웹서버보안프로그래밍

4. MongoDB CRUD

중부대학교 정보보호학과
이병천 교수
sultan@joongbu.ac.kr

전체 목차

1. 강의 개요
2. 웹개발환경 구축
3. Next.js 소개
4. **MongoDB CRUD**
5. 쇼핑몰 예제 프로젝트
6. PBL 팀프로젝트 발표

5. MongoDB CRUD Example

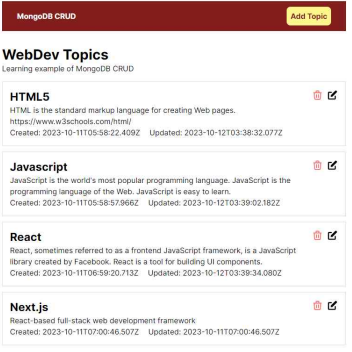
1. MongoDB CRUD
2. Next-Auth 인증
3. Deploy

MongoDB

- MongoDB란?
  - No-SQL, 문서 기반 DB
  - JSON object를 그대로 저장하므로 빠르고 활용이 쉬움
  - 무료 클라우드 서비스를 제공 https://www.mongodb.com/

예제 프로젝트

- 소스코드
  - https://github.com/lbcsultan/mongodb-crud
- 웹사이트
  - https://mongodb-crud-sooty.vercel.app/
- 동영상 강의
  - https://www.youtube.com/watch?v=wNWyMsrpbz0
- 사용 기술
  - Next.js, Tailwind CSS
  - Mongoose
  - MongoDB CRUD (Create, Read, Update, Delete)



1. 템플릿 프로젝트 생성

- 생성된 프로젝트 폴더 열기
- 새로운 프로젝트 생성
  - > npx create-next-app@latest .
- 필요한 패키지 설치하기
  - > npm install mongoose react-icons
  - Mongoose: MondoDB 서비스에 연결하는 기능을 가짐
  - React-icons: 아이콘 기능 활용 예정
- 개발 서버 열기
  - > npm run dev

```
D:₩WebDev₩2024-2₩nextjs₩crud>npx create-next-app@latest .
Need to install the following packages:
create-next-app@14.2.15
Ok to proceed? (y)

√ Would you like to use TypeScript? ... No / Yes
√ Would you like to use ESLint? ... No / Yes
√ Would you like to use Tailwind CSS? ... No / Yes
√ Would you like to use `src/` directory? ... No / Yes
√ Would you like to use App Router? (recommended) ... No / Yes
√ Would you like to customize the default import alias (@/*)? ... No / Yes
Creating a new Next.js app in D:₩WebDev₩2024-2₩nextjs₩crud.
```

```
"dependencies": {
    "mongoose": "^8.7.1",
    "next": "14.2.15",
    "react": "^18",
    "react-dom": "^18",
    "react-icons": "^5.3.0"
  },
```

Frontend UI 개발 - 메뉴 생성, 등록

/components/Navbar.tsx

```tsx
import Link from 'next/link'

export default function Navbar() {
  return (
    <nav className="flex justify-between items-center bg-red-900
px-8 py-4">
      <Link className="text-white text-lg font-bold " href="/">
        MongoDB CRUD
      </Link>
      <Link
        className="bg-yellow-200 text-lg font-bold px-4 py-2
rounded-md"
        href="/addTopic"
      >
        Add Topic
      </Link>
    </nav>
  )
}
```

/app/layout.tsx

```tsx
import type { Metadata } from 'next'
import localFont from 'next/font/local'
import './globals.css'
import Navbar from '@/components/Navbar'

const geistSans = localFont({
  src: './fonts/GeistVF.woff',
  variable: '--font-geist-sans',
  weight: '100 900',
})
const geistMono = localFont({
  src: './fonts/GeistMonoVF.woff',
  variable: '--font-geist-mono',
  weight: '100 900',
})

export const metadata: Metadata = {
  title: 'MogngoDB CRUD',
  description: 'Create, Read, Update, and Delete in MongoDB',
}

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode
}>) {
  return (
    <html lang="en">
      <body
        className={`${geistSans.variable} ${geistMono.variable} antialiased`}
      >
        <div className="max-w-4xl mx-auto">
          <Navbar />
          <div className="mt-8">{children}</div>
        </div>
      </body>
    </html>
  )
}
```

Hello World!

