      `DELETE FROM mydb1.board WHERE ID_PK = ${id};`,
      (err, data) => {
        if (err) throw err;
        cb(data);
      }
    );
  },
```

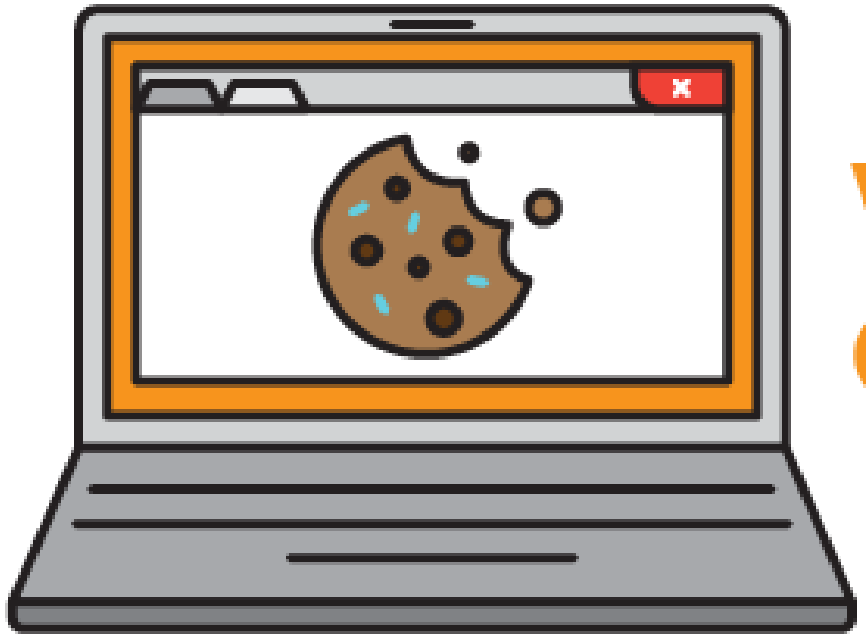
라우터

```
router.delete('/delete/:id', (req, res) => {
  boardDB.deleteArticle(req.params.id, (data) => {
    console.log(data);
    if (data.affectedRows >= 1) {
      res.send('삭제 완료!');
    } else {
      const err = new Error('글 삭제 실패');
      throw err;
    }
  });
});
```

프론트

```
<a class="btn blue" href="#" onclick="deleteArticle('<%=
ARTICLE[i].ID_PK %>')">삭제</a>

<script>
  function deleteArticle(id) {
    fetch(`dbBoard/delete/${id}`, {
      method: 'delete',
      headers: {
        'Content-type': 'application/json'
      },
    }).then((res) => {
      location.href = '/dbBoard';
    })
  }
</script>
```

Cookie!?



WEBSITE COOKIES

https://www.youtube.com/watch?v=tosLBcAX1vk&t=278s

프론트에서 쿠키 사용하기

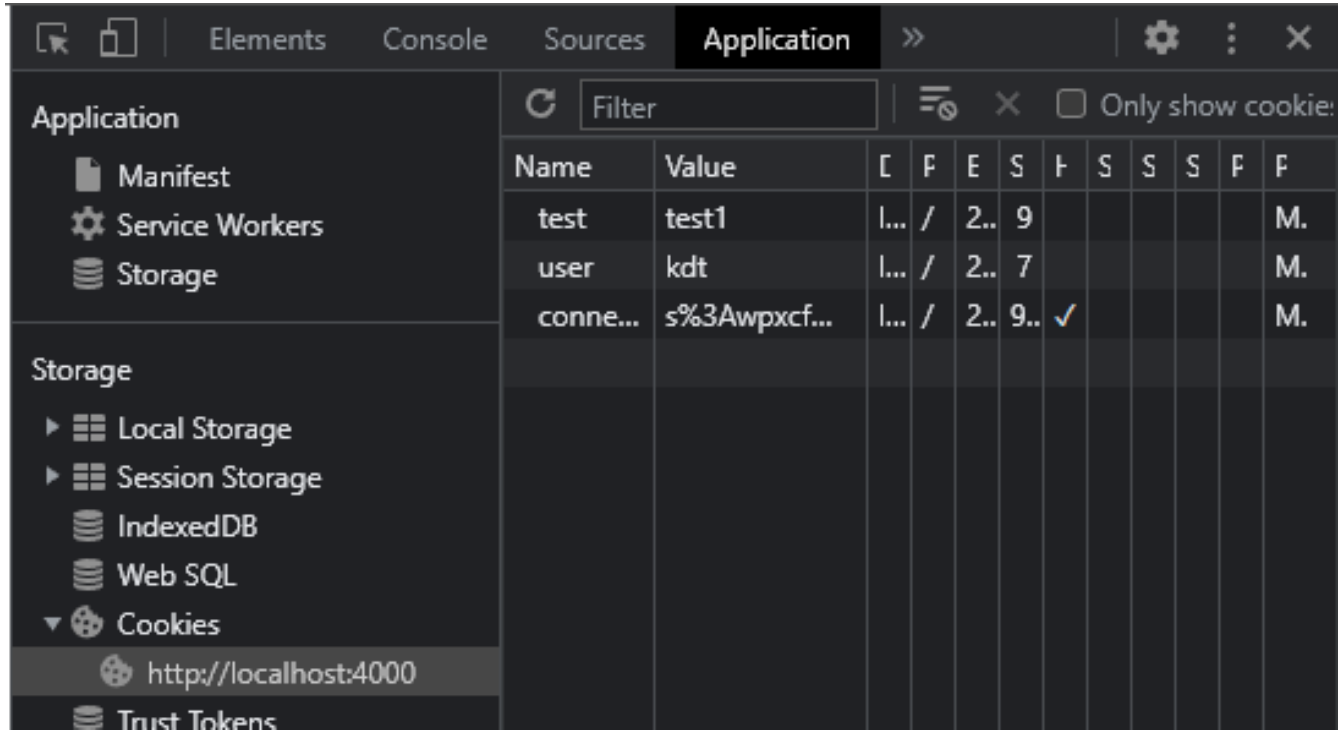
- Document.cookie 로 바로 만들어서 사용하면 됩니다

