Hello,

KDT 웹 개발자 양성 프로젝트

5기!

with

MongoDB

Query 배우기

삽입

insertOne

- 하나의 도큐먼트를 삽입합니다

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertOne(
      {
        name: "pororo",
        age: 5,
      },
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        console.log(insertResult);
      }
    );
  });
});
```

```
{
  acknowledged: true,
  insertedId: new ObjectId("64185fa319e4137e9b9ccc6e")
}
```

```
_id: ObjectId('641860a7ce5bf85071433f3c')
name: "pororo"
age: 5
```

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        console.log(insertResult);
      }
    );
  });
});
```

```
{
  acknowledged: true,
  insertedCount: 3,
  insertedIds: {
    '0': new ObjectId("641861305e5008f364de434e"),
    '1': new ObjectId("641861305e5008f364de434f"),
    '2': new ObjectId("641861305e5008f364de4350")
  }
}
```

```
_id: ObjectId('641861305e5008f364de434e')
name: "pororo"
age: 5
```

```
_id: ObjectId('641861305e5008f364de434f')
name: "loopy"
age: 6
```

```
_id: ObjectId('641861305e5008f364de4350')
name: "crong"
age: 4
```

삭제

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        test.deleteOne({ name: "crong" },
        (deleteOneErr, deleteOneResult) => {
          console.log(deleteOneResult);
        });
      }
    );
  });
});
```

```
[nodemon] starting `node mongo.js`
{ acknowledged: true, deletedCount: 1 }
```

```
_id: ObjectId('641861e28815cb7f3d67f769')
name: "pororo"
age: 5
```

```
_id: ObjectId('641861e28815cb7f3d67f76a')
name: "loopy"
age: 6
```

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        test.deleteMany(
          { age: { $gte: 5 } },
          (deleteOneErr, deleteOneResult) => {
            console.log(deleteOneResult);
          }
        );
      }
    );
  });
});
```

```
{ acknowledged: true, deletedCount: 2 }
```

```
_id: ObjectId('6418623ed0a4aeec08e85f84')
name: "crong"
age: 4
```

수정

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
         test.updateOne(
          { name: "loopy" },
          { $set: { name: "루피" } },
          (updateErr, updateResult) => {
            if (updateErr) throw updateErr;
            console.log(updateResult);
          }
        );
      }
    );
  });});});
```

```
{
  acknowledged: true,
  modifiedCount: 1,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 1
}
```

```
_id: ObjectId('641863106d515e6c5c088460')
name: "pororo"
age: 5
```

```
_id: ObjectId('641863106d515e6c5c088461')
name: "루피"
age: 6
```

```
_id: ObjectId('641863106d515e6c5c088462')
name: "crong"
age: 4
```

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        test.updateMany(
          { age: { $gte: 5 } },
          { $set: { name: "5살 이상인 친구들" } },
          (updateErr, updateResult) => {
            if (updateErr) throw updateErr;
            console.log(updateResult);
          }
        );
      }
    );
  });});
```

```
{
  acknowledged: true,
  modifiedCount: 2,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 2
}
```

```
_id: ObjectId('641863c99b2b873f95d1911e')
name: "5살 이상인 친구들"
age: 5
```

```
_id: ObjectId('641863c99b2b873f95d1911f')
name: "5살 이상인 친구들"
age: 6
```

```
_id: ObjectId('641863c99b2b873f95d19120')
name: "crong"
age: 4
```

검색

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        test.findOne({ name: "loopy" }, (findErr, findData) => {
          console.log(findData);
        });
      }
    );
  });
});
```

```
{
  _id: new ObjectId("64186cce2f4c66eaea8e7716"),
  name: 'loopy',
  age: 6
}
```

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        const findCursor = test.find({ name: "loopy" });
        console.log(findCursor);
      }
    );
  });
});
```

```
  raw: false
},
[Symbol(filter)]: { name: 'loopy' },
[Symbol(builtOptions)]: {
  raw: false,
  promoteLongs: true,
  promoteValues: true,
  promoteBuffers: false,
  ignoreUndefined: false,
  bsonRegExp: false,
  serializeFunctions: false,
  fieldsAsRaw: {},
  writeConcern: WriteConcern { w: 'majority' },
  readPreference: ReadPreference {
    mode: 'primary',
    tags: undefined,
```

Find 쿼리는 기존 쿼리와는 달리
콜백을 사용하지 않는
시간이 걸리지 않는 쿼리!

대신 원하는 데이터를 찾아 주는 것이
아니라 해당 데이터가 있는 위치 정보
를 가르키는 cursor 를 리턴

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        const findCursor = test.find({ name: "loopy" });
        console.log(findCursor);
        findCursor.toArray((toArrErr, arrData) => console.log(arrData));

      }
    );
  });
});
```

단 Cursor 에서 데이터를 뽑을 때에는 시간이 필요하여 콜백을 사용!

$set

$set: {}

- MongoDB 의 도큐먼트를 수정할 때 사용합니다.
- 수정 Query 에서 도큐먼트를 수정 할 때 $set: { 수정할 내용 } 으로 수정을 해야 합니다.

```
users.updateOne(
    {
      name: 'loopy',
    },
    {
      $set: {
        name: '루피',
      },
    }
```

조건 → name 프로퍼티가 'loopy'인 도큐먼트 찾기

찾은 도큐먼트의 name 프로퍼티를 해당 값으로 변경!

비교식

쿼리	설명
$eq	일치하는 값을 찾는다.
$gt	지정된 값보다 큰 값을 찾는다.
$gte	크거나 같은 값을 찾는다.
$lt	지정된 값보다 작은 값을 찾는다.
$lte	작거나 같은 값을 찾는다.
$ne	일치하지 않는 모든 값을 찾는다.($eq의 부정)
$in	배열에 지정된 값 중 하나와 일치한 값을 찾는다.
$nin	배열에 지정된 값과 일치하지 않는 값을 찾는다.

논리식

- 여러 조건을 걸 때에는 조건들을 배열에 담긴 객체 형태로 전달 합니다!

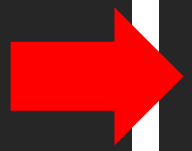
```
db.컬렉션명.find({쿼리: [{조건1}, {조건2}, ...]})
```

쿼리	설명
$or	조건들 중 하나라도 true면 반환 (true: 조건과 일치, false: 조건과 불일치)
$and	조건들이 모두 true일 때 반환
$not	조건이 false일 때 반환
$nor	조건들이 모두 false일 때 반환

콜백 지옥을

Async / Await 로!

```
client.connect((err) => {
  const member = client.db('kdt5').collection('member');
  member.deleteMany({}, (deleteManyErr, deleteManyResult) => {
    if (deleteManyErr) throw deleteManyErr;
    member.insertMany(
      [
        { name: '신상아', age: 24 },
        { name: '김호준', age: 29 },
        { name: '홍성범', age: 32 },
        { name: '구슬기', age: 30 },
      ],
      (insertManyErr, insertManyResult) => {
        if (insertManyErr) throw insertManyErr;
        member.insertOne(
          { name: '이효석', age: 39 },
          (insertOneErr, insertOneReuslt) => {
            if (insertOneErr) throw insertOneErr;
            member.deleteOne(
              { name: '신상아' },
              (deleteOneErr, deleteOneResult) => {
                if (deleteOneErr) throw deleteOneErr;
                member.updateOne(
                  { name: '이효석' },
                  { $set: { name: '신상아', age: 24 } },
                  (updateOneErr, updateOneResult) => {
                    if (updateOneErr) throw updateOneErr;
                    const oldCursor = member.find({ age: { $gte: 25 } });
                    oldCursor.toArray((toArrErr, toArrData) => {
                      if (toArrErr) throw toArrErr;
                      console.log(toArrData);
                    });
                  },
                );
              },
            );
          },
        );
      },);});});});
```



```
async function main() {
  try {
    await client.connect();

    const member = client.db('kdt5').collection('member');

    await member.deleteMany({});
    await member.insertMany([
      { name: '신상아', age: 24 },
      { name: '김호준', age: 29 },
      { name: '홍성범', age: 32 },
      { name: '구슬기', age: 30 },
    ]);
    await member.insertOne({ name: '이효석', age: 39 });
    await member.deleteOne({ name: '신상아' });
    await member.updateOne(
      { name: '이효석' },
      { $set: { name: '신상아', age: 24 } },
    );
    const findCursor = member.find({ age: { $gte: 25 } });
    const dataArr = await findCursor.toArray();
    console.log(dataArr);
  } catch (err) {
    console.error(err);
  }
}
main();
```

MongoDB로 기능 구현

MongoDB 접속 클라이언트를 모듈화!

- Controllers 폴더에 mongoConnect.js 파일을 만들고 몽고 디비 접속 클라이언트를 모듈화 시켜 줍시다!

