22. MongoDB 연동

22.3 mongoose 설치 및 적용

- 21.koa프로젝트 복사 & npm install
- 설치 : `yarn add mongoose@6.10.5 dotenv`
  - mongodb 6.0.4버전환경에서 mongoose@7.0.4버전설치시 에러발생
    - npm view mongoose
      - 5x: 5.13.17 latest: 7.0.4 legacy: 6.10.5 next: 7.0.0-rc0 unstable: 3.9.7
    - mongoose 6.10.5버전설치
  - dotenv는 환경변수를 저장하고 사용할 수 있게 하는 개발도구
  - dotenv에 mongoose 접속정보를 저장
  - .gitignore에 민감한 정보는 제외하도록 설정
- .env작성

.env

```
txt
PORT=4001
MONGO_URI=mongodb://127.0.0.1:27017/blog
```

src/index.js

```
require('dotenv').config();
const Koa = require("koa");
const Router = require('koa-router');
const bodyParser = require('koa-bodyparser');
const mongoose = require('mongoose');


const api = require('./api');

// 비구조화 할당을 통해 process.env내부값에 대한 레퍼런스 만들기
const { PORT, MONGO_URI } = process.env;

mongoose
  .connect(MONGO_URI)
  .then(() => {
    console.log("Conntected to MongoDB");
  })
  .catch(e => {
    console.error(e);
})

const app = new Koa();
const router = new Router();

// 라우터설정
router.use('/api', api.routes()); // api 라우트 적용
```

```
// 라우터적용전에 bodyParser적용
app.use(bodyParser());

// app 인스턴스에 라우터 적용
app.use(router.routes()).use(router.allowedMethods());

// PORT가 지정되어 있지 않다면 4000사용
const port = PORT || 4000;

app.listen(port, () => {
  console.log('Listening to port %d', port)
});
```

22.4 esm을 ES모듈 import/export문법 사용하기

- 기존 react 프로젝트에서 사용하던 ES모듈 import/export문법은 아직 Node.js에서 정식으로 지원하지 않는다.
- Node.js에 해당기능이 구현되어있긴 하지만 실험단계이기 때문에 기본옵션으로 사용할 수 없다.
- 확장자를 .mjs로 하고 node를 실행할 때 --experimental-modules옵션을 넣어 주어야 한다.
- Node.js에서 import/export문법을 사용해야할 필요는 없지만 이 문법을 사용하면 자동로딩, 코드가 간결해 진다.
- 설치 : `yarn add esm`

src/index.js

- src/index.js를 main.js로 변경후 src/index.js 작성

```
// 이 파일에서만 no-global-assign ESLint 옵션을 비활성화
/* eslint-disable no-global-assign */

require = require('esm')(module /*, options */);
module.exports = require('./main.js');
```

package.json

```
{
  {
    // ( ... )
  },
  "scripts": {
    "start": "node -r esm src",
    "start:dev": "nodemon --watch src/ -r esm src/index.js"
  }
}
```

.eslintrc.json

```
"parserOptions": {
      "ecmaVersion": 2018,
      "sourceType": "module"
    },
```

- 이제 프로젝트에서 import/export를 자유롭게 사용할 수 있다.

22.4.1 기존 코드를 ES Module형태로 바꾸기

api/posts/posts.ctl.js - exports코드를 export const로 변경

- 일괄변경 : 'exports.'을 마우스로 drag -> ctrl+shift+l -> export const ```js let postId = 1; // id의 초깃값입니다.

// posts 배열 초기 데이터 const posts = [{ id: 1, title: '제목', body: '내용', },];

/ *포스트 작성 POST /api/posts { title, body }*/ export const write = ctx => { // REST API의 request body는 ctx.request.body에서 조회할 수 있습니다. const { title, body } = ctx.request.body; postId += 1; // 기존 postId 값에 1을 더합니다. const post = { id: postId, title, body }; posts.push(post); ctx.body = post; };

/ *포스트 목록 조회 GET /api/posts*/ export const list = ctx => { ctx.body = posts; };

/ *특정 포스트 조회 GET /api/posts/:id*/ export const read = ctx => { const { id } = ctx.params; // 주어진 id 값으로 포스트를 찾습니다. // 파라미터로 받아 온 값은 문자열 형식이니 파라미터를 숫자로 변환하거나, // 비교할 p.id 값을 문자열로 변경해야 합니다. const post = posts.find(p => p.id.toString() === id); // 포스트가 없으면 오류를 반환합니다. if (!post) { ctx.status = 404; ctx.body = { message: '포스트가 존재하지 않습니다.', }; return; } ctx.body = post; };