필요한 Component 생성

/components/TopicsList.tsx

```
import Link from 'next/link'
import { HiPencilAlt } from 'react-icons/hi'
import RemoveBtn from './RemoveBtn'
export default function TopicsList() {
  return (
    <>
      <div className="p-4 border border-slate-300 my-3 flex justify-between
gap-5 items-start">
        <div>
          <h2 className="text-2xl font-bold">Topic Title</h2>
          <div>Topic description</div>
        </div>
        <div className="flex gap-2">
          <RemoveBtn />
          <Link href={'/editTopic/123'}>
            <HiPencilAlt size={24} />
          </Link>
        </div>
      </div>
    </>
  )
}
```

/components/RemoveBtn.tsx

```
import { HiOutlineTrash } from 'react-icons/hi'
export default function RemoveBtn() {
  return (
    <button className="text-red-400">
      <HiOutlineTrash size={24} />
    </button>
  )
}
```

메인페이지에서 컴포넌트 활용

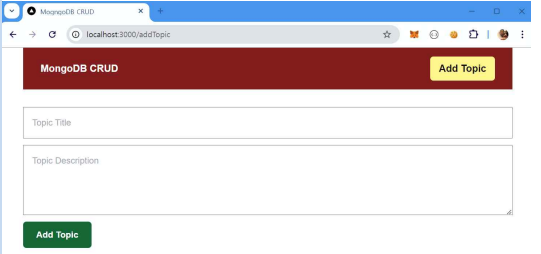
/app/page.tsx

```
import TopicsList from '@/components/TopicsList'

export default function Home() {
  return (
    <div>
      <h1 className="text-3xl font-bold">WebDev Topics</h1>
      <p className="mb-4">MongoDB CRUD Example</p>
      <TopicsList />
      <TopicsList />
    </div>
  )
}
```

Add Topic 페이지 생성

/app/addTopic/page.tsx

```
export default function AddTopic() {
  return (
    <form className="flex flex-col gap-3">
      <input
        className="border border-slate-500 p-4"
        type="text"
        placeholder="Topic Title"
      />
      <textarea
        className="border border-slate-500 p-4 h-32"
        placeholder="Topic Description"
      />
      <button className="bg-green-800 text-white font-bold px-6
py-3 w-fit rounded-md">
        Add Topic
      </button>
    </form>
  )
}
```



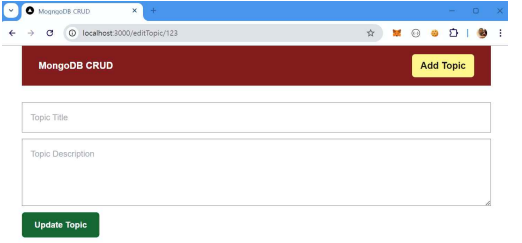
Edit Topic 페이지 생성

/app/components/EditTopicForm.tsx

```
export default function EditTopicForm() {
  return (
    <form className="flex flex-col gap-3">
      <input
        className="border border-slate-500 p-4"
        type="text"
        placeholder="Topic Title"
      />
      <textarea
        className="border border-slate-500 p-4 h-32"
        placeholder="Topic Description"
      />
      <button className="bg-green-800 text-white font-bold px-6
py-3 w-fit rounded-md">
        Update Topic
      </button>
    </form>
  )
}
```

/app/editTopic/[id]/page.tsx

```
import EditTopicForm from '@/components/EditTopicForm'

export default function EditTopic() {
  return <EditTopicForm />
}
```



MongoDB 사용 준비

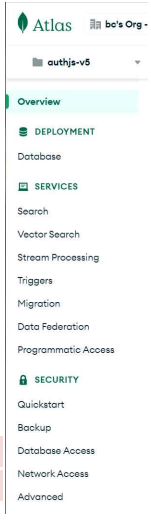
- MongoDB Atlas
  - 무료로 사용할 수 있는 클라우드 DB 서비스 제공
  - 홈페이지 : https://www.mongodb.com/
  - 회원가입 : https://www.mongodb.com/cloud/atlas/register
  - 로그인 : https://account.mongodb.com/account/login

- 용어 계층
  - Cluster (물리적 서버의 집합)
  - Organization 생성 (여러 프로젝트를 관리하는 데 사용되는 논리적 컨테이너)
  - Project 생성 (데이터베이스의 집합)
  - Database 생성 (데이터를 저장하는 논리적 컨테이너)
  - Collection 생성 (데이터 테이블)