```
const { MongoClient, ServerApiVersion } = require('mongodb');

const uri =
  'mongodb+srv://xenosign1:qwer1234@cluster0.8sphltr.mongodb.net/?retryWrites=true&w=majority';

const client = new MongoClient(uri, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  serverApi: ServerApiVersion.v1,
});

module.exports = client;
```

MongoDB 용 컨트롤러 생성

기존 중복 회원 체크 기능!

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

- 콜백을 Async / Await 으로 변경
- MySQL을 몽고 디비로 변경!

새로운 중복 회원 체크 기능!

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

const userDB = {
  userCheck: async (userId) => {
    const client = await mongoClient.connect();
    const user = client.db('kdt5').collection('user');
    const findUser = await user.findOne({ id: userId });
    if (!findUser) return false;
    return findUser;
  },
};

module.exports = userDB;
```

기존 회원 가입 기능!

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

- 콜백을 Async / Await 으로 변경
- MySQL을 몽고 디비로 변경!

새로운 회원 가입 기능!

```
registerUser: async (newUser) => {
    const client = await mongoClient.connect();
    const user = client.db('kdt5').collection('user');

    const insertResult = await user.insertOne(newUser);
    if (!insertResult.acknowledged) throw new Error('회원 등록 실패');
    return true;
  },
```

회원 가입 라우터

코드 수정!

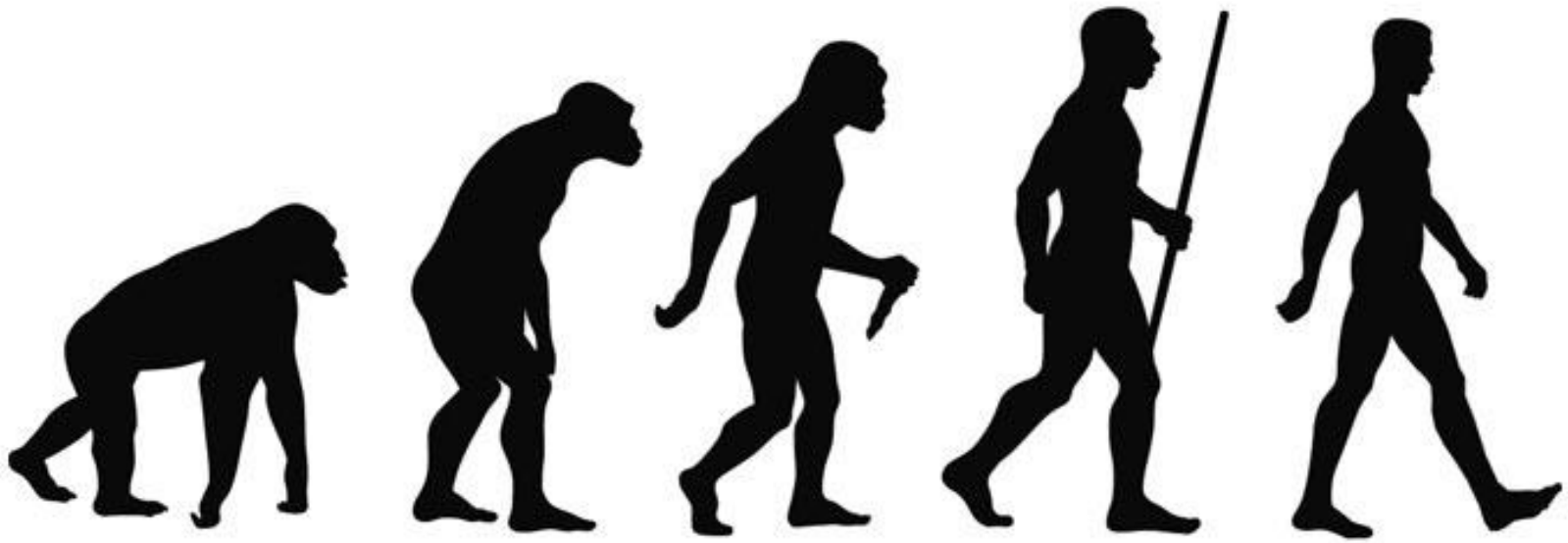
기존 회원 가입 라우터 코드

```
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">로그인으로 이동</a>');
        } else {
          res.status(500);
          res.send(
            '회원 가입 실패! 알 수 없는 문제 발생<br><a href="/register">회원 가입으로 이동</a>',
          );
        }
      });
    } else {
      res.status(400);
      res.send(
        '동일한 ID를 가진 회원이 존재 합니다!<br><a href="/register">회원 가입으로 이동</a>',
      );
    }
  });
});
```

새로운 회원 가입 라우터 코드

```
router.post('/', async (req, res) => {
  const duplicatedUser = await userDB.userCheck(req.body.id);
  if (!duplicatedUser) {
    const registerResult = await userDB.registerUser(req.body);
    if (registerResult) {
      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.send(
      '중복된 id 가 존재합니다.<br><a href="/register">회원가입 페이지로 이동</a>',
    );
  }
});
```


지금 시작합니다



Refactoring

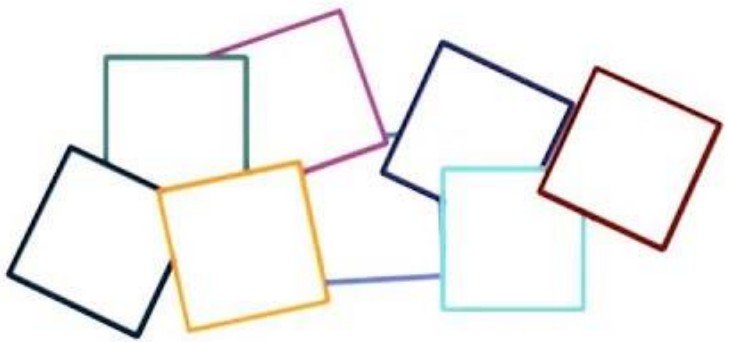
코드

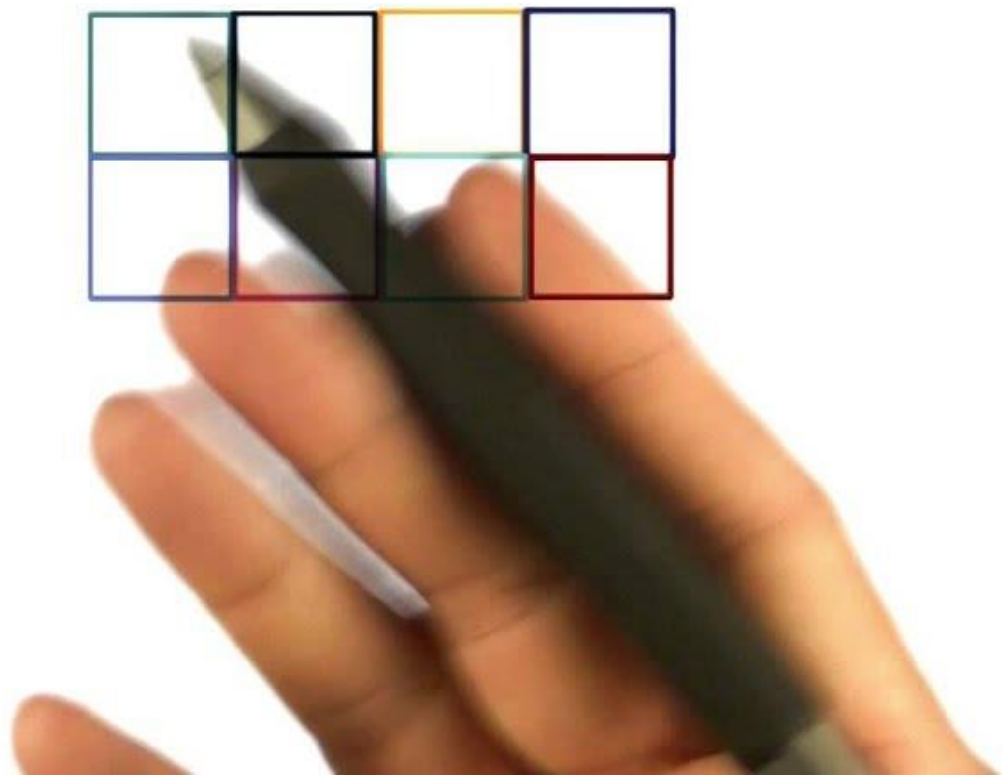
리팩토링!