/ *특정 포스트 제거 DELETE /api/posts/:id*/ export const remove = ctx => { const { id } = ctx.params; // 해당 id를 가진 post가 몇 번째인지 확인합니다. const index = posts.findIndex(p => p.id.toString() === id); // 포스트가 없으면 오류를 반환합니다. if (index === -1) { ctx.status = 404; ctx.body = { message: '포스트가 존재하지 않습니다.', }; return; } // index번째 아이템을 제거합니다. posts.splice(index, 1); ctx.status = 204; // No Content };

/ *포스트 수정(교체) PUT /api/posts/:id { title, body }*/ export const replace = ctx => { // PUT 메서드는 전체 포스트 정보를 입력하여 데이터를 통째로 교체할 때 사용합니다. const { id } = ctx.params; // 해당 id를 가진 post가 몇 번째인지 확인합니다. const index = posts.findIndex(p => p.id.toString() === id); // 포스트가 없으면 오류를 반환합니다. if (index === -1) { ctx.status = 404; ctx.body = { message: '포스트가 존재하지 않습니다.', }; return; } // 전체 객체를 덮어씌웁니다. // 따라서 id를 제외한 기존 정보를 날리고, 객체를 새로 만듭니다. posts[index] = { id, ...ctx.request.body, }; ctx.body = posts[index]; };

/ *포스트 수정(특정 필드 변경) PATCH /api/posts/:id { title, body }*/ export const update = ctx => { // PATCH 메서드는 주어진 필드만 교체합니다. const { id } = ctx.params; // 해당 id를 가진 post가 몇 번째인지 확인합니다. const index = posts.findIndex(p => p.id.toString() === id); // 포스트가 없으면 오류를 반환합니다. if (index === -1) { ctx.status = 404; ctx.body = { message: '포스트가 존재하지 않습니다.', }; return; } // 기존 값에 정보를 덮어씌웁니다. posts[index] = { ...posts[index], ...ctx.request.body, }; ctx.body = posts[index]; };

````
##### src/api/posts/index.js

```js
import Router from 'koa-router';
import * as postsCtrl from './posts.ctrl';

const posts = new Router();

posts.get('/', postsCtrl.list);
posts.post('/', postsCtrl.write);
posts.get('/:id', postsCtrl.read);
posts.delete('/:id', postsCtrl.remove);
posts.put('/:id', postsCtrl.replace);
posts.patch('/:id', postsCtrl.update);

export default posts;
````

src/api/index.js

```
import Router from 'koa-router';
import posts from './posts';

const api = new Router();

api.get('/test', ctx => {
  ctx.body = "<h1>Test Success!!!</h1>";
});

api.use('/posts', posts.routes());

// 라우터 내보내기
export default api;
```

src/main.js

```
require('dotenv').config();
import Koa from "koa";
import Router from 'koa-router';
import bodyParser from 'koa-bodyparser';
import mongoose from 'mongoose';

import api from './api';

// 비구조화 할당을 통해 process.env내부값에 대한 레퍼런스 만들기
const { PORT, MONGO_URI } = process.env;

mongoose
  .connect(MONGO_URI)
  .then(() => {
    console.log("Conntected to MongoDB");
  })
  .catch(e => {
    console.error(e);
})

const app = new Koa();
const router = new Router();
```
```

```
// 라우터설정
router.use('/api', api.routes()); // api 라우트 적용

// 라우터적용전에 bodyParser적용
app.use(bodyParser());

// app 인스턴스에 라우터 적용
app.use(router.routes()).use(router.allowedMethods());

// PORT가 지정되어 있지 않다면 4000사용
const port = PORT || 4000;

app.listen(port, () => {
  console.log('Listening to port %d', port)
});
```

- yarn start:dev
- postman
  - http://localhost:4000/api/posts

22.5 데이터베이스의 스키마와 모델

- mongodb에서의 schema와 model
  - schema : collection내부의 document의 각 필드의 형식을 정의
  - model : 스키마를 사용하여 만든 인스턴스, 실제 작업을 처리할 수 있는 함수들을 가지고 있는 객체
- Schema에서 기본적으로 지원하는 타입

타입	설명
String	문자열
Number	숫자
Date	날짜
Buffer	파일을 담을 수 있는 버퍼
Boolean	true or false
Mixed(Schema.Types.Mixed)	어떤 데이터도 넣을 수 있는 형식
ObjecId(Schema.Types.ObjecId)	객체아이디, 주로 다른 객체를 참조할 때 넣음
Array	배열 형태의 값을 []로 감싸서 사용

22.5.1 스키마 & 모델 생성

src/models/post.js

```
import mongoose from 'mongoose';

const { Schema } = mongoose;

const PostSchema = new Schema({
  title: String,
  body: String,
```

```
  tags: [String], // 문자열로 이루어진 배열
  publishedDate: {
    type: Date,
    default: Date.now, // 현재 날짜를 기본 값으로 지정
  },
});

const Post = mongoose.model('Post', PostSchema);

export default Post;
```

- model(스키마이름, 스키마객체)함수는 기본적으로 2개의 파라미터가 필요
- 스키마이름을 Post로 하면 스키마객체이름은 posts로 된다.
- 컬렉션이름을 만들 때 권장되는 컨벤션은 구분자를 사용하지 않고 복수형태로 사용
- 이 컨벤션을 준수하지 않는다면 세번째 파라미터에 원하는 이름을 입력하면 된다.