MongoDB 사용 준비

클러스터

프로젝트



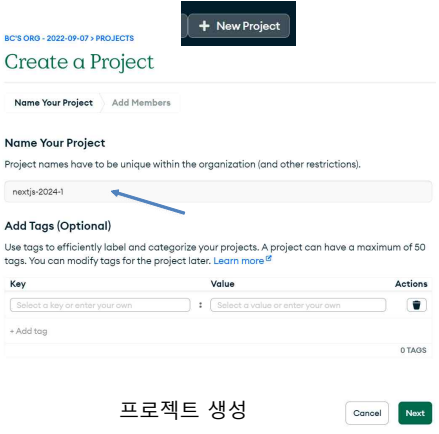
- DB 사용자 생성 (Database access)
  - 웹서비스에서 DB에 접근하기 위한 사용자 생성.
  - 안전한 패스워드 설정. 개발시와 서비스시 패스워드 설정 확인.
  - 한 프로젝트에서 여러 개의 ID 생성 가능
  - DB에의 read/write 권한 설정
- 네트워크 접근 권한 설정 (Network access)
  - 특정 IP만 접근을 허용하도록 설정 가능
  - 모든 IP에서 접근 가능 설정 : 0.0.0.0/0
  - 노트북 환경에서는 IP가 수시로 바뀌므로 0.0.0.0/0로 설정할 필요가 있음
- **프로젝트 생성**
  - 새 프로젝트를 생성하면 사용자 생성과 네트워크 접근권한 설정을 함께 진행
  - 사용자를 여러 개 생성할 수 있음
  - 한 프로젝트 내에서 여러 개의 데이터베이스를 생성 가능