Code Refactoring (코드 리팩토링)

- 기존 코드를 더 좋게 변경하는 것을 의미합니다~!
- 새로운 기능을 추가할 때
- 예전 기술을 새로운 기술로 변경 할 때
- 기존 코드를 더 가독성이 좋거나, 확장성이 좋은 코드로 변경할 때
- 등등 다양한 상황에서 코드를 개선하는 행위를 의미합니다!

REFACTORING






너흰 아직 준비가 됐다!


참... 좋다!

현재 코드의 문제점?

- 컨트롤러와 라우터가 중복된 작업을 합니다~!
- 하나의 곳에서 처리가 가능할 일을 굳이 둘로 나누어 처리하는 느낌?
- 그럼!?

→ 회원 가입에 대한 처리는 컨트롤러에서 전부 처리하고, 라우터는 주소 연결만 해봅시다!

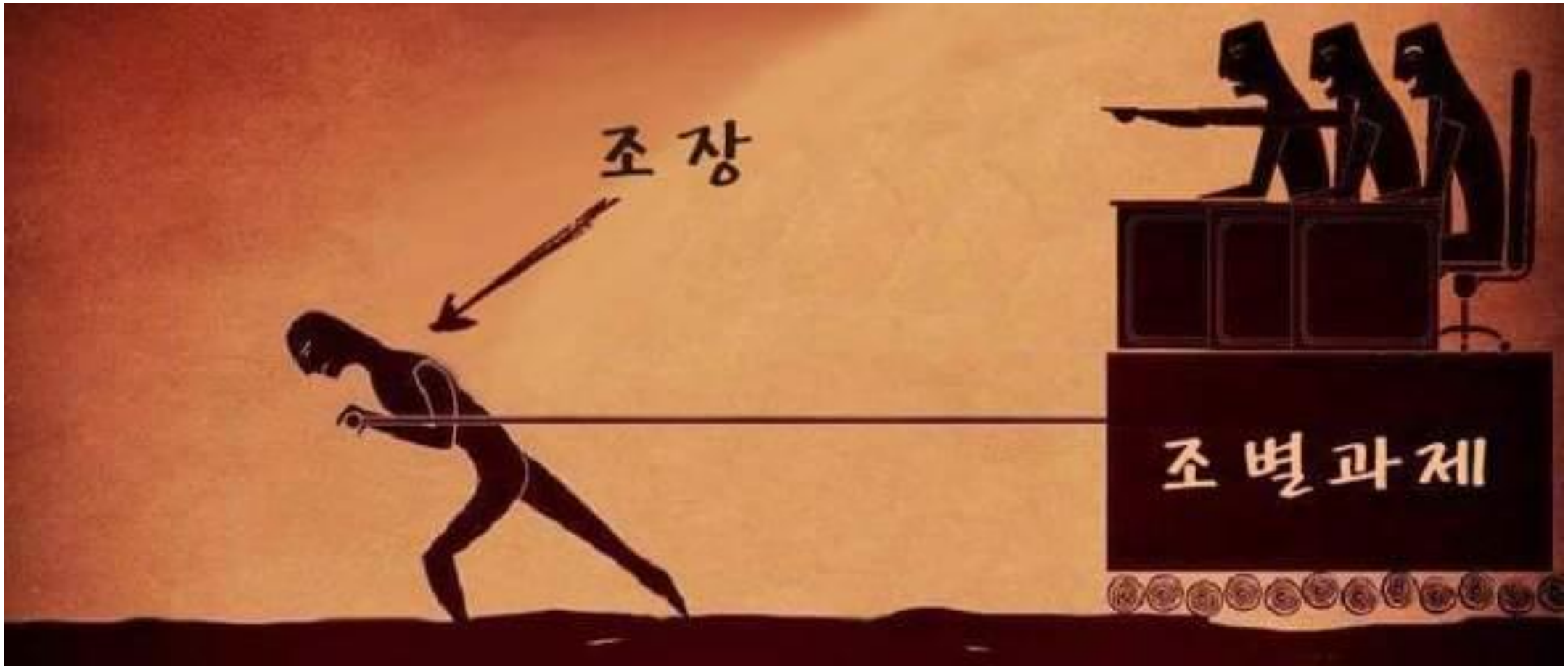
→ 중복 회원 체크 기능을 빼고 회원 가입 기능, 로그인 기능으로 확실하게 처리

ESLint


```
module.exports = {
  extends: ['airbnb-base'],
  rules: {
    'linebreak-style': 0,
    'no-console': 'off',
    'operator-linebreak': 'off',
    'consistent-return': 'off',
    'nonblock-statement-body-position': 'off',
    'import/no-extraneous-dependencies': 'off',
    curly: 'off',
  },
  parserOptions: {
    ecmaVersion: 'latest',
  },
  env: {
    es6: true,
  },
};
```

컨트롤러가

이제 일을 다합니다!


조 장
조 별 과 제

컨트롤러가 다합니다!

- 회원 가입에 대한 처리는 컨트롤러에서 전부 처리가 가능
- 굳이 컨트롤러에서 결과를 라우터로 보내서 처리할 필요가 없습니다
- 이제 모든 처리 및 응답을 컨트롤러가 전담합니다!
- 기존에 하나의 객체에 메소드로 전부 구현했던 방식도 메모리 상으로 더 효율적인 코드를 위해 각각 함수로 분리하여 구현 합니다~!

```
const mongoClient = require('./mongoConnect');
const REGISTER_DUPLICATED_MSG =
  '동일한 ID를 가진 회원이 존재 합니다!<br><a href="/register">회원 가입으로 이동</a>';
const REGISTER_SUCCESS_MSG =
  '회원 가입 성공!<br><a href="/login">로그인으로 이동</a>';
const REGISTER_UNEXPECTED_MSG =
  '회원 가입 실패! 알 수 없는 문제 발생<br><a href="/register">회원 가입으로 이동</a>';

const registerUser = async (req, res) => {
  try {
    const client = await mongoClient.connect();
    const user = client.db('kdt5').collection('user');

    const duplicatedUser = await user.findOne({ id: req.body.id });
    if (duplicatedUser) return res.status(400).send(REGISTER_DUPLICATED_MSG);

    await user.insertOne(req.body);
    res.status(200).send(REGISTER_SUCCESS_MSG);
  } catch (err) {
    console.error(err);
    res.status(500).send(REGISTER_UNEXPECTED_MSG);
  }
};
module.exports = {
  registerUser,
};
```

```
const mongoClient = require('./mongoConnect');
const REGISTER_DUPLICATED_MSG =
  '동일한 ID를 가진 회원이 존재 합니다!<br><a href="/register">회원 가입으로 이동</a>';
const REGISTER_SUCCESS_MSG =
  '회원 가입 성공!<br><a href="/login">로그인으로 이동</a>';
const REGISTER_UNEXPECTED_MSG =
  '회원 가입 실패! 알 수 없는 문제 발생<br><a href="/register">회원 가입으로 이동</a>';

const registerUser = async (req, res) => {
  try {
    const client = await mongoClient.connect();
    const user = client.db('kdt5').collection('user');

    const duplicatedUser = await user.findOne({ id: req.body.id });
    if (duplicatedUser) return res.status(400).send(REGISTER_DUPLICATED_MSG);

    await user.insertOne(req.body);
    res.status(200).send(REGISTER_SUCCESS_MSG);
  } catch (err) {
    console.error(err);
    res.status(500).send(REGISTER_UNEXPECTED_MSG);
  }
};
module.exports = {
  registerUser,
};
```

라우터에서 콜백으로 받아온
req, res 사용

전체 객체를 빼는게 아니라
필요한 함수만 빼기

라우터는?

라우팅만~!

기존 라우터는!?

- 컨트롤러에서 결과를 보고 바로 해결이 가능하던 일을 굳이 받아와서 정신 없이 처리 합니다!
- 이제는 라우터는 라우팅 기능만 하면 됩니다 → 특정 주소를 받아오면 컨트롤러를 연결!

기존 라우터 코드

```
router.post('/', async (req, res) => {
  const duplicatedUser = await userDB.userCheck(req.body.id);
  if (!duplicatedUser) {
    const registerResult = await userDB.registerUser(req.body);
    if (registerResult) {
      res.status(200);
      res.send('회원 가입 성공!<br><a href="/login">로그인으로 이동</a>');
    } else {
      res.status(500);
      res.send(
        '회원 가입 실패! 알 수 없는 문제 발생<br><a href="/register">회원 가입으로 이동</a>',
      );
    }
  } else {
    res.status(400);
    res.send(
      '동일한 ID를 가진 회원이 존재 합니다!<br><a href="/register">회원 가입으로 이동</a>',
    );
  }
});
```

수정 라우터 코드

```
const { registerUser } = require('../controllers/userController');

router.post('/', registerUser);
```

회원가입

아이디

13

비밀번호

••

회원가입

회원 가입 성공!