22.6 MongoDB Compas설치

22.7 데이터생성과 조회

22.7.1 데이터생성

src/api/posts/index.js

- replace는 구현하지 않을 것이기 때문에 posts.put('/:id', postsCtrl.replace);
  삭제

```
import Router from 'koa-router';
import * as postsCtrl from './posts.ctrl';

const posts = new Router();

posts.get('/', postsCtrl.list);
posts.post('/', postsCtrl.write);
posts.get('/:id', postsCtrl.read);
posts.delete('/:id', postsCtrl.remove);
posts.patch('/:id', postsCtrl.update);

export default posts;
```

src/api/posts/posts.ctl.js - write

```
/*
  POST /api/posts
  {
    title: '제목',
    body: '내용',
    tags: ['태그1', '태그2']
  }
*/
export const write = async ctx => {
  const {title, body, tags} = ctx.request.body;
  const post = new Post({
    title,
    body,
```

```
    tags,
  });
  try {
    await post.save();
    ctx.body = post;
  } catch (e) {
    ctx.throw(500, e);
  }
};
```

- 인스턴스를 만들 때 new 키워드를 사용, 그리고 생성자함수의 파라미터에 정보를 지닌 객체를 넣는다.
- 데이터베이스에 저장하기 위해서 save()함수를 실행해야 한다.
- 이 함수의 반환값은 Promise이기 때문에 async/await문법을 사용, await를 사용하려면 try cath으로 예외처리 해야 한다.
- postman으로 요청처리
  - POST http://localhost:4000/api/posts

src/api/posts/posts.ctl.js - wlist

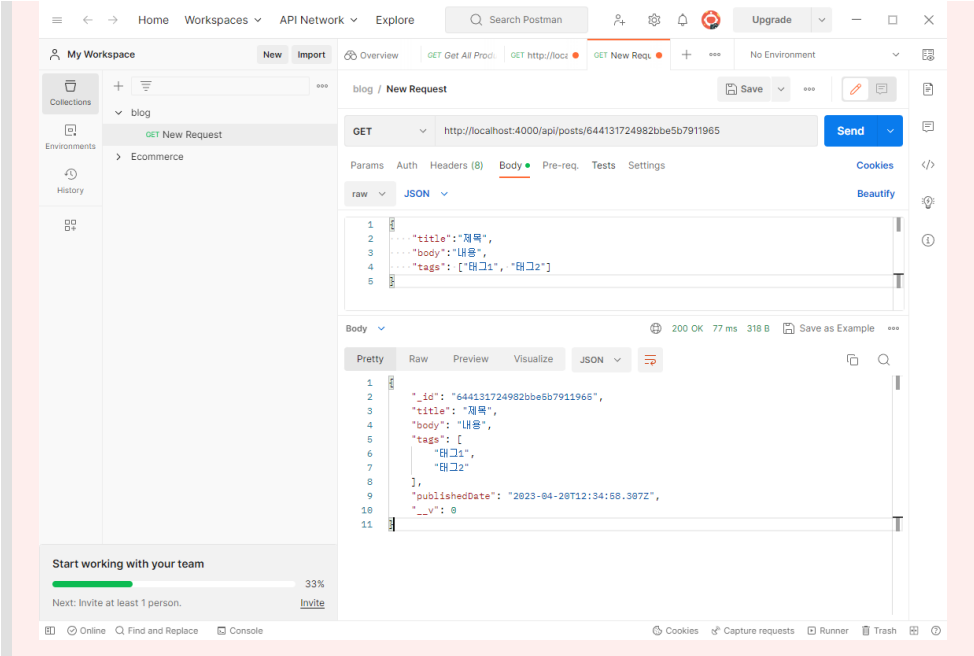
```
/*
  GET /api/posts
*/
export const list = async ctx => {
  try {
    const posts = await Post.find().exec();
    ctx.body = posts;
  } catch(e) {
    ctx.throw(500, e);
  }
};
```

- GET http://localhost:4000/api/posts

src/api/posts/posts.ctl.js - read

```
/*
  GET /api/posts/:id
*/
export const read = async ctx => {
  const {id} = ctx.params;
  try {
    const post = await Post.findById(id).exec();
    if(!post) {
      ctx.status = 404;
      return;
    }
    ctx.body = post;
  } catch(e) {
    ctx.throw(500, e);
  }
};
```

