고급 웹프로그래밍 과제3

2020113486 김대건

1. 웹 서버 인프라 환경

배포환경 : AWS EC2
프론트엔드 : TypeScript, NextJS
백엔드 : TypeScript, NestJS, TypeORM, SQLite
https://github.com/toothlessdev/web-programming/tree/main/assignment3

2.

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { TypeOrmModule } from '@nestjs/typeorm';
import { PostsModule } from './posts/posts.module';
import { PostModel } from './posts/model/post.model';

const TypeORMRootModule = TypeOrmModule.forRoot({
  type: 'sqlite',
  database: 'database.db',
  synchronize: true,
  entities: [PostModel],
});

@Module({
  imports: [TypeORMRootModule, PostsModule],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

먼저 데이터베이스를 연동합니다.
SQLite 를 TypeORM 과 연동해 줍니다.

```
import {
  Controller,
  Param,
  Get,
  Post,
  Delete,
  Patch,
  Body,
  Query,
  BadRequestException,
} from '@nestjs/common';
import { PostsService } from './posts.service';
import { CreatePostDto } from './dto/create-post.dto';
import { UpdatePostDto } from './dto/update-post.dto';

@Controller('posts')
export class PostsController {
```

```
  constructor(private readonly postsService: PostsService) {}

  @Get()
  public async readPosts(
    @Query('page') page: string,
    @Query('per_page') perPage: string,
  ) {
    if (!page || !perPage)
      throw new BadRequestException(
        'page Param 과 perPage Param 은 필수입니다',
      );

    return this.postsService.readPosts(parseInt(page), parseInt(perPage));
  }

  @Get(':id')
  public async readPostById(@Param('id') id: string) {
    return this.postsService.readPostById(Number(id));
  }

  @Post()
  public async createPost(@Body() body: CreatePostDto) {
    return this.postsService.createPost(body);
  }

  @Patch(':id')
  public async updatePost(
    @Param('id') id: string,
    @Body() body: UpdatePostDto,
  ) {
    return this.postsService.updatePost(Number(id), body);
  }

  @Delete(':id')
  public async deletePost(@Param('id') id: string) {
    return this.postsService.deletePost(Number(id));
  }
}
```

먼저 요청에 대해 각 포스트들을 응답으로 보여줄 WAS 를 구성합니다. PostController 를 만들어 해당 요청을 핸들링 하고, PostService 를 이용해 해당 요청을 처리합니다.

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { Repository } from 'typeorm';
import { PostModel } from './model/post.model';
import { InjectRepository } from '@nestjs/typeorm';
import { CreatePostDto } from './dto/create-post.dto';
import { UpdatePostDto } from './dto/update-post.dto';

@Injectable()
export class PostsService {
  constructor(
    @InjectRepository(PostModel)
    private readonly postsRepository: Repository<PostModel>,
  ) {}
```

```
  public async readPosts(page: number, perPage: number) {
    const posts = await this.postsRepository.find({
      take: perPage,
      skip: perPage * (page - 1),
      select: ['id', 'title', 'author', 'createdAt'],
    });

    return { posts, page, perPage };
  }

  public async readPostById(id: number) {
    return this.postsRepository.findOne({ where: { id } });
  }

  public async createPost(body: CreatePostDto) {
    const newPost = await this.postsRepository.create(body);
    return this.postsRepository.save(newPost);
  }

  public async updatePost(id: number, body: UpdatePostDto) {
    const post = await this.postsRepository.findOne({ where: { id } });
    if (!post) throw new NotFoundException();

    for (const key of Object.keys(body)) post[key] = body[key];

    return this.postsRepository.save(post);
  }

  public async deletePost(id: number) {
    const post = await this.postsRepository.findOne({ where: { id } });
    if (!post) throw new NotFoundException();

    return this.postsRepository.delete(id);
  }
}
```

Post Service 에서는 postsRepository 를 NestJS로 부터 의존성 주입을 받아 Post Model 을 조회, 수정, 삭제 합니다. readPosts 에서는 controller 로 부터 받은 searchParam 인 page 와 per_page 를 받아 페이지네이션 합니다.

프론트엔드는 NextJS Page Router 를 사용해 구현하였습니다