로그인으로 이동

로그인 라우터

코드 수정!

인생은 실전이야

실습, 로그인 코드 변경하기!

- MySQL → MongoDB
- 콜백 → Async / Await
- 코드 리팩토링(컨트롤러가 모든 일을 하고, 라우터는 라우팅 기능만!)
- 위의 조건을 만족하도록 로그인 컨트롤러와 라우터 코드를 작성해 봅시다!
- 기존 로그인 라우터에서 했던 작업들(코드)를 자연스럽게 컨트롤러로 옮기시면 됩니다~!
- 단, 로그인이 성공하면 res.send('로그인 성공'); 으로 처리해주세요!

로그인

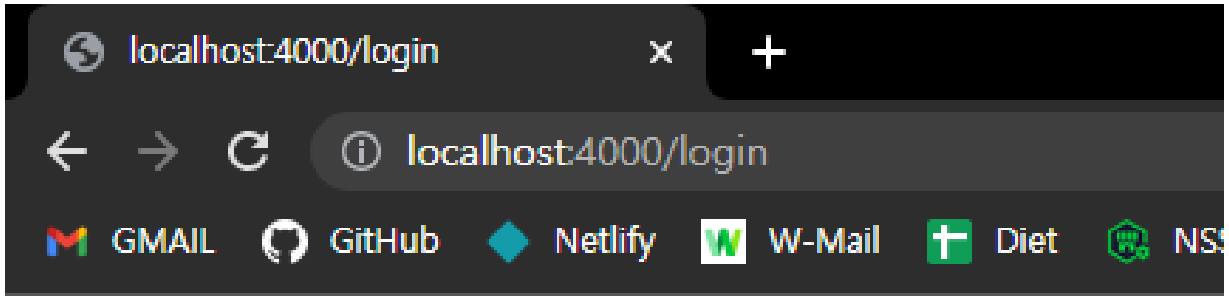
아이디

11

비밀번호

••

로그인

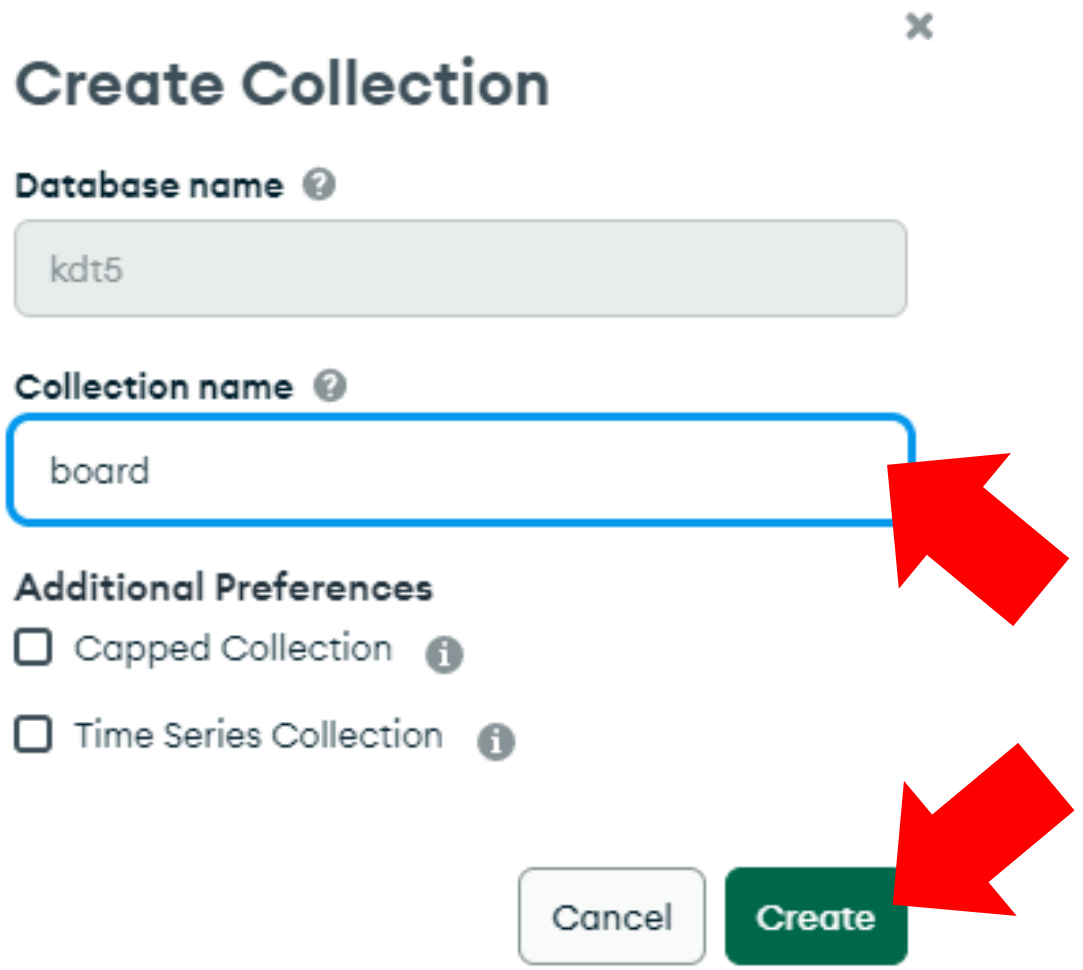


로그인 성공!

게시판도

MongoDB로!

게시판용 콜렉션과 도큐먼트 생성하기!



+ Create Database

Search Namespaces

- kdt-test
- kdt1
- kdt4
- kdt5
  - board
  - test
  - users
- login

kdt5.board

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 0B TOTAL DOCUMENTS: 0 INDEXES TOTAL SIZE: 4KB

Find Indexes Schema Anti-Patterns 0 Aggregation Search Indexes

INSERT DOCUMENT

FILTER { field: 'value' }

Apply Reset

QUERY RESULTS: 0

Insert to Collection board

VIEW { } ☰

```
1   _id: 638a4b32545fa5ecb5be50cb                ObjectId
2   USERID: "11  //"                             String
3   TITLE: "제목1//"                             String
4   CONTENT: "내용1//"                           String
```

Cancel Insert

```
_id: ObjectId('638a4b32545fa5ecb5be50cb')
USERID: "11"
TITLE: "제목1"
CONTENT: "내용1"
```

게시판용

컨트롤러 만들기!

MongoDB 용 컨트롤러 코드 작성!

- 몽고 디비 접속용 클라이언트 모듈 불러오기

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

전체 게시글

가져오기 컨트롤러

새로운 전체 게시글 가져오기 컨트롤러 코드

- 이제 해당 컨트롤러가 페이지 까지 그려주는 작업을 완료 합니다!
- 라우터는 연결만 하는 것이죠!

새로운 전체 게시글 가져오기 컨트롤러 코드

```
const UNEXPECTED_MSG = '<br><a href="/">메인 페이지로 이동</a>';

const getAllAtricles = async (req, res) => {
  try {
    const client = await mongoClient.connect();
    const board = client.db('kdt5').collection('board');

    const allArticleCusor = board.find({});
    const ARTICLE = await allArticleCusor.toArray();
    res.render('db_board', {
      ARTICLE,
      articleCounts: ARTICLE.length,
      userId: req.session.userId,
    });
  } catch (err) {
    console.error(err);
    res.status(500).send(err.message, UNEXPECTED_MSG);
  }
};
```

기존 전체 게시글 보여주기 라우터 코드 수정

- 전체 게시글을 MongoDB에서 받아오는 컨트롤러를 만들었으니, 해당 기능을 이용 기존 라우터를 변경해 봅시다!
- 라우터는 이제 컨트롤러 기능만 연결하면 끝입니다!

기존 게시판 페이지 라우터

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

새로운 게시판 페이지 라우터

```
const { getAllAtricles } = require('../controllers/boardController’);

// 게시판 페이지 호출
router.get('/', isLogin, getAllAtricles);
```

Tetz Board

현재 등록 글 : 1 글쓰기 로그아웃

작성자 : 11

제목1

내용1

수정 삭제

글 쓰기

페이지로 이동

글쓰기로 이동하는 기능은?