```
<script>
    console.log(document.cookie);
    document.cookie = "user=kdt; expires=26 Nov 2023 13:00:00 GMT; path=/";
    document.cookie = "test=test1; expires=26 Nov 2023 13:00:00 GMT; path=/";
    console.log(document.cookie);
  </script>
```

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

브라우저에서 쿠키 확인하기

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



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

- 쿠키를 만드려면 res.cookie('쿠키 이름', '데이터', '옵션 객체'); 를 써서 쿠키를 구우면 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 JS 로 해당 쿠키를 읽으려 하면 웹브라우저가 이를 차단

```
router.get('/', (req, res) => {
  res.cookie('alert', true, {
    expires: new Date(Date.now() + 1000 * 60),
    httpOnly: true,
  });
  console.log(req.cookies);
  res.render('index');
});
```

```
서버는 4000번 포트에서 실행 중입니다!
[Object: null prototype] {}
{ alert: 'true' }
```

처음 쿠키 발행 시에는
브라우저에만 있으므로
빈 객체가 리턴 {}

다시 페이지를 새로고침하면
브라우저의 쿠키를 서버가 받아
쿠키 값이 출력


지금 시작합니다

백엔트 쿠키

기능들!

쿠키의 값을 전달

- 프론트에서 쿠키의 값을 잘라서 쓰기 싫다면?
- 백엔드에서 쿠키의 값을 구해서 프론트에 전달해도 됩니다!

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

쿠키의 값을 전달

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

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

쿠키의 최대 나이? 정하기!

- 쿠키의 생존은 expires 로 정해도 되지만 maxAge 로도 설정이 가능합니다!
- maxAge 는 쿠키의 생성 시간을 기준으로 밀리 세컨드 단위로 생존 시간을 결정합니다!

```
router.get('/cookie', (req, res) => {
  res.cookie('cookie', true, {
    maxAge: 1000 * 60,
    httpOnly: false,
  });
  res.send('쿠키 굽기 성공!');
});
```

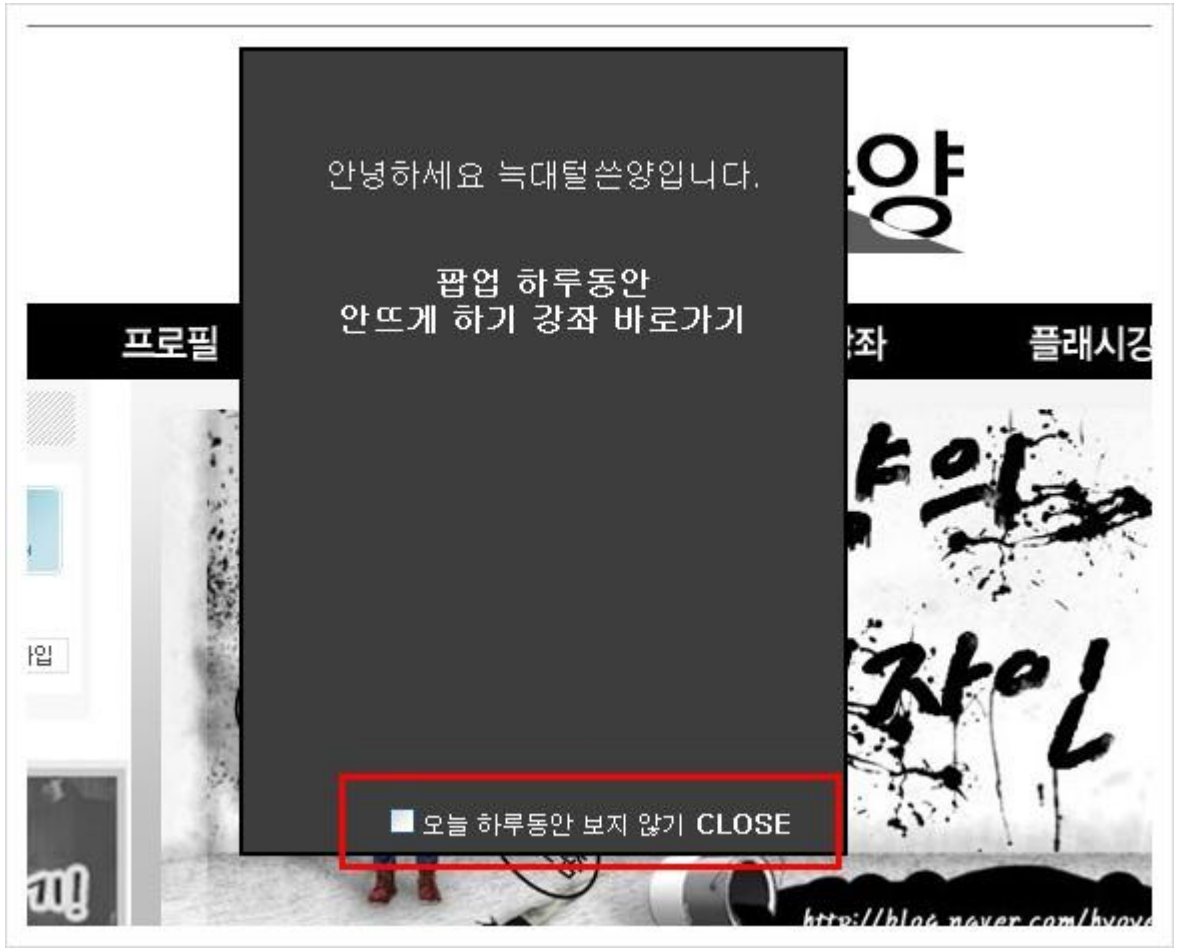
쿠키 삭제하기!

- 쿠키를 삭제하는 방법은 res.clearCookie('쿠키 이름') 을 사용하면 됩니다!