- GET http://localhost:4000/api/posts/644131724982bbe5b7911965



22.8 데이터 삭제와 수정

22.8.1 데이터 삭제

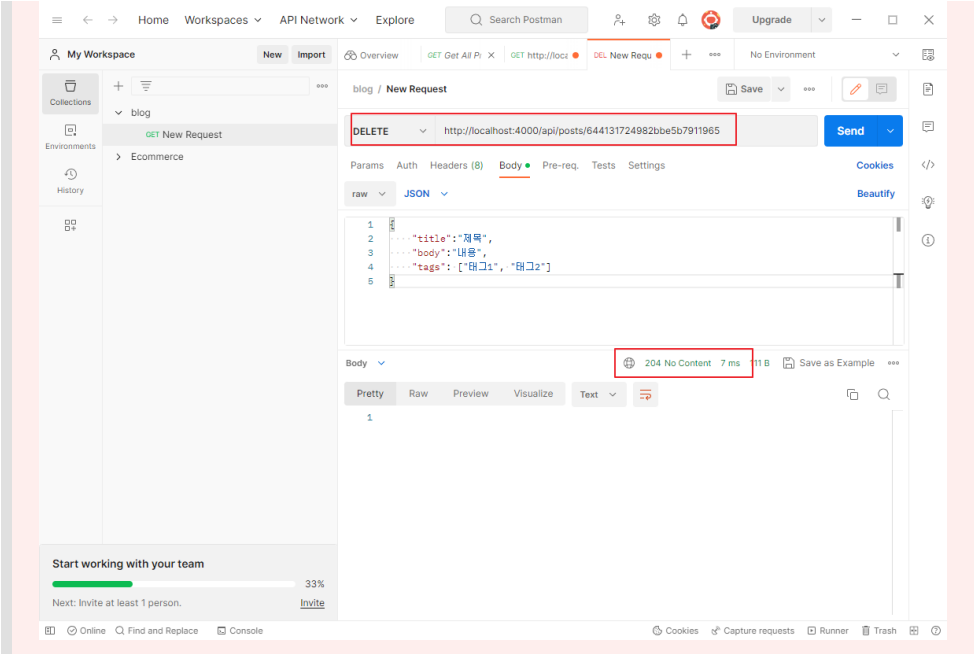
- remove() : 특정조건을 만족하는 데이터를 모두 삭제
- findByIdAndRemove() : 특정 id 삭제
- findOneAndRemove() : 특정조건 데이터 하나를 삭제

src/api/posts/posts.ctl.js - remove

```
/*
  DELETE /api/posts/:id
*/

export const remove = async ctx => {
  const {id} = ctx.params;
  try {
    const post = await Post.findByIdAndRemove(id).exec();
    ctx.status = 204; // no content 성공했지만 응답데이터 없음
    ctx.body = post;
  } catch(e) {
    ctx.throw(500, e);
  }
};
```