- 글쓰기로 이동 할때는 컨트롤러 기능(DB에서 데이터를 받는 작업)을 전혀 사용하지 않으므로 기존 라우터 코드를 그대로 사용합니다!

```
// 게시판 페이지 호출
router.get('/', isLogin, getAllAtricles);

// 글쓰기 페이지 호출
router.get('/write', isLogin, (req, res) => {
  res.render('db_board_write');
});
```

글 쓰기

기능 구현

기존 글 쓰기 컨트롤러 코드

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

새로운 글 쓰기 컨트롤러 코드

```
const writeArticle = async (req, res) => {
  try {
    const client = await mongoClient.connect();
    const board = client.db('kdt5').collection('board’);

    const newArticle = {
      USERID: req.session.userId,
      TITLE: req.body.title,
      CONTENT: req.body.content,
    };
    await board.insertOne(newArticle);
    res.redirect('/dbBoard');
  } catch (err) {
    console.error(err);
    res.status(500).send(err.message + UNEXPECTED_MSG);
  }
};

module.exports = { getAllAtricles, writeArticle };
```

기존 글 쓰기 라우터 코드

```
router.post('/write', isLogin, (req, res) => {
  if (req.body.title && req.body.content) {
    const newArticle = {
      userId: req.session.userId,
      title: req.body.title,
      content: req.body.content,
    };
    boardDB.writeArticle(newArticle, (data) => {
      if (data.affectedRows >= 1) {
        res.redirect('/dbBoard');
      } else {
        const err = new Error('글 쓰기 실패');
        err.statusCode = 500;
        throw err;
      }
    });
  } else {
    const err = new Error('글 제목 또는 내용이 없습니다!');
    err.statusCode = 400;
    throw err;
  }
});
```

새로운 글 쓰기 라우터 코드

```
const {
  getAllAtricles,
  writeArticle,
} = require('../controllers/boardController’);

// 데이터 베이스에 글쓰기
router.post('/write', isLogin, writeArticle);
```

글 쓰기

제목

테스트

내용

테스트

글 작성하기

Tetz Board

현재 등록 글 : 3 글쓰기 로그아웃

작성자 : 11

제목1

내용1

수정 삭제

작성자 : 11

테스트

테스트

수정 삭제

글 수정하기

코드 수정!

엇? 그런데!?

- 기존 MySQL 에서는 ID_PK 라는 컬럼을 이용해서 게시글을 특정 할 수 있었습니다!
- 그런데 MongoDB에는 ID_PK 값이 없네요!?!?
- 이럴 땐, objectID 인 _id 값을 사용하면 됩니다!

```
_id: ObjectId('638a4b32545fa5ecb5be50cb')
USERID: "11"
TITLE: "제목1"
CONTENT: "내용1"
```

수정을 위해 ejs 파일 코드 수정

- 기존의 ID_PK 값을 전달 하던 것을, _id 값을 전달 하도록 수정해 줍시다!

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

기존 게시글 찾기 컨트롤러 코드

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

새로운 게시글 찾기 컨트롤러 코드

- ObjectID 를 사용하려면 mongdb 모듈의 ObjectID 클래스를 가져와야만 합니다!₩

```
_id: ObjectId('638a4b32545fa5ecb5be50cb')
```

- _id 는 단순한 문자열로 보이지만 해당 문자열은 특정 의미를 가지고 있으며 해당 의미는 ObjectID 클래스로만 해독이 가능하기 때문이죠!

```
const { ObjectId } = require('mongodb');
const mongoClient = require('./mongoConnect');
```

MongoDB의 ObjectID

ObjectID는 12byte 크기의 문자와 숫자로 구성된 값입니다. ObjectID()의 값을 반환하면 12byte의 hexadecimal 값으로 결과를 반환합니다. 그리고 이 값들은 각각의 의미를 가지고 있습니다.

- 첫 4byte는 timestamp 값을 의미합니다. 이 값은 Unix시대부터 초단위로 측정된 값을 의미합니다.
- 다음 5byte는 랜덤으로 생성된 값입니다.
- 다음 3byte는 증가하는 count이며, 최초값은 랜덤으로 생성됩니다.

새로운 게시글 수정 모드 이동 컨트롤러 코드

```
const getArticle = async (req, res) => {
  try {
    const client = await mongoClient.connect();
    const board = client.db('kdt5').collection('board');

    const selectedArticle = await board.findOne({
      _id: ObjectId(req.params.id),
    });
    res.render('db_board_modify', selectedArticle);
  } catch (err) {
    console.error(err);
    res.status(500).send(err.message + UNEXPECTED_MSG);
  }
};

module.exports = { getAllAtricles, writeArticle, getArticle };
```

기존 수정 모드로 이동 라우터 코드

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

새로운 수정 모드로 이동 라우터 코드

```
const {
  getAllAtricles,
  writeArticle,
  getArticle,
} = require('../controllers/boardController');

// 글 수정 모드로 이동
router.get('/modify/:id', isLogin, getArticle);
```

작성자 : 11

테스트

테스트

수정 삭제

Write Mode

제목

테스트

내용

테스트

글 수정하기

실습, 게시글 수정 기능 완성하기!

- 게시글 수정 모드로 이동 까지는 완성 했습니다!
- 그럼 이제 글 수정하기 버튼을 클릭하면 해당 글이 수정 되도록 코드를 수정 해주시면 됩니다!
- 먼저 글을 수정하는 ejs 파일에 가서 ID_PK 가 아닌 _id 값을 전달 하도록 수정
- 글을 수정해서 DB에 Update 하는 modifyArticle 컨트롤러를 리팩토링 측면에서 수정해 주세요~!

글 삭제하기

코드 수정!

ejs 파일 코드 수정

- 수정 때와 마찬 가지로 기존의 ID_PK 값을 전달 하던 것을, _id 값을 전달 하도록 수정해 줍시다!

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

기존 삭제 컨트롤러 코드

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

새로운 삭제 컨트롤러 코드

```
const deleteArticle = async (req, res) => {
  try {
    const client = await mongoClient.connect();
    const board = client.db('kdt5').collection('board');

    await board.deleteOne({ _id: ObjectId(req.params.id) });
    res.status(200).json('삭제 성공');
  } catch (err) {
    console.error(err);
    res.status(500).send(err.message + UNEXPECTED_MSG);
  }
};

module.exports = { getAllAtricles, writeArticle, getArticle, modifyArticle, deleteArticle };
```

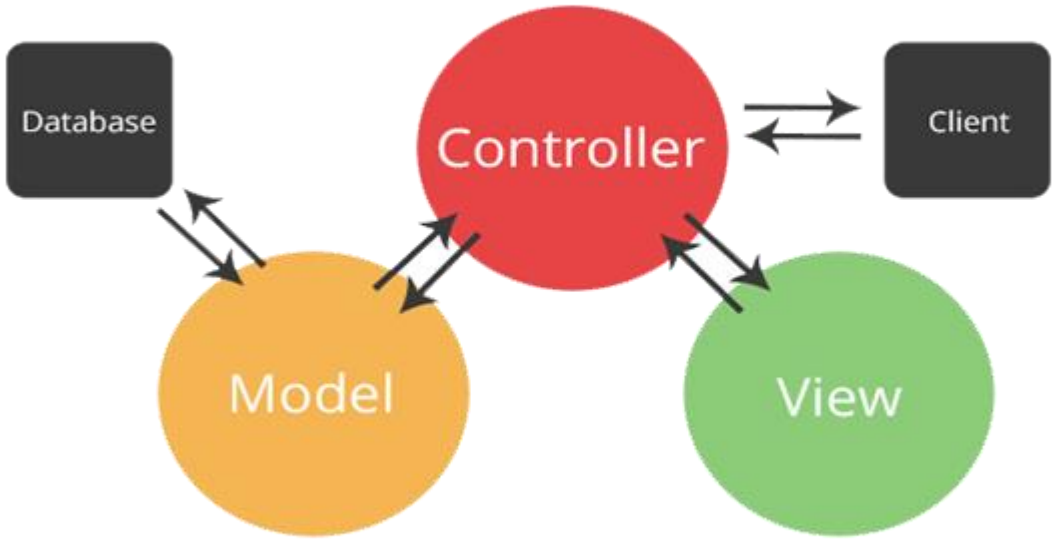
기존 삭제 라우터 코드

```
router.delete('/delete/:id', isLogin, (req, res) => {
  db.deleteArticle(req.params.id, (data) => {
    console.log(data);
    if (data.protocol41) {
      res.send('삭제 완료!');
    } else {
      const err = new Error('글 삭제 실패');
      throw err;
    }
  });
});
```

새로운 삭제 라우터 코드

```
const {
  getAllAtricles,
  writeArticle,
  getArticle,
  modifyArticle,
  deleteArticle,
} = require('../controllers/boardController');

// 글 삭제
router.delete('/delete/:id', isLogin, deleteArticle);
```

MVC 패턴


Database
Controller
Client
Model
View

Mongoose { }

Mongoose?

- MongoDB는 사용상의 제약이 전혀 없어서 편리 합니다!
- 하지만 너무 편리해서 문제가 생기게 되죠.
- 예를 들어서 어제 만든 user 라는 컬렉션에 임의로 데이터를 추가해 봅시다!