Database

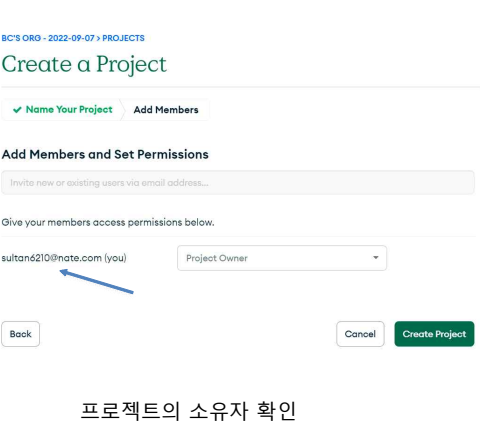
Database access

Network access

새 프로젝트 생성



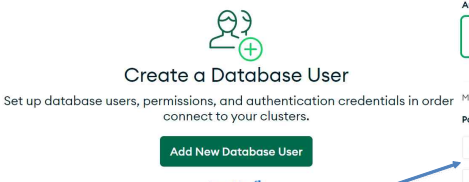
프로젝트 생성

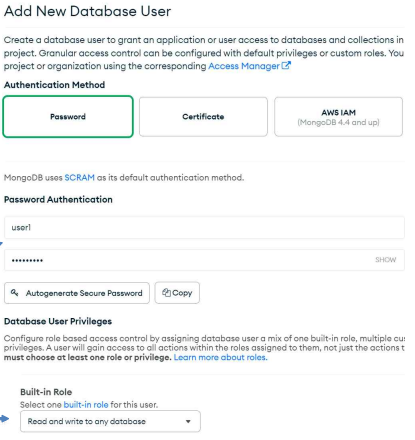


프로젝트의 소유자 확인

Database user 생성 및 권한 설정

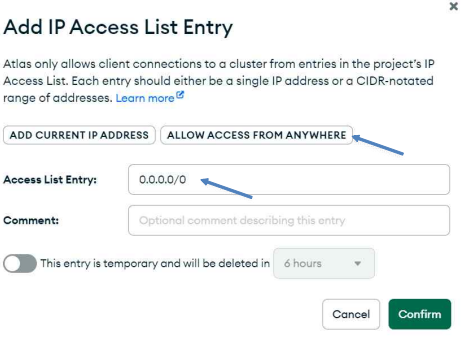
- Database access 클릭





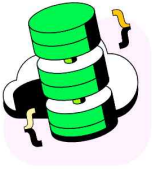
접근제어 설정

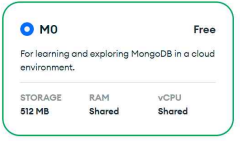
- Network access 클릭

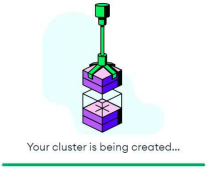


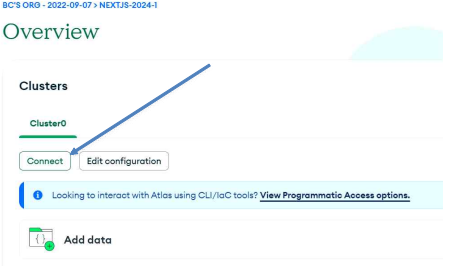
클러스터 생성

- Database 클릭



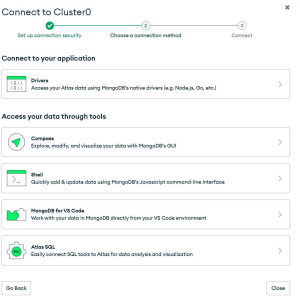






Connection string 생성 및 저장

- Connection string 생성
  - Connect > MongoDB for VS Code
  - Connection string에 ID, password, db명 입력
- 환경변수 저장
  - Connection string을 .env 또는 .env.local 파일에 저장



mongodb+srv://user1:<db_password>@cluster0.cixb7.mongodb.net/

환경변수에 저장

- .env 파일에 저장

```
MONGODB_URI=mongodb+srv://user1:user11234@cluster0.cixb7.mongodb.net/crud
```

아이디 패스워드 DB명

- 환경변수가 바뀌면 개발서버를 다시 시작
  - > npm run dev

Mongoose

- Mongoose란?
  - Node.js와 MongoDB를 위한 **ODM(Object Data Mapping) 라이브러리**
  - https://mongoosejs.com/
  - https://mongoosejs.com/docs/guide.html

Mongoose

- Connect()
  - MongoDB에 연결
- Schema()
  - MongoDB에 저장할 데이터의 형태를 지정, MongoDB collection과 데이터를 매핑
- Model()
  - Schema를 컴파일해서 만드는 생성자, 인스턴스 객체
  - MongoDB에서 문서를 만들고 읽어내는 기본 단위
- Model Methods
  - Create: save(), create(), insertMany()
  - Query: find(), findOne()
  - Delete: deleteOne()
  - Update: updateOne()

MongoDB 연결 함수 생성

/libs/mongodb.ts

```
import mongoose from 'mongoose'

export default async function connectMongoDB() {
  try {
    await mongoose.connect(process.env.MONGODB_URI as string)
    console.log('Connected to MongoDB')
  } catch (error) {
    console.log(error)
  }
}
```

데이터 모델 생성

/models/topic.ts

```
import mongoose, { Schema } from 'mongoose'

const topicSchema = new Schema(
  {
    title: String,
    description: String,
  },
  {
    timestamps: true,
  }
)
const Topic = mongoose.models.Topic || mongoose.model('Topic', topicSchema)
export default Topic
```

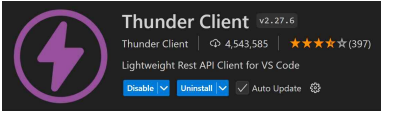
BackEnd CRUD 기능 구현

- Backend API 폴더 생성
  - /app/api/topics
  - route.ts 파일 생성
  - POST 기능 구현

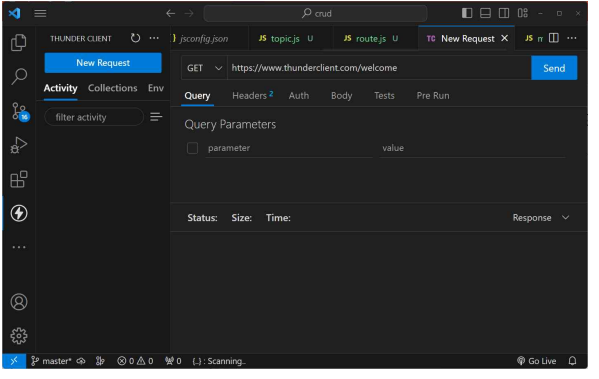
/app/api/topics/route.ts

```
import connectMongoDB from '@/libs/mongodb'
import Topic from '@/models/topic'
import { NextRequest, NextResponse } from 'next/server'

export async function POST(request: NextRequest) {
  try {
    const { title, description } = await request.json()
    if (!title || !description) {
      return NextResponse.json(
        { message: 'Title and description are required' },
        { status: 400 }
      )
    }
    await connectMongoDB()
    await Topic.create({ title, description })
    return NextResponse.json({ message: 'Topic created' }, { status: 201 })
  } catch (error) {
    console.error('Error in POST /api/topics:', error)
    return NextResponse.json(
      { message: 'Internal Server Error' },
      { status: 500 }
    )
  }
}
```