- DELETE http://localhost:4000/api/posts/644131724982bbe5b7911965

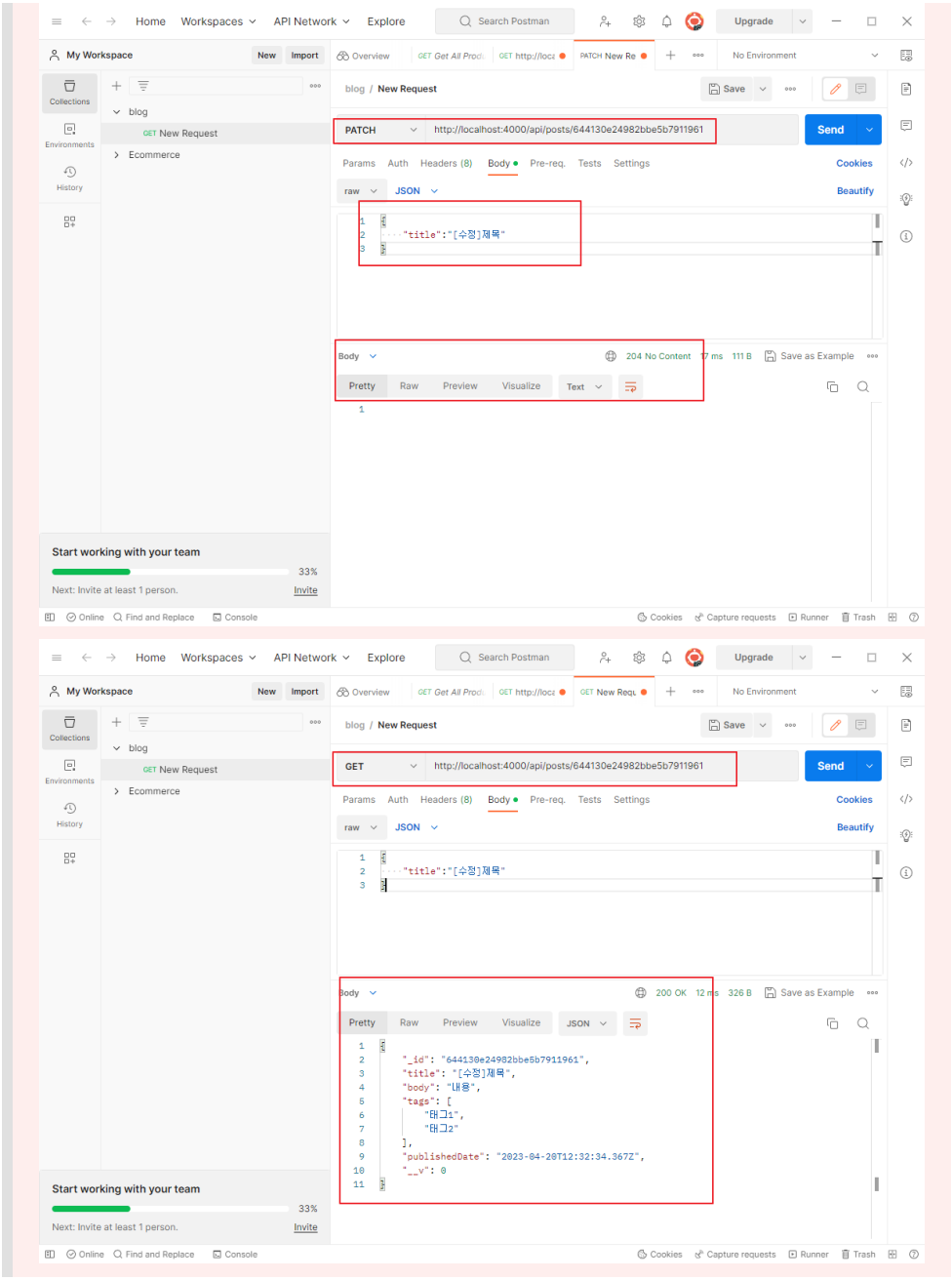


22.8.2 데이터 수정

src/api/posts/posts.ctl.js - update

```
/*
  DELETE /api/posts/:id
*/
export const update = async ctx => {
  const {id} = ctx.params;
  try {
    const post = await Post.findByIdAndUpdate(id, ctx.request.body, {
      new: true, // 이 값을 설정하면 업데이트된 데이터를 반환한다, false일 경우 업데이트전 값을 리턴
    }).exec();
    if(!post) {
      ctx.status = 404;
      return;
    }
    ctx.status = 204; // no content 성공했지만 응답데이터 없음
    ctx.body = post;
  } catch(e) {
    ctx.throw(500, e);
  }
};
```

- PATCH http://localhost:4000/api/posts/644131724982bbe5b7911965



22.9 요청검증

22.9.1 ObjectId 검증

- id가 ObjectId형식이 아니면 500에러가 발생 500에러는 보통 서버에서 처리하지 않아 내부적으로 문제시 발생
- ObjectId를 검증하는 방법 ```js import mongoose from 'moongose';

```
const { ObjectId } = mongoogse.Types; ObjectId.isValid(id);

    * 각 함수에 삽입하면 중복이 되기 때문에 미들웨어를 만들어서 적용

    ###### src/api/posts/posts.ctl.js - ObjecId 적용

    ```js
    import Post from '../../models/post';
    import mongoose from 'mongoose';
    import Joi from '@hapi/joi';
```
```

```
  const { ObjectId } = mongoose.Types;

  export const checkObjectId = (ctx, next) => {
    const { id } = ctx.params;
    if (!ObjectId.isValid(id)) {
      ctx.status = 400; // Bad Request
      return;
    }
    return next();
  };

  // 중략
```

src/api/posts/index.js

```
import Router from 'koa-router';
import * as postsCtrl from './posts.ctrl';

const posts = new Router();

posts.get('/', postsCtrl.list);
posts.post('/', postsCtrl.write);
posts.get('/:id', postsCtrl.checkObjectId, postsCtrl.read);
posts.delete('/:id', postsCtrl.checkObjectId, postsCtrl.remove);
posts.patch('/:id', postsCtrl.checkObjectId, postsCtrl.update);

export default posts;
```

src/api/posts/index.js - refactoring

```
import Router from 'koa-router';
import * as postsCtrl from './posts.ctrl';

const posts = new Router();

posts.get('/', postsCtrl.list);
posts.post('/', postsCtrl.write);

const post = new Router(); // /api/posts/:id
post.get('/', postsCtrl.read);
post.delete('/', postsCtrl.remove);
post.patch('/', postsCtrl.update);

posts.use('/:id', postsCtrl.checkObjectId, post.routes());

export default posts;
```

- /api/posts/:id 경로를 위한 라우터를 새로 만들고 posts에 해당 라우터를 등록

22.9.2 Request Body 검증

- 포스트를 작성할 때 서버는 title, body, tags값을 모두 전달 받아야 한다.
- 값이 누락되면 400오류가 발생해야 한다. 따라 처리하지 않으면 빈값이 등록된다
- if문으로 처리할 수 있지만 편하게 처리해주는 `joi 라이브러리를 설치`
- `yarn add joi`

src/api/posts/posts.ctl.js - Request Body 검증 by joi