Mongoose?

- 예를 들어서 어제 만든 user 라는 컬렉션에 임의로 데이터를 추가해 봅시다!

Insert to Collection user

VIEW { } ≡

```
1   _id: 638b706808787c75c256972b        ObjectId
2   userid: 11                           Int32
+   pw: "11"                             String
```

Cancel Insert

```
_id: ObjectId('638aad0ee37936135e72219d')
id: "11"
password: "11"
```

```
_id: ObjectId('638b706808787c75c256972b')
userid: 11
pw: "11"
```

Mongoose?

- 자 이렇게 되면 어떤 문제가 발생하게 될까요?
- 회원 정보를 가져 올 때 key의 값이 전혀 다르게 되겠죠? 그럼 버그가 발생합니다!
- MySQL 이었다면? 데이터 삽입을 하는 순간에 이미 제약 조건에 의해서 데이터가 삽입이 안되었을 겁니다 → 데이터의 일관성이 유지 되는 것이죠!
- 그럼 MongoDB 에서는 이런 문제를 어떻게 해결할 수 있을까요?

Mongoose { }

Mongoose!

- 바로 몽구스가 해결을 해줍니다!
- MongoDB의 장점은 살리면서 단점을 커버할 수 있게 해주는 것이 바로 Mongoose 모듈입니다!

몽구스

설치

Mongoose 설치

- Npm i mongoose -S

```
lhs@DESKTOP-86MUCGC MINGW64 /d/git/4th_backend (main)
$ npm i mongoose

added 8 packages, and audited 362 packages in 3s

70 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Mongoose 접속 용 모듈 생성

- Controllers 폴더에 mongooseConnect.js 파일 생성
- 이번 기회에 mongoDB 접속 URI 를 dotenv 에 등록 하시죠!

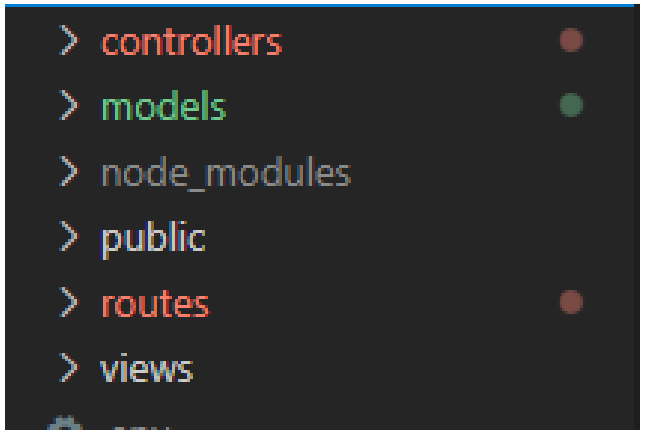
```
PORT = 4000

MYSQL_USER = root
MYSQL_PASSWORD = dlrladk
MYSQL_DB = mydb1

MDB_URI =
mongodb+srv://xenosign1:qwer1234@cluster0.8sphltr.mongodb.net/?retryWrites=true&w=majority
```

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

const { MDB_URI } = process.env;
const connect = async () => {
  try {
    await mongoose.connect(MDB_URI, {
      dbName: 'kdt5',
      useNewUrlParser: true,
    });

    console.log('몽구스 접속 성공!');

    mongoose.connection.on('error', (err) => {
      console.error('몽고 디비 연결 에러', err);
    });
    mongoose.connection.on('disconnected', () => {
      console.error('몽고 디비 연결이 끊어졌습니다. 연결을 재시도 합니다.');
      connect();
    });
  } catch (err) {
    console.error(err);
  }
};

module.exports = connect;
```

User

스키마 생성

스키마 설정을 위한 Models 폴더 만들기



- user 스키마를 작성을 위한 user.js 파일 생성

회원 스키마 정의하기

- 몽구스 모듈 임포트 & 몽구스 모듈의 Schema 클래스 임포트

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

const { Schema } = mongoose;
```

회원 스키마 정의하기

- userSchema 정의
- _id 는 알아서 생성 되므로 정의할 필요가 없습니다!
- id : 문자열 / 필수 / 유니크
- password : 문자열 / 필수
- createAt : 시간 / 생성 시간이 기본으로 삽입
- 컬렉션 이름은 mongoose-user 로 생성!
  - 해당 옵션을 붙이지 않으면 xxxSchema 의 xxx에 s 가 붙는 형태로 생성 됩니다!

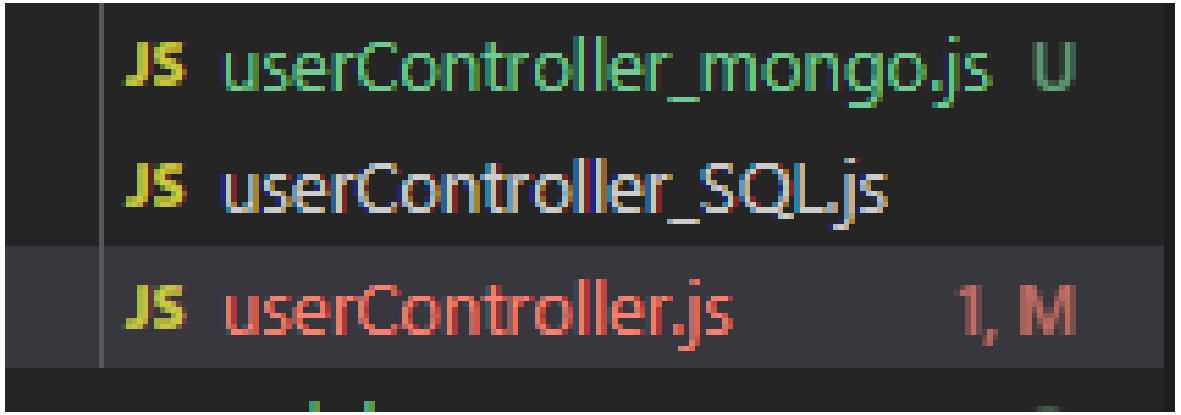
```
const mongoose = require('mongoose');
const { Schema } = mongoose;
const userSchema = new Schema(
  {
    id: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    createdAt: {
      type: Date,
      default: Date.now,
    },
  },
  {
    collection: 'mongoose-user',
  },
);
module.exports = mongoose.model('User', userSchema);
```

컨트롤러

코드 수정

먼저 기본 mongoDB 버전의 컨트롤러 백업!

- 컨트롤러 코드가 수정 될 예정이므로 백업해 둡시다!



몽고 디비 접속 모듈 및 user 스키마 임포트

```
const connect = require('./mongooseConnect');
const User = require('../models/user');

connect();
```

- 클라이언트는 한 번만 접속해도 사용이 가능하니 바로 접속을 시킵시다!

```
서버는 4000번에서 실행 중입니다!
몽고 디비 연결 성공
```

회원 가입

컨트롤러 작성

컨트롤러 변경을 별게 없습니다~!

- 기존에는 컬렉션을 선택해서 컬렉션에 쿼리를 날리던 것을 이제는 모델을 불러서 날리면 됩니다~! 그리고 명령어가 조금 다릅니다!

```
const client = await mongoClient.connect();
const user = client.db('kdt5').collection('user’);
const duplicatedUser = await user.findOne({ id: req.body.id });
```