```
router.get('/cookie', (req, res) => {
  res.cookie('cookie', true, {
    maxAge: 1000 * 60,
    httpOnly: false,
  });
  res.clearCookie('cookie');
  res.send('쿠키 굽기 성공!');
});
```

쿠키

활용하기



ID 아이디를 입력해주세요.

PW 비밀번호를 입력해주세요.

로그인

아이디 저장 아이디 찾기 비밀번호 찾기

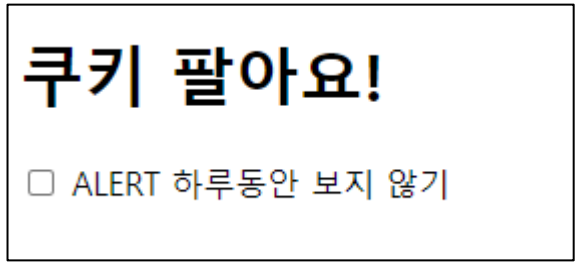
쿠키를 활용하는 cookie.ejs 페이지 작성

- 쿠키가 없으면 alert 창을 띄우는 cookie.ejs 페이지를 만들어 봅시다!

```
<body>
  <h1>쿠키 팔아요!</h1>
  <input type="checkbox" name="check" id="check" />
  <label>ALERT 하루동안 보지 않기</label>