```
import Post from '../../models/post';
import mongoose from 'mongoose';
import Joi from 'joi';

const { ObjectId } = mongoose.Types;

export const checkObjectId = (ctx, next) => {
  const { id } = ctx.params;
  if (!ObjectId.isValid(id)) {
    ctx.status = 400; // Bad Request
    return;
  }
  return next();
};

/*
  POST /api/posts
  {
    title: '제목',
    body: '내용',
    tags: ['태그1', '태그2']
  }
*/
export const write = async ctx => {
  const schema = Joi.object().keys({
    // 객체가 다음 필드를 가지고 있음을 검증
    title: Joi.string().required(), // required() 가 있으면 필수 항목
    body: Joi.string().required(),
    tags: Joi.array()
      .items(Joi.string())
      .required(), // 문자열로 이루어진 배열
  });

  // 검증 후, 검증 실패시 에러처리
  const result = schema.validate(ctx.request.body);
  if (result.error) {
    ctx.status = 400; // Bad Request
    ctx.body = result.error;
    return;
  }

  const { title, body, tags } = ctx.request.body;
  const post = new Post({
    title,
    body,
    tags,
  });
  try {
    await post.save();
    ctx.body = post;
  } catch (e) {
    ctx.throw(500, e);
  }
};

/*
```

```
  GET /api/posts
*/
export const list = async ctx => {
  try {
    const posts = await Post.find().exec();
    ctx.body = posts;
  } catch(e) {
    ctx.throw(500, e);
  }
};

/*
  GET /api/posts/:id
*/
export const read = async ctx => {
  const {id} = ctx.params;
  try {
    const post = await Post.findById(id).exec();
    if(!post) {
      ctx.status = 404;
      return;
    }
    ctx.body = post;
  } catch(e) {
    ctx.throw(500, e);
  }
};

/*
  DELETE /api/posts/:id
*/

export const remove = async ctx => {
  const {id} = ctx.params;
  try {
    const post = await Post.findByIdAndRemove(id).exec();
    ctx.status = 204; // no content 성공했지만 응답데이터 없음
    ctx.body = post;
  } catch(e) {
    ctx.throw(500, e);
  }
};

/*
  PATCH /api/posts/:id
  {
    title: '수정',
    body: '수정 내용',
    tags: ['수정', '태그']
  }
*/
export const update = async ctx => {

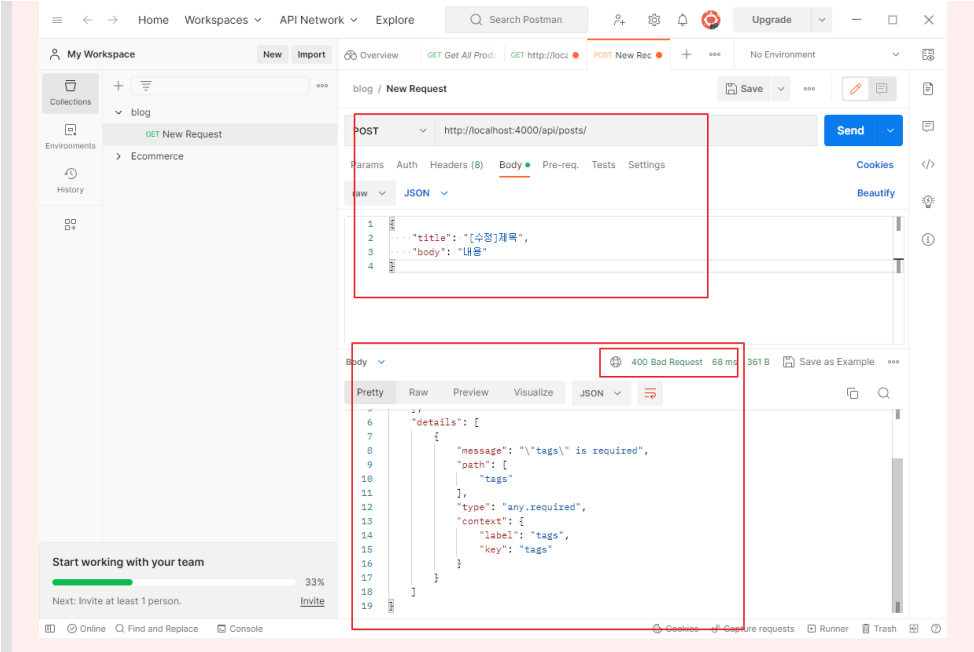
  const { id } = ctx.params;
  // write 에서 사용한 schema 와 비슷한데, required() 가 없습니다.
  const schema = Joi.object().keys({
    title: Joi.string(),
```