```
const mongooseConnect = require('./mongooseConnect');
const User = require('../models/users');
mongooseConnect();
const duplicatedUser = await User.findOne({ id: req.body.id });
```

몽구스의 CRUD 쿼리

CRUD	함수명
CREATE	create
READ	find, findById, findOne
UPDATE	updateOne, updateMany, findByIdAndUpdate, findOneAndUpdate
DELETE	deleteOne, deleteMany, findByIdAndDelete, findOneAndDelete

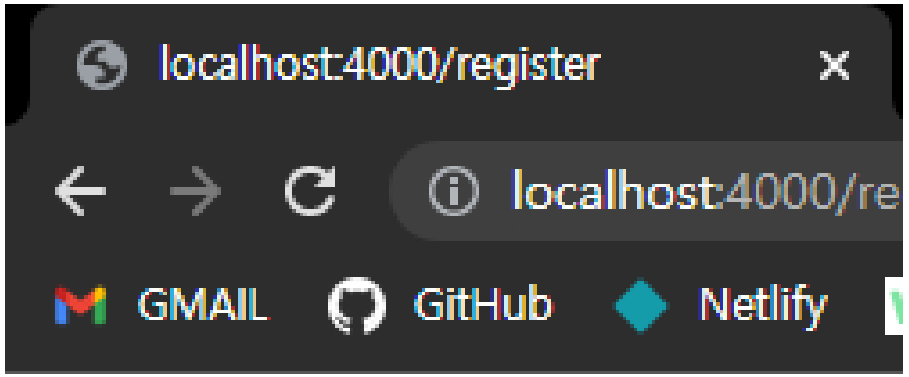
- 생성할 때 insertOne, insertMany 가 아닌 create 문 하나만 씁니다!

기존 컨트롤러 코드

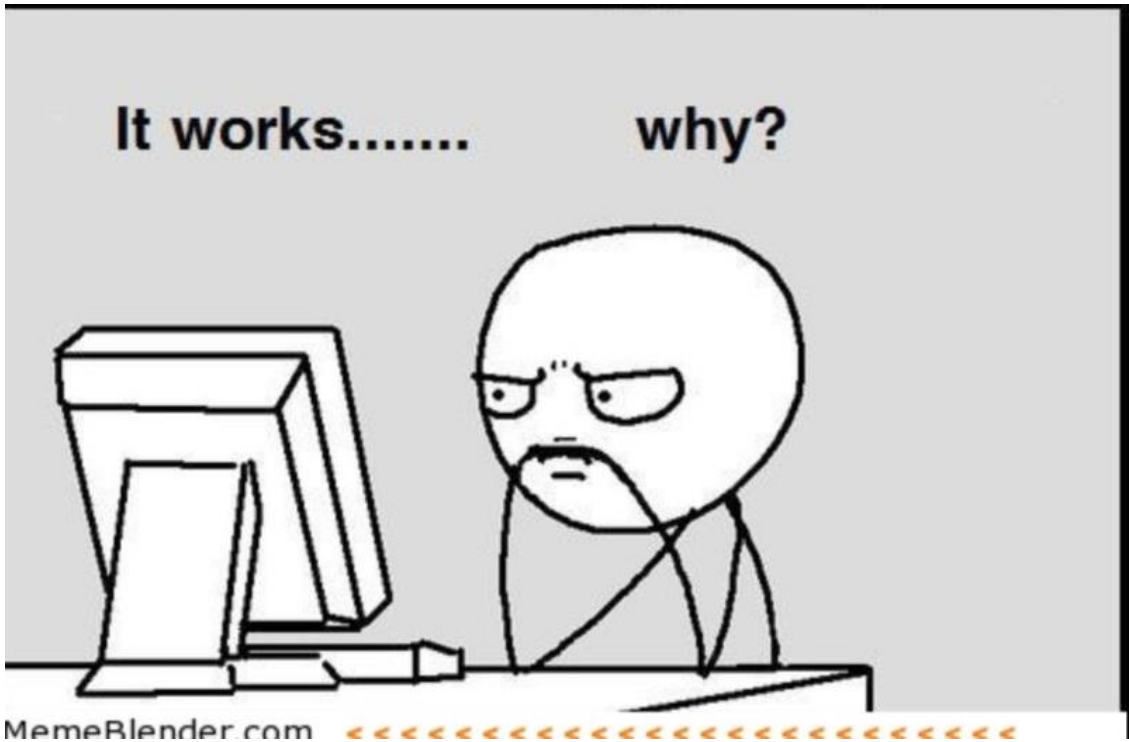
```
const registerUser = async (req, res) => {
  try {
    const client = await mongoClient.connect();
    const user = client.db('kdt5').collection('user');

    const duplicatedUser = await user.findOne({ id: req.body.id });
    if (duplicatedUser) return res.status(400).send(REGISTER_DUPLICATED_MSG);

    await user.insertOne(req.body);
    res.status(200).send(REGISTER_SUCCESS_MSG);
  } catch (err) {
    console.error(err);
    res.status(500).send(REGISTER_UNEXPECTED_MSG);
  }
};
```

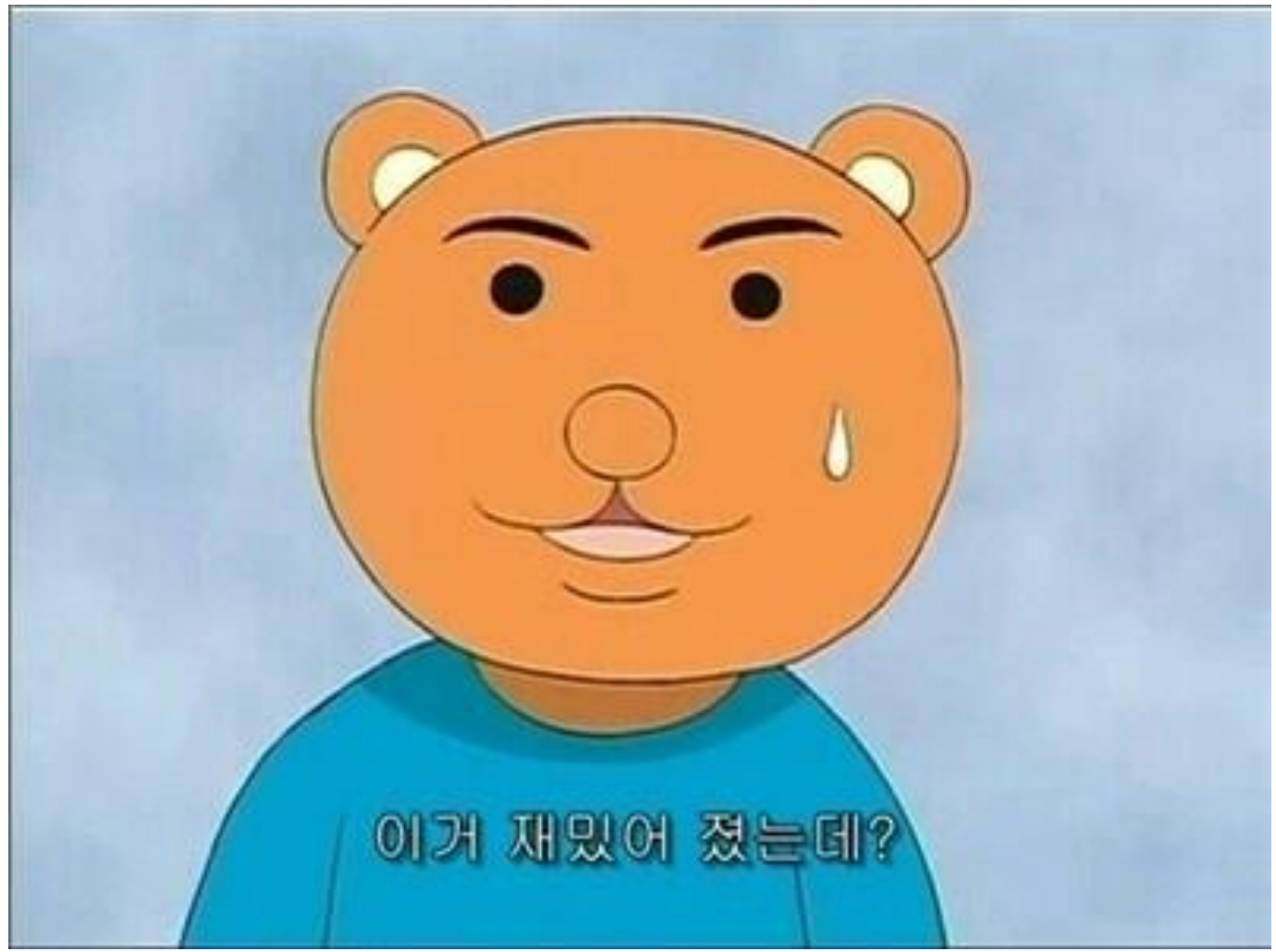
새로운 몽구스 컨트롤러 코드

```
const registerUser = async (req, res) => {
  try {
    const duplicatedUser = await User.findOne({ id: req.body.id });
    if (duplicatedUser) return res.status(400).send(REGISTER_DUPLICATED_MSG);

    await User.create(req.body);
    res.status(200).send(REGISTER_SUCCESS_MSG);
  } catch (err) {
    console.error(err);
    res.status(500).send(REGISTER_UNEXPECTED_MSG);
  }
};
```



회원 가입 성공!
로그인으로 이동




이거 재밌어 졌는데?

어? 생각보다 잘되네요??????

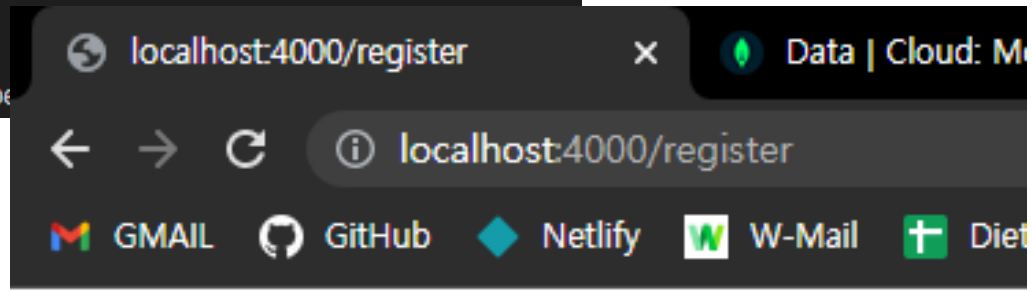
- 그런데 사실 이건 Schema 의 역할을 하나도 테스트 못한 상태입니다
- 기존 몽고 디비처럼 그냥 데이터 받아서 넣고 있으니까요! 자 그럼 이제 Schema 에 정의한 것들이 정상 작동하는지 하나하나 보겠습니다!
- 자, 그럼 이제 회원 가입을 할 때 임의의 값을 전달해서 실제로 Schema 가 역할을 하는지 확인해 보겠습니다!

필수 값을 다르게 전달 해보기