```tsx
import { Controller } from "@/components/Controller";
import { Post } from "@/components/Post";
import { postService } from "@/services/post.service";
import { Post as PostType } from "@/services/post.types";
import styles from "@/styles/Home.module.css";
import { GetServerSideProps, GetServerSidePropsContext } from "next";

export interface PageProps {
    posts: Omit<PostType, "content">[];
}

export default function Home(props: PageProps) {
    return (
        <main className={styles["main"]}>
            <Controller maxPage={5} />

            {props.posts.map((element) => {
                return <Post key={element.id} id={element.id} title={element.title}
author={element.author} createdAt={element.createdAt}></Post>;
            })}
        </main>
    );
}

export const getServerSideProps: GetServerSideProps<PageProps> = async (context:
GetServerSidePropsContext) => {
    let page = context.query.page ?? "1";
    let per_page = context.query.per_page ?? "10";

    const data = await postService.readPosts(page as string, per_page as string);

    return {
        props: {
            posts: data.posts,
        },
    };
};
```

먼저 Home Page 에서는 각 Posts 들을 page 와 per_page searchParam 에 맞게 getServerSideProps 함수를 통해 서버 사이드에서 데이터를 가져와 사전 렌더링한 HTML 을 클라이언트에 전송합니다.


```tsx
import { useRouter } from "next/router";
import styles from "./Post.module.css";

export interface IPost {
    id: number;
    title: string;
    author: string;
    createdAt: string;
}

export const Post: React.FC<IPost> = ({ id, title, author, createdAt }) => {
    const router = useRouter();

    return (
        <div
            className={styles["post-wrapper"]}
            onClick={() => {
                router.push(`/posts/${id}`);
            }}>
            <h3>{title}</h3>
            <p>글쓴이 : {author}</p>
            <p>작성시간 : {createdAt}</p>
        </div>
    );
};
```

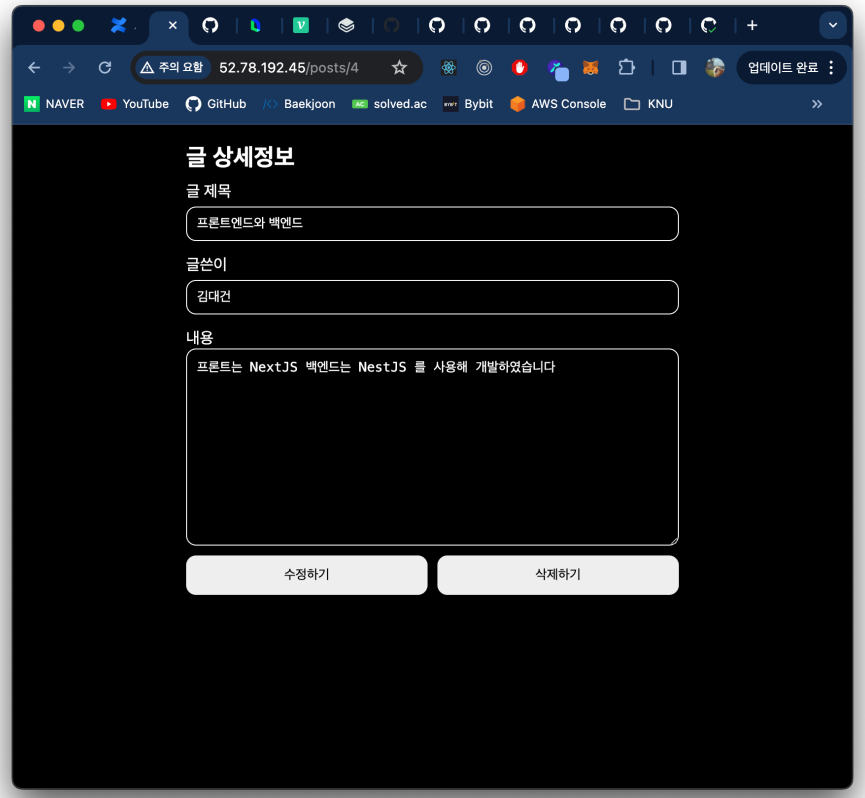
post 컴포넌트 입니다. 해당 컴포넌트를 클릭시에, 포스트에 대한 상세 페이지인 /posts/${id} 로 이동합니다