```
    body: Joi.string(),
    tags: Joi.array().items(Joi.string()),
  });

  // 검증 후, 검증 실패시 에러처리
  const result = schema.validate(ctx.request.body);
  if (result.error) {
    ctx.status = 400; // Bad Request
    ctx.body = result.error;
    return;
  }
  try {
    const post = await Post.findByIdAndUpdate(id, ctx.request.body, {
      new: true, // 이 값을 설정하면 업데이트된 데이터를 반환한다, false일 경우 업데이트전 값을 리턴
    }).exec();
    if(!post) {
      ctx.status = 404;
      return;
    }
    ctx.status = 204; // no content 성공했지만 응답데이터 없음
    ctx.body = post;
  } catch(e) {
    ctx.throw(500, e);
  }
};
```

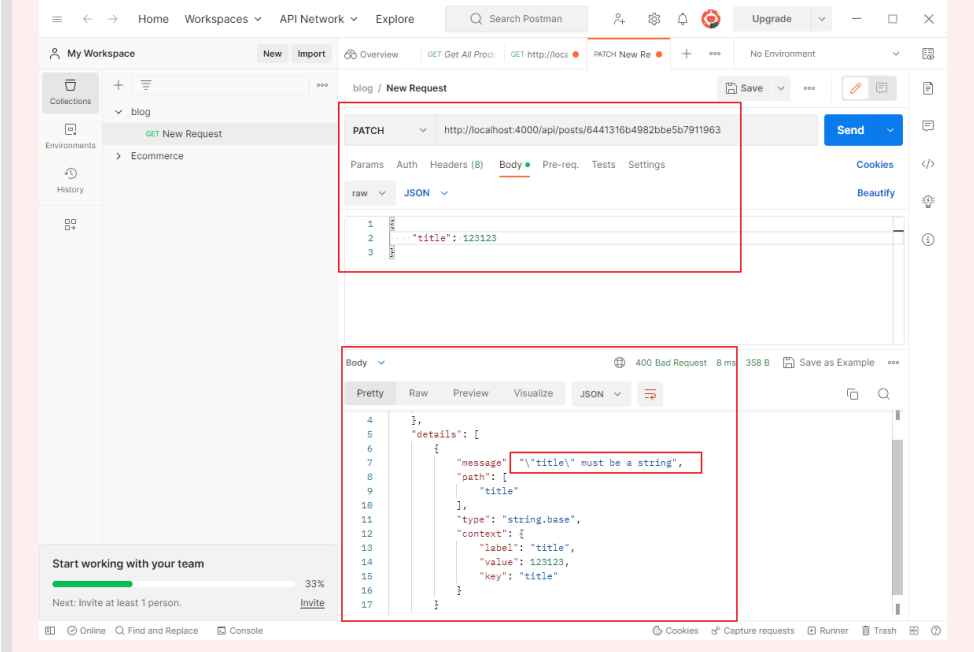
- POST http://localhost:4000/api/posts
- tags데이터 누락 검증

```
{
  "title": "[수정]제목",
  "body": "내용"
}
```



- PATCH http://localhost:4000/api/posts
- title 문자열이 아닐 경우 검증

```
{
  "title": 123123
}
```



22.10 페이지네이션 구현

20.10.1 dummy data 생성

src/createFakeData.js

```
import Post from './models/post';

export default function createFakeData() {
  // 0, 1, ... 39 로 이루어진 배열 생성 후 포스트 데이터로 변환
  const posts = [...Array(40).keys()].map(i => ({
    title: `포스트 #${i}`,
    // https://www.lipsum.com/ 에서 복사한 200자 이상 텍스트
    body:
      'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.',
    tags: ['가짜', '데이터'],
  }));
  Post.insertMany(posts, (err, docs) => {
    console.log(docs);
  });
}
```

src/main.js

```
require('dotenv').config();
import Koa from 'koa';
import Router from 'koa-router';
import bodyParser from 'koa-bodyparser';
import mongoose from 'mongoose';
```

```
import api from './api';
import createFakeData from './createFakeData';

// 비구조화 할당을 통해 process.env내부값에 대한 레퍼런스 만들기
const { PORT, MONGO_URI } = process.env;

mongoose
  .connect(MONGO_URI)
  .then(() => {
    console.log("Conntected to MongoDB");
    createFakeData();
  })
  .catch(e => {
    console.error(e);
})

const app = new Koa();
const router = new Router();