  <script>
    if (!document.cookie) alert('쿠키 팔아요!');
  </script>
</body>
```

실습, 쿠키 활용하기

- 체크 박스를 체크하면 백엔드에서 쿠키를 발행하여 더 이상 alert 창이 뜨지 않도록 기능을 만들어 봅시다!



- CheckBox 를 클릭하면 GET 방식으로 /cookie/cook 이라는 주소 요청을 보내게 되고, 백엔드는 이 요청을 받으면 alert=true 값을 가지는 쿠키를 발행해 줍니다!

실습, 쿠키 활용하기

- 쿠키는 최대 5초 동안 유지가 되며, 5초가 지나면 자동으로 사라지도록 만들어 주세요!
- 프론트에서 `if (!document.cookie) alert('쿠키 팔아요!');`

  코드를 통해 alert 창을 컨트롤 하므로 httpOnly 옵션은 false 를 주세요!

실습, 쿠키 활용하기

- 프론트 코드입니다!

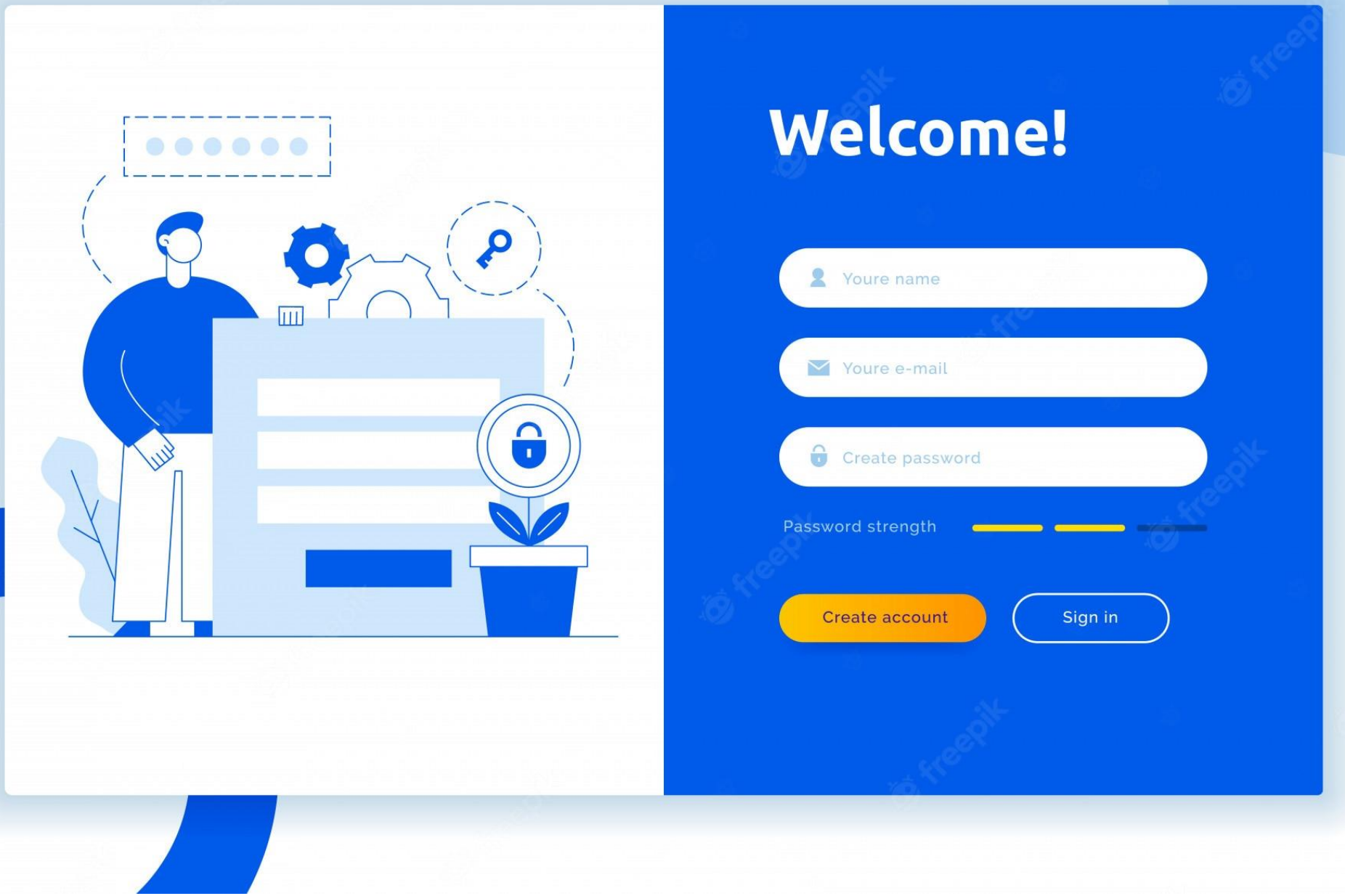
```
<script>
  if (!document.cookie) alert('쿠키 팔아요!');

  const checkBox = document.getElementById('check');
  checkBox.addEventListener('click', () => {
    // 백엔드에 쿠키 발행을 요청하는 코드 작성하기
  });
</script>
```

실습, 쿠키 활용하기

- 백엔드 코드입니다!

```
const express = require('express');

const router = express.Router();

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

router.get('/cook', (req, res) => {
  // 쿠키 발행 코드 작성하기!
});

module.exports = router;
```


LOGIN


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

회원테이블

만들기!

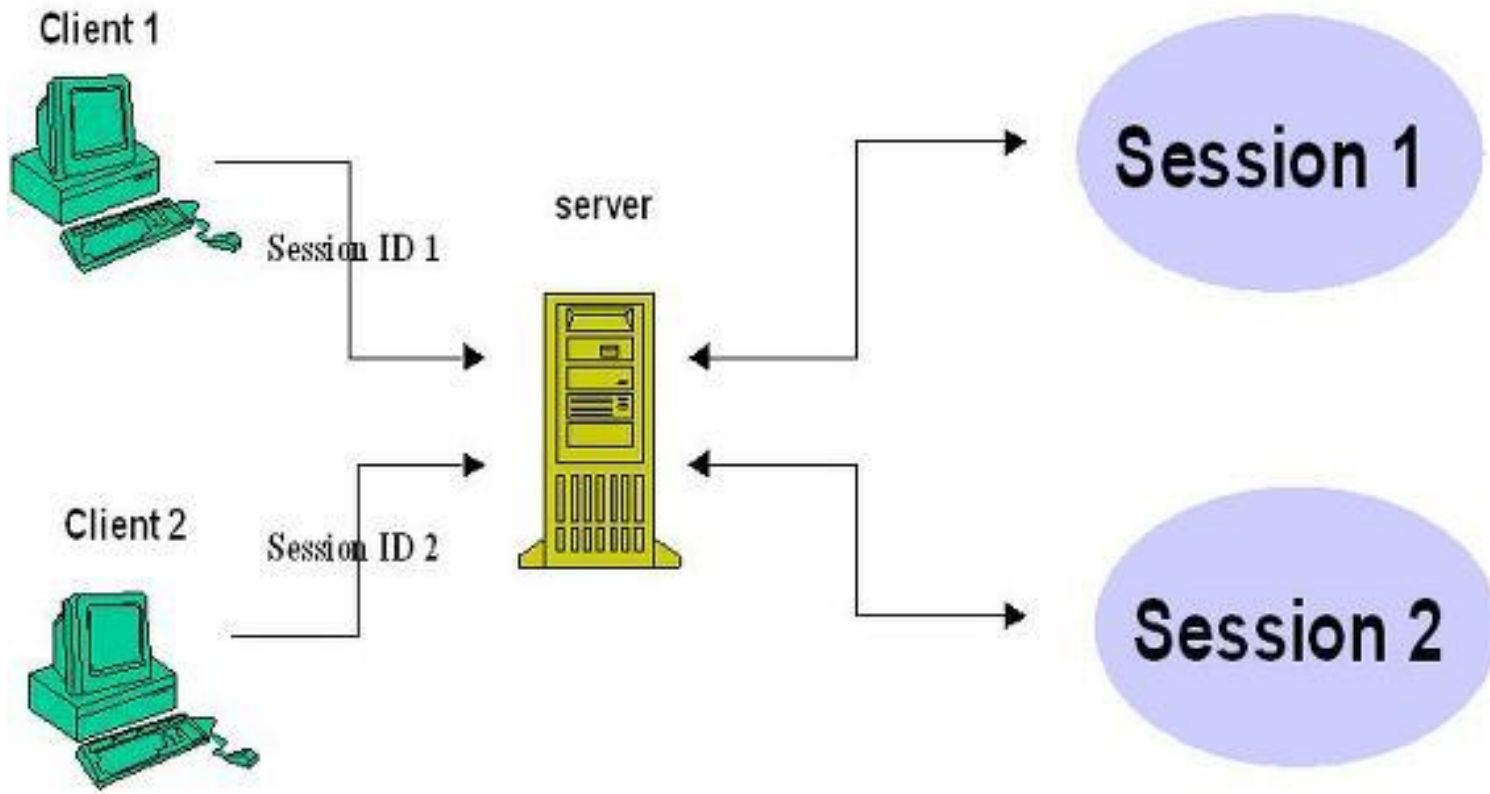
```sql
CREATE TABLE user (
    `ID_PK` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    `USERID` VARCHAR(20) NOT NULL UNIQUE,
    `PASSWORD` VARCHAR(20) NOT NULL,
    `REGISTER_TIME` DATETIME DEFAULT CURRENT_TIMESTAMP,
    `UPDATE_TIME` DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

```sql
SELECT * FROM user;
INSERT INTO user (USERID, PASSWORD) VALUES ('tetz', '11');
INSERT INTO user (USERID, PASSWORD) VALUES ('pororo', '11');
```