```
const registerUser = async (req, res) => {
  try {
    const duplicatedUser = await User.findOne({ id: req.body.id });
    if (duplicatedUser) return res.status(400).send(REGISTER_DUPLICATED_MSG);

    await User.create({ idd: req.body.id, password: '' });
    res.status(200).send(REGISTER_SUCCESS_MSG);
  } catch (err) {
    console.error(err);
    res.status(500).send(REGISTER_UNEXPECTED_MSG);
  }
};
```

- 기존 몽고 디비였으면 뭐 그냥 idd 라는 키로 데이터를 입력 했겠죠?

```
서버는 4000번 포트에서 실행 중입니다!
몽구스 접속 성공!
Error: User validation failed: id: Path `id` is required.
    at ValidationError.inspect (D:\git\express-board\node_modules\mongoose\lib\error\validation.js:50:26)
    at formatValue (node:internal/util/inspect:782:19)
    at inspect (node:internal/util/inspect:347:10)
    at formatWithOptionsInternal (node:internal/util/inspect:2167:40)
    at formatWithOptions (node:internal/util/inspect:2029:10)
    at console.value (node:internal/console/constructor:332:14)
    at console.warn (node:internal/console/constructor:365:61)
    at registerUser (D:\git\express-board\controllers\userController.js:28:13)
    at processTicksAndRejections (node:internal/process/task_queues:96:5) {
  errors: {
    id: ValidatorError: Path `id` is required.
        at validate (D:\git\express-board\node_modules\mongoose\lib\schematype
```



회원 가입 실패! 알 수 없는 문제 발생
회원 가입으로 이동

오! 역할을 하는 군요!!

- Schema 의 정의를 따르지 않았기 때문에 에러가 발생함을 볼 수 있습니다!
- 즉, 이제 데이터의 완결성은 Schema가 처리를 해주는 것이죠!

그렇다면!?

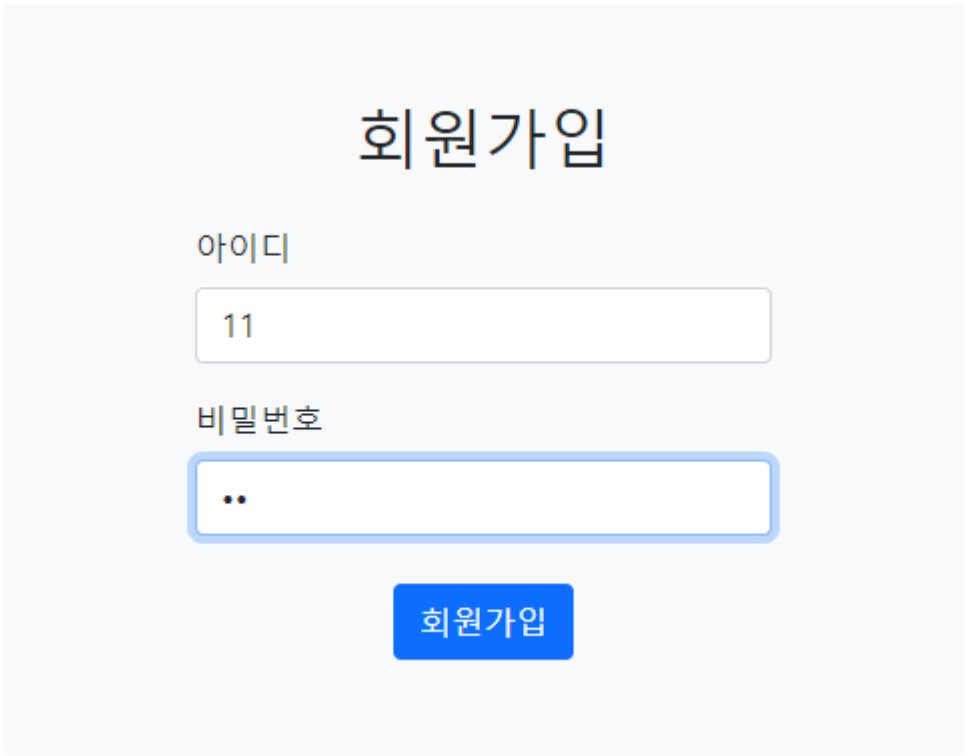
```
const { Schema } = mongoose;
const userSchema = new Schema(
  {
    id: {
      type: String,
      required: true,
      unique: true,
    },
```

- 이제 Schema 가 회원 중복 여부를 체크해 줍니다! → 회원 중복 여부를 체크하는 코드를 제거하고 회원 가입을 시도해 봅시다!

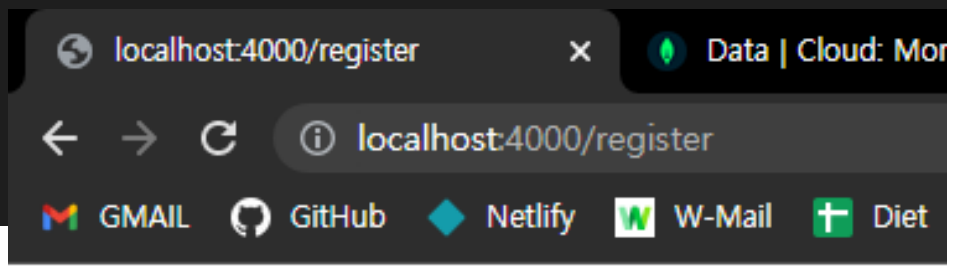
컨트롤러 코드에서 중복 체크 제거

```
const registerUser = async (req, res) => {
  try {
    // const duplicatedUser = await User.findOne({ id: req.body.id });
    // if (duplicatedUser) return res.status(400).send(REGISTER_DUPLICATED_MSG);

    await User.create(req.body);
    res.status(200).send(REGISTER_SUCCESS_MSG);
  } catch (err) {
    console.error(err);
    res.status(500).send(REGISTER_UNEXPECTED_MSG);
  }
};
```

현재 가입 된 회원 정보 확인!



```
MongoServerError: E11000 duplicate key error collection: kdt4.mongoose-user index: id_1 dup key: { id: "11" }
    at D:\git\4th_backend\node_modules\mongoose\node_modules\mongodb\lib\operations\insert.js:53:33
    at D:\git\4th_backend\node_modules\mongoose\node_modules\mongodb\lib\cmap\connection_pool.js:308:25
    at D:\git\4th_backend\node_modules\mongoose\node_modules\mongodb\lib\sdam\server.js:213:17
    at handleOperationResult (D:\git\4th_backend\node_modules\mongoose\node_modules\mongodb\lib\sdam\server.js:329:20)
    at Connection.onMessage (D:\git\4th_backend\node_modules\mongoose\node_modules\mongodb\lib\cmap\connection.js:219:9)
    at MessageStream.<anonymous> (D:\git\4th_backend\node_modules\mongoose\node_modules\mongodb\lib\cmap\connection.js:60:60)
    at MessageStream.emit (node:events:526:28)
    at processIncomingData (D:\git\4th_backend\node_modules\mongoose\node_modules\mongodb\lib\cmap\message_stream.js:132:20)
    at MessageStream._write (D:\git\4th_backend\node_modules\mongoose\node_modules\mongodb\lib\cmap\message_stream.js:33:9)
    at writeOrBuffer (node:internal/streams/writable:389:12) {
  index: 0,
  code: 11000,
  keyPattern: { id: 1 },
  keyValue: { id: '11' },
  [Symbol(errorLabels)]: Set(0) {}
}
```



회원 가입 실패! 알 수 없는 문제 발생

회원 가입으로 이동

실습, 로그인 기능도 Model 코드로 변경!

- User 모델이 생성이 되었으므로, 기존 로그인 컨트롤러 코드를 모델을 적용해서 변경해 봅시다!


수고하셨습니다!