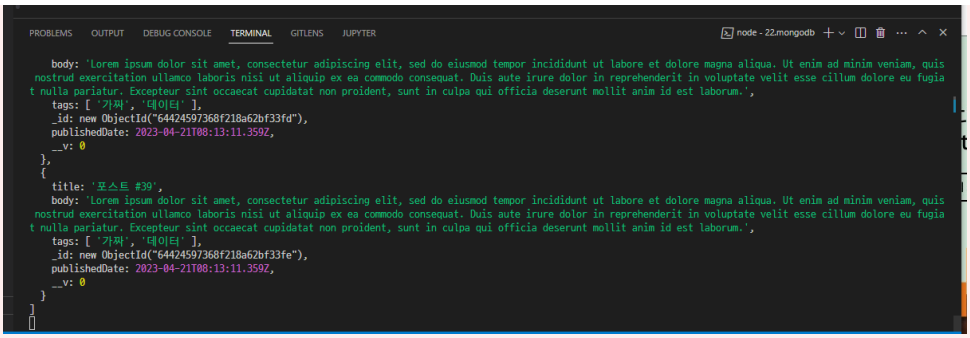
// 라우터설정
router.use('/api', api.routes()); // api 라우트 적용

// 라우터적용전에 bodyParser적용
app.use(bodyParser());

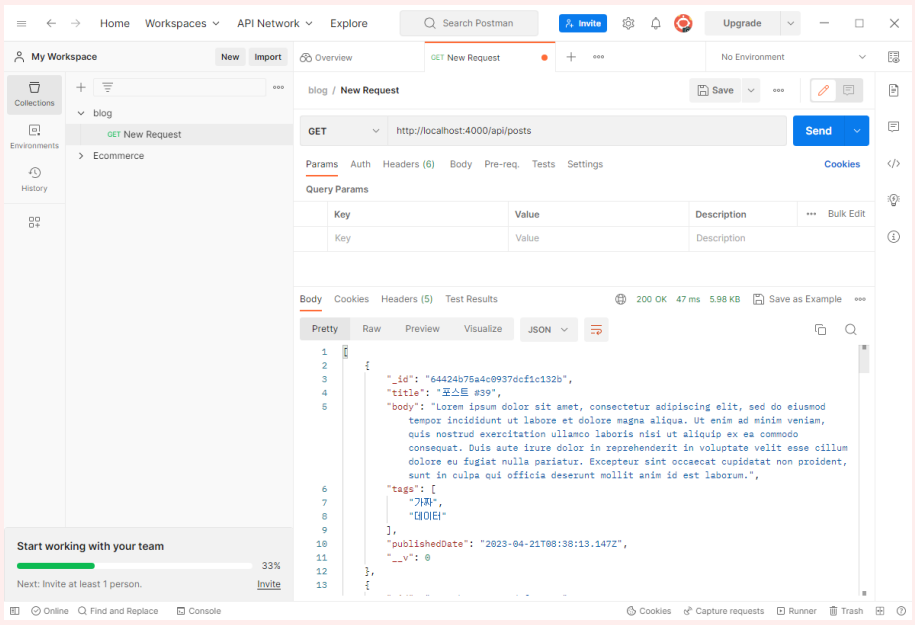
// app 인스턴스에 라우터 적용
app.use(router.routes()).use(router.allowedMethods());

// PORT가 지정되어 있지 않다면 4000사용
const port = PORT || 4000;

app.listen(port, () => {
  console.log('Listening to port %d', port)
});
```



- GET http://localhost:4000/api/posts



20.10.2

- 포스트를 역순으로 조회 , limt, page, last page
- 내용길이제한 : post.body.length < 200 ? post.body : `${post.body.slice(0, 200)}...` ,

src/api/posts/posts.ctrl.js - list descending / limit(10)

```
// 생략
/*
  GET /api/posts
*/
export const list = async ctx => {
  // query 는 문자열이기 때문에 숫자로 변환해주어야합니다.
  // 값이 주어지지 않았다면 1 을 기본으로 사용합니다.
  const page = parseInt(ctx.query.page || '1', 10);

  if (page < 1) {
    ctx.status = 400;
    return;
  }

  try {
    const posts = await Post.find()
      .sort({ _id: -1 })
      .limit(10)
      .skip((page - 1) * 10)
      .lean()
      .exec();
    const postCount = await Post.countDocuments().exec();
    ctx.set('Last-Page', Math.ceil(postCount / 10));
    ctx.body = posts.map(post => ({
      ...post,
      body:
        post.body.length < 200 ? post.body : `${post.body.slice(0, 200)}...`,
    }));
  } catch (e) {
```

```
        ctx.throw(500, e);
    }
};
```