	ID_PK	USERID	PASSWORD	REGISTER_TIME	UPDATE_TIME
▶	1	tetz	11	2022-11-26 09:28:54	2022-11-26 09:28:54
	3	pororo	11	2022-11-26 09:29:54	2022-11-26 09:29:54
*	NULL	NULL	NULL	NULL	NULL

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
- 부트스트랩을 이용하여 간단하게 만든 페이지 입니다!
- login.ejs / register.ejs 파일로 추가!

회원 가입 페이지

https://github.com/xenosign/git_4th_backend/blob/main/views/login.ejs

로그인 페이지

https://github.com/xenosign/git_4th_backend/blob/main/views/posts.ejs

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-2 text-center"><a href="/login">로그인 바로가기</a></h2>
    <h2 class="mt-2 text-center"><a href="/register">회원가입 바로가기</a></h2>
    <h2 class="mt-2 text-center"><a href="/dbBoard">게시판 바로가기</a></h2>
</div>
```

Session

모듈 추가하기

세션 모듈 추가하기

- 먼저 express-session 모듈 부터 설치 합시다
  - npm i express-session
- 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 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
- ‘uesr’ 테이블에서 회원 데이터를 관리
- Req.body 에 담겨있는 ID 가 DB 에 존재하는지 확인하고, 존재할 경우 id 중복 안내 → 회원 가입 페이지로 이동
- ID가 중복되지 않을 경우, DB에 새로운 회원을 등록하고 회원 가입 성공 메시지를 출력 → 로그인 페이지로 이동

중복 회원 찾기 컨트롤러 작성

- User 테이블에서 중복 된 회원이 있는지 검사하는 컨트롤러 부터 만들어 봅시다!
- 중복이 없어야만 회원 가입이 가능하므로, 해당 기능을 만들고 난 뒤 실제 회원 가입 컨트롤러를 만들어 주시면 됩니다!

```
userCheck: (userId, cb) => {
    userDB.query(
      `SELECT * FROM mydb1.user WHERE USERID = '${userId}';`,
      (err, data) => {
        if (err) throw err;
        console.log(data);
        cb(data);
      },
    );
  },
```

회원 가입 컨트롤러 작성

- 중복이 없다면 이제 회원 가입을 하는 컨트롤러를 만들어 주시면 됩니다!

```
registerUser: (newUser, cb) => {
    userDB.query(
      `INSERT INTO mydb1.user (USERID, PASSWORD) VALUES ('${newUser.id}', '${newUser.password}');`,
      (err, data) => {
        if (err) throw err;
        console.log(data);
        cb(data);
      },
    );
  },
```

회원 가입 라우터 작성

- 먼저 중복 여부를 체크한 다음, 중복 회원이 없을 경우 회원 가입을 진행 시키면 됩니다!