```tsx
import { postService } from "@/services/post.service";
```

```tsx
import styles from "@/styles/NewPostPage.module.css";
import { GetServerSideProps, GetServerSidePropsContext } from "next";
import { useRouter } from "next/router";

export interface PageProps {
    title: string;
    author: string;
    content: string;
    createdAt: string;
}

export default function PostDetailPage(props: PageProps) {
    const router = useRouter();

    const onEditBtnClicked = () => {
        router.push(`/posts/edit/${router.query.id}`);
    };
    const onDeleteBtnClicked = () => {
        postService.deletePost(Number(router.query.id)).then(() => {
            alert("글이 성공적으로 삭제되었습니다");
            router.push(`/`);
        });
    };

    return (
        <main className={styles["main"]}>
            <h2>글 상세정보</h2>

            <div className={styles["input-item"]}>
                <label htmlFor="title">글 제목</label>
                <input id="title" type="text" value={props.title} />
            </div>

            <div className={styles["input-item"]}>
                <label htmlFor="author">글쓴이</label>
                <input id="author" type="text" value={props.author} />
            </div>

            <div className={styles["input-item"]}>
                <label htmlFor="content">내용</label>
                <textarea name="content" id="content" cols={30} rows={10}
value={props.content}></textarea>
            </div>

            <div className={styles["input-item"]} style={{ display: "flex", gap: "10px" }}>
                <button onClick={onEditBtnClicked}>수정하기</button>
                <button onClick={onDeleteBtnClicked}>삭제하기</button>
            </div>
        </main>
    );
}

export const getServerSideProps: GetServerSideProps<PageProps> = async (context:
GetServerSidePropsContext) => {
    const { id } = context.query;
```

```
    const { title, author, content, createdAt } = await postService.readPostById(Number(id));

    return {
        props: { title, author, content, createdAt },
    };
};
```

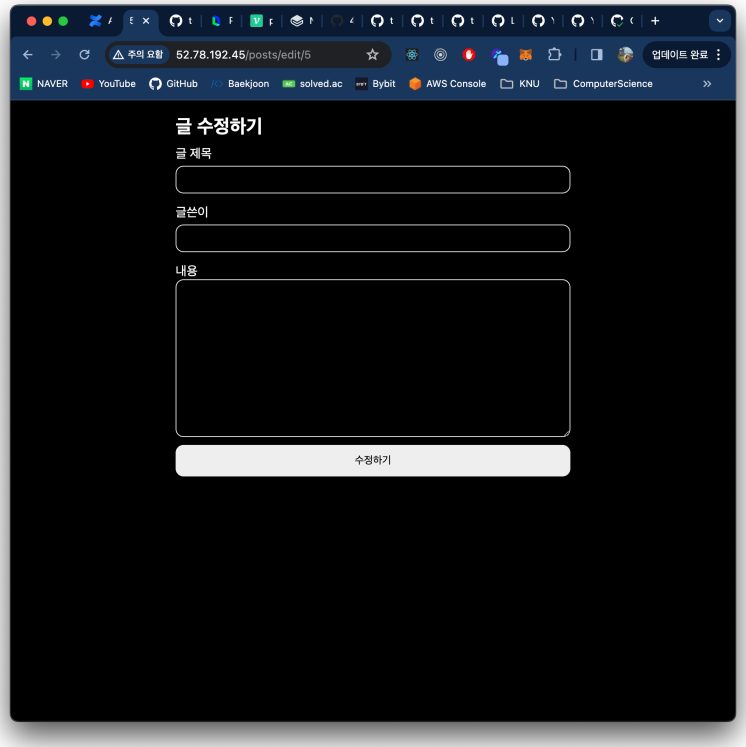
Post 에 대한 상세 페이지입니다.
getServerSideProps 를 통해 NestJS 서버로부터 데이터 페칭 후, pageProps 로 넘겨주어 서버사이드 렌더링을 진행합니다.