```
router.post('/', (req, res) => {
  userDB.userCheck(req.body.id, (data) => {
    if (data.length === 0) {
      userDB.registerUser(req.body, (result) => {
        if (result.affectedRows >= 1) {
          res.status(200);
          res.send(
            '회원 가입 성공!<br><a href="/login">로그인 페이지로 이동</a>',
          );
        } else {
          res.status(500);
          res.send(
            '회원 가입 문제 발생.<br><a href="/register">회원가입 페이지로 이동</a>',
          );
        }
      });
    } else {
      res.status(400);
      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 로그인 기능을 모듈화

```
const express = require('express');
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

- 먼저 입력 받은 ID 가 DB 에 존재 하는지를 체크! → 이미 만든 컨트롤러가 있네요!? userCheck 를 사용!
- DB에 입력 받은 ID가 존재하면 입력받은 password 와 DB에 있는 회원의 password 가 동일한지를 체크하고 로그인 처리를 하면 됩니다!
- 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('/', (req, res) => {
  userDB.userCheck(req.body.id, (data) => {
    if (data.length > 0) {
      if (data[0].PASSWORD === req.body.password) {
        req.session.login = true;
        req.session.userId = req.body.id;
        res.status(200);
        res.redirect('/dbBoard');
      } else {
        res.status(400);
        res.send(
          '비밀번호가 다릅니다.<br><a href="/login">로그인 페이지로 이동</a>',
        );
      }
    } else {
      res.status(400);
      res.send(
        '해당 id 가 존재하지 않습니다.<br><a href="/register">회원가입 페이지로 이동</a>',
      );
    }
  });
});
```

로그 아웃 구현

로그 아웃 버튼 만들기

- dbBoard.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) => {
    if (err) throw 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('/', (req, res) => {
  if (req.session.login) {
    db.getAllArticles((data) => {
      const ARTICLE = data;
      const articleCounts = ARTICLE.length;
      res.render('dbBoard', {
        ARTICLE,
        articleCounts,
        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

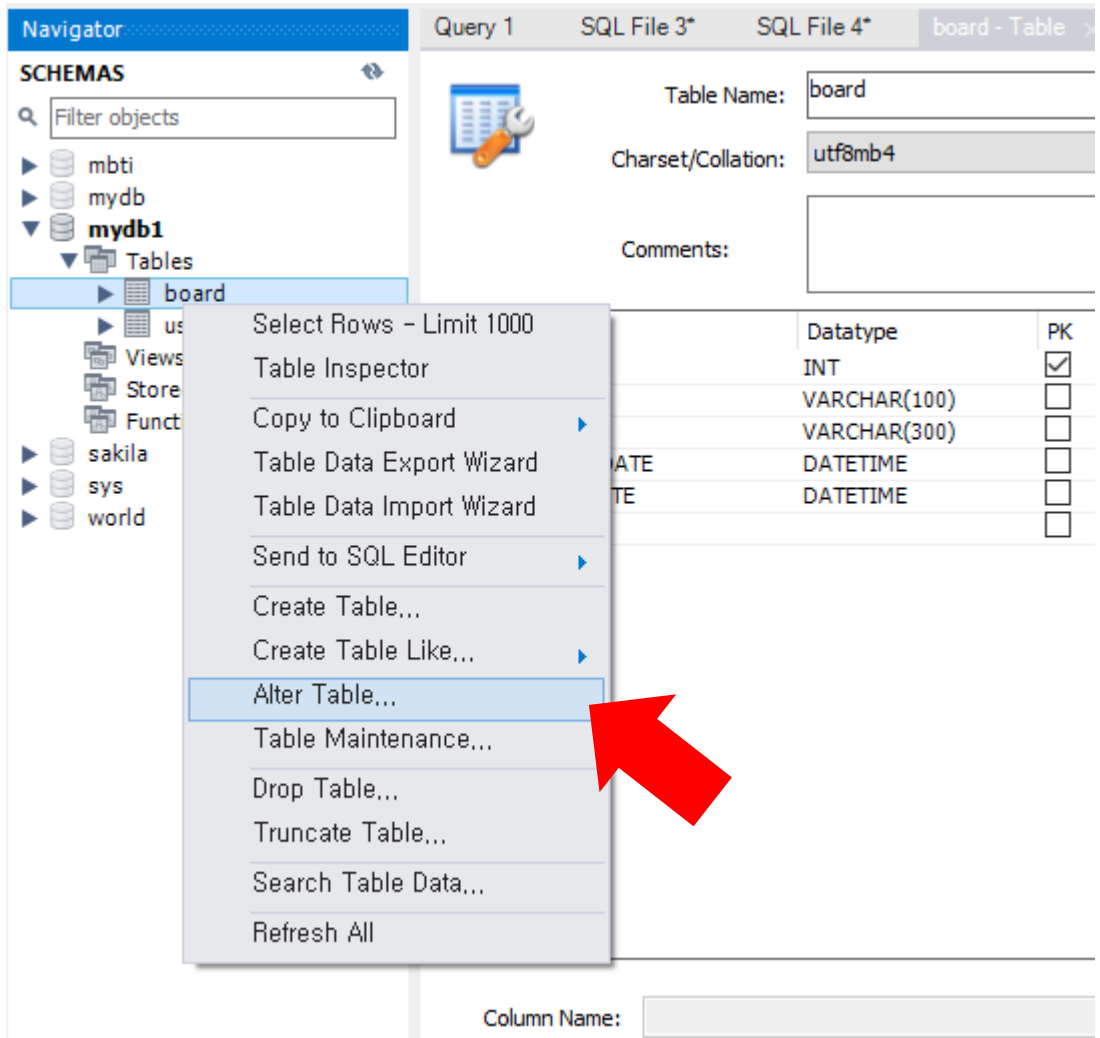
```
function isLogin(req, res, next) {
  if (req.session.login) {
    next();
  } else {
    res.send('로그인 해주세요.<br><a href="/login">로그인 페이지로 이동</a>');
  }
}

router.get('/', isLogin, (req, res) => {
  db.getAllArticles((data) => {
    const ARTICLE = data;
    const articleCounts = ARTICLE.length;
    res.render('dbBoard', {
      ARTICLE,
      articleCounts,
      userId: req.session.userId,
    });
  });
});
```

게시글에

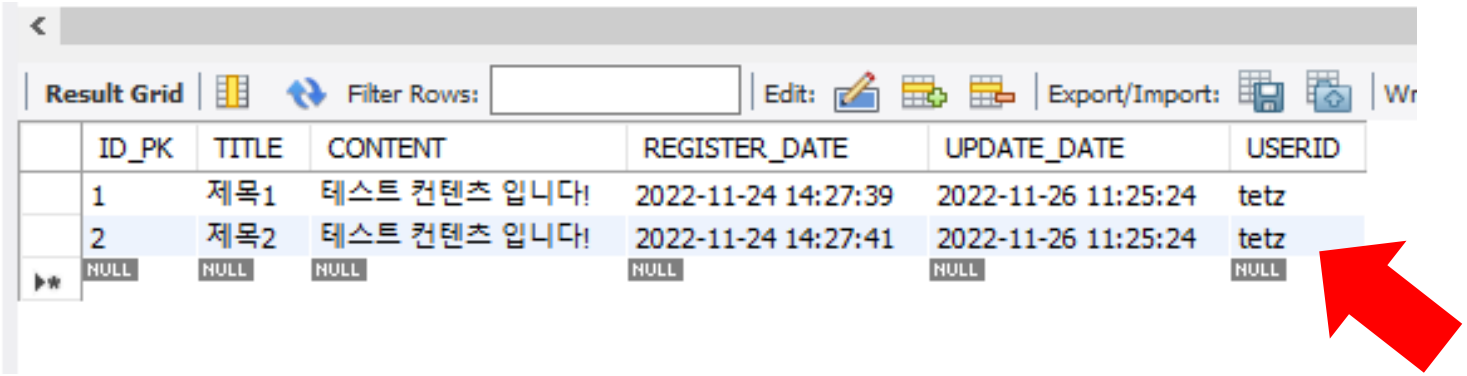
작성자 id 정보 추가

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!
- Workbench 에 가서 일단 추가 해봅시다!


Navigator
SCHEMAS
Filter objects
mbti
mydb
mydb1
Tables
board
Views
Store
Funct
sakila
sys
world
Query 1
SQL File 3*
SQL File 4*
board - Table
Table Name:
board
Charset/Collation:
utf8mb4
Comments:
Datatype
PK
INT
VARCHAR(100)
VARCHAR(300)
DATETIME
DATETIME
Select Rows - Limit 1000
Table Inspector
Copy to Clipboard
Table Data Export Wizard
Table Data Import Wizard
Send to SQL Editor
Create Table...
Create Table Like...
Alter Table...
Table Maintenance...
Drop Table...
Truncate Table...
Search Table Data...
Refresh All
Column Name:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
ID_PK	INT	☑	☑	☐	☐	☐	☐	☑	☐	
TITLE	VARCHAR(100)	☐	☑	☐	☐	☐	☐	☐	☐	
CONTENT	VARCHAR(300)	☐	☑	☐	☐	☐	☐	☐	☐	
REGISTER_DATE	DATETIME	☐	☐	☐	☐	☐	☐	☐	☐	CURRENT_TIMESTAMP
UPDATE_DATE	DATETIME	☐	☐	☐	☐	☐	☐	☐	☐	CURRENT_TIMESTAMP ON...
USERID	VARCHAR(20)	☐	☑	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	☐	



ID_PK	TITLE	CONTENT	REGISTER_DATE	UPDATE_DATE	USERID
1	제목1	테스트 컨텐츠 입니다!	2022-11-24 14:27:39	2022-11-26 11:25:24	tetz
2	제목2	테스트 컨텐츠 입니다!	2022-11-24 14:27:41	2022-11-26 11:25:24	tetz
NULL	NULL	NULL	NULL	NULL	NULL