```
import { postService } from "@/services/post.service";
import styles from "@/styles/NewPostPage.module.css";
import { GetServerSideProps, GetServerSidePropsContext } from "next";
import { useRouter } from "next/router";
import { useRef } from "react";

export default function EditPostPage() {
    const router = useRouter();

    const titleRef = useRef<HTMLInputElement>(null);
    const authorRef = useRef<HTMLInputElement>(null);
    const contentRef = useRef<HTMLTextAreaElement>(null);

    const onEditBtnClicked = () => {
        postService
            .patchPost(Number(router.query.id), {
                title: titleRef.current?.value as string,
```

```
                author: authorRef.current?.value as string,
                content: contentRef.current?.value as string,
            })
            .then(() => {
                alert("글이 성공적으로 수정되었습니다");
                router.push(`/`);
            });
    };

    return (
        <main className={styles["main"]}>
            <h2>글 수정하기</h2>

            <div className={styles["input-item"]}>
                <label htmlFor="title">글 제목</label>
                <input id="title" type="text" ref={titleRef} />
            </div>

            <div className={styles["input-item"]}>
                <label htmlFor="author">글쓴이</label>
                <input id="author" type="text" ref={authorRef} />
            </div>

            <div className={styles["input-item"]}>
                <label htmlFor="content">내용</label>
                <textarea name="content" id="content" cols={30} rows={10}
ref={contentRef}></textarea>
            </div>

            <div className={styles["input-item"]} style={{ display: "flex", gap: "10px" }}>
                <button onClick={onEditBtnClicked}>수정하기</button>
            </div>
        </main>
    );
}
```

글 수정 페이지 입니다.



글 상세정보 페이지에서 글 수정하기 버튼을 누르면 수정하기 페이지로 이동합니다.
글 제목 글쓴이 내용 을 수정한 후 수정하기 버튼을 누르면 수정됩니다.

글 삭제하기 버튼을 누르면 글이 삭제됩니다.

```
const API_BASE_URL = process.env.NEXT_PUBLIC_API_BASE_URL;

export const api = {
    Get: async <ResponseBody>(url: string) => {
        const response = await fetch(API_BASE_URL + url, {
            headers: {
                "Content-Type": "application/json",
            },
        });
        if (!response.ok) throw new Error("Get Request Failed");
        const data = await response.json();
        return data as ResponseBody;
    },
    Post: async <RequestBody, ResponseBody>(url: string, body: RequestBody) => {
        const response = await fetch(API_BASE_URL + url, {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify(body),
        });
        if (!response.ok) throw new Error("Post Request Failed");
        const data = await response.json();
        return data as ResponseBody;
    },
    Patch: async <RequestBody, ResponseBody>(url: string, body: RequestBody) => {
        const response = await fetch(API_BASE_URL + url, {
            method: "PATCH",
```

```
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify(body),
        });
        if (!response.ok) throw new Error("PATCH Request Failed");
        const data = await response.json();
        return data as ResponseBody;
    },
    Delete: async (url: string) => {
        const response = await fetch(API_BASE_URL + url, {
            method: "DELETE",
            headers: {
                "Content-Type": "application/json",
            },
        });
        if (!response.ok) throw new Error("Delete Request Failed");
        const data = await response.json();
        return data;
    },
};
```

클라이언트측 데이터페칭은 각 요청별로 Get, Post, Patch, Delete 메서드를 추상화하였습니다.

```
import { api } from "./api";
import { ICreatePostRequestBody, IReadPostByIdResponseBody, IReadPostsResponseBody } from
"./post.types";

export const postService = {
    readPosts: async (page: string, per_page: string) => {
        return api.Get<IReadPostsResponseBody>(`/posts?page=${page}&per_page=${per_page}`);
    },
    readPostById: async (id: number) => {
        return api.Get<IReadPostByIdResponseBody>(`/posts/${id}`);
    },
    createPost: async (body: ICreatePostRequestBody) => {
        return api.Post(`/posts`, body);
    },
    deletePost: async (id: number) => {
        return api.Delete(`/posts/${id}`);
    },
    patchPost: async (id: number, body: ICreatePostRequestBody) => {
        return api.Patch(`/posts/${id}`, body);
    },
};
```

이후 Service Layer 에서 해당 api 를 이용해 데이터를 페칭합니다.

http://52.78.192.45/
해당서비스는 여기서 사용하실 수 있습니다.