Board.ejs

파일 수정

작성자 표시

- 제목 위에 작성자의 id 를 보여주는 부분 추가

```
<li>
  <div class="author">
    작성자 : <%= ARTICLE[i].USERID %>
  </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].USERID === userId) { %>
  <a class="btn orange" href="dbBoard/modify/<%= ARTICLE[i].ID_PK %>">수정</a>
  <a class="btn blue" href="#" onclick="deleteArticle('<%= ARTICLE[i].ID_PK %>')">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('/write', isLogin, (req, res) => {
  if (req.body.title && req.body.content) {
    const newArticle = {
      id: req.session.userId,
      title: req.body.title,
      content: req.body.content,
    };
    boardDB.writeArticle(newArticle, (data) => {
      if (data.affectedRows >= 1) {
        res.status(200);
        res.redirect('/dbBoard');
      } else {
        const err = new Error('글 쓰기 실패');
        throw err;
      }
    });
  } else {
    const err = new Error('글 제목 또는 내용이 없습니다!');
    throw err;
  }
});
```

```
  writeArticle: (newArticle, cb) => {
    boardDB.query(
      `INSERT INTO mydb1.board (USERID, TITLE, CONTENT) VALUES ('${newArticle.id}',
'${newArticle.title}', '${newArticle.content}')`,
      (err, data) => {
        if (err) throw err;
        cb(data);
      },
    );
  },
```

게시글 수정 기능

수정

게시글 수정 페이지로 이동

- 로그인 상태가 아니면 해당 페이지로 이동이 안되도록 설정

```
router.get('/modify/:id', isLogin, (req, res) => {
  boardDB.getArticle(req.params.id, (data) => {
    if (data.length > 0) {
      res.render('dbBoard_modify', { selectedArticle: data[0] });
    }
  });
});
```

게시글 수정 기능

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

```
router.post('/modify/:id', isLogin, (req, res) => {
  if (req.body.title && req.body.content) {
    db.modifyArticle(req.params.id, req.body, (data) => {
      console.log(data);
      if (data.affectedRows >= 1) {
        res.redirect('/dbBoard');
      } else {
        const err = new Error('글 수정 실패');
        throw err;
      }
    });
  } else {
    const err = new Error('글 제목 또는 내용이 없습니다!');
    throw err;
  }
});
```

게시글 삭제 기능 수정

게시글 삭제 기능

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

```
router.delete('/delete/:id', isLogin, (req, res) => {
  db.deleteArticle(req.params.id, (data) => {
    console.log(data);
    if (data.affectedRows >= 1) {
      res.send('삭제 완료!');
    } else {
      const err = new Error('글 삭제 실패');
      throw err;
    }
  });
});
```

쿠키를 사용한

자동 로그인 구현

쿠키를 사용한 자동 로그인

- 실습에서 하루동안 팝업을 뜨지 않게 한 것처럼, 로그 아웃 후 60초 동안은 세션이 없어도 자동 로그인이 되도록 구현해 봅시다
- 왜 쿠키를 쓸까요?
- 세션은 브라우저를 종료하면 사라지지만 쿠키는 만료일 까지 남아 있게 됩니다! 따라서, 쿠키를 이용해서 구현을 합니다.

쿠키를 사용한 자동 로그인

- 먼저 로그인을 하면 쿠키를 발행
  - 사용자 id 정보
  - 60초 의 expires 설정
  - httpOnly 옵션 켜기
  - Signed 옵션 켜기(사용자 ID 가 저장 되므로) → 서버의 Cookie-parser 에 암호화 키 설정 필요

```
app.use(cookieParser('tetz'));
```

```
router.post('/', (req, res) => {
  db.userCheck(req.body.id, (data) => {
    if (data.length > 0) {
      if (data[0].PASSWORD === req.body.password) {
        req.session.login = true;
        req.session.userId = req.body.id;
        // 쿠키 발행
        res.cookie('user', req.body.id, {
          maxAge: 1000 * 10,
          httpOnly: true,
          signed: true,
        });
        res.redirect('/dbBoard');
      } else {
        res.status(400);
        res.send(
          '비밀번호가 다릅니다.<br><a href="/login">로그인으로 이동</a>',
        );
      }
    } else {
      res.status(400);
      res.send(
        '회원 ID를 찾을 수 없습니다.<br><a href="/login">로그인으로 이동</a>',
      );
    }
  });
});
```

쿠키를 사용한 자동 로그인

- isLogin 함수에 쿠키에 의한 로그인 처리 기능 추가

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

쿠키가 정상 작동 하는지 테스트!

- 로그인 후, 브라우저를 종료 하고 다시 켠 다음 게시판 서비스로 접근하기!
- 특정 시간 이후 접근이 안되는지 확인!

로그아웃을 하면 쿠키도…

- 로그 아웃은 사용자가 로그 아웃을 하겠다는 의사를 밝힌 것이므로 쿠키도 같이 삭제가 되어야 합니다!
- 로그 아웃을 한 다음에 다른 사용자가 와서 접근 했을 때 로그인 처리가 되면 안되니까요!

```
// 로그 아웃 처리
router.get('/logout', (req, res) => {
  req.session.destroy((err) => {
    if (err) throw err;
    res.clearCookie('user');
    res.redirect('/');
  });
});
```

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
- 모듈 호출하기

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

DOTENV, 중요 정보를 관리하는 모듈

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

```
PORT = 4000
DB_USER = root
DB_PASSWORD = dlrladk
DB_DATABASE = mydb
```

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

```
const PORT = process.env.PORT;
```

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!

.gitignore 가 안되나요?

- Git 도 캐쉬를 사용하기 때문에 로컬에 해당 내용이 남아 있어서 그렇게 됩니다 → 깃 캐쉬를 삭제하고, 다시 푸쉬해 주시면 됩니다!
- git rm -r --cached .
- git add .
- git commit –m “clear git cache”
- git push --all


수고하셨습니다!