Thunder Client를 이용한 동작 테스트

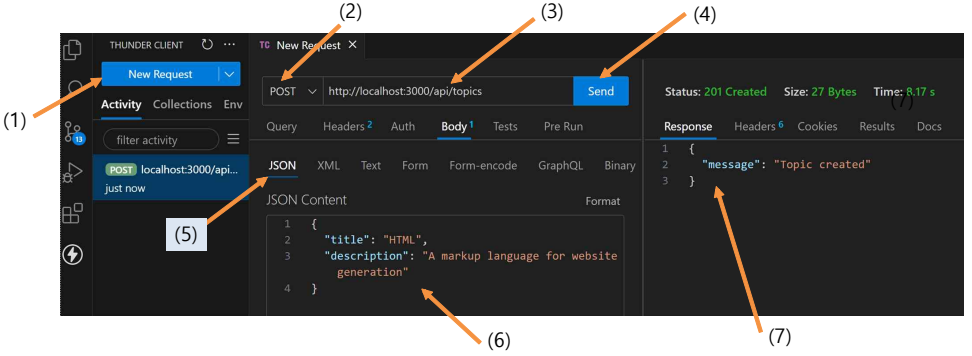


- Thunder Client 확장 프로그램 설치



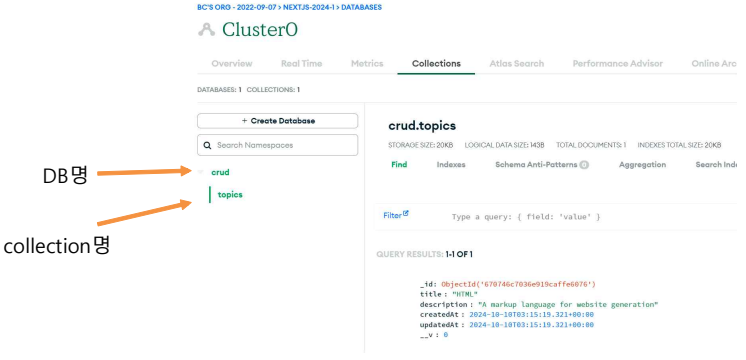
Thunder Client를 이용한 POST 테스트

- New POST Request 생성, Response 테스트



MongoDB에서 저장 내용 확인

- Database → [illegible]



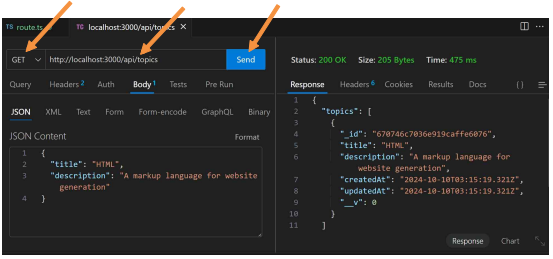
_id 자동 생성
title, description 저장
timestamp 자동 저장

GET 기능 구현

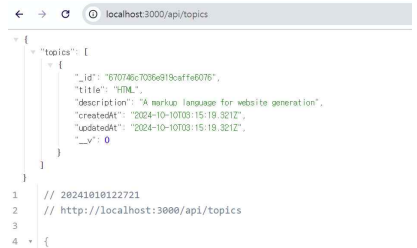
/app/api/topics/route.ts

```
import connectMongoDB from '@/libs/mongodb'
import Topic from '@/models/topic'
import { NextRequest, NextResponse } from 'next/server'

export async function POST(request: NextRequest) {

...중략...

}

export async function GET() {
  try {
    await connectMongoDB()
    const topics = await Topic.find()
    return NextResponse.json({ topics })
  } catch (error) {
    console.error('Error in GET /api/topics:', error)
    return NextResponse.json(
      { message: 'Internal Server Error' },
      { status: 500 }
    )
  }
}
```

GET 기능 구현



Thunder client로 확인



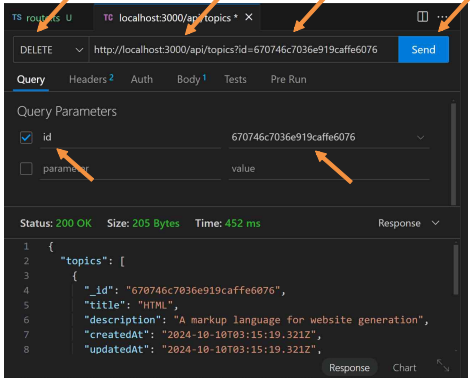
브라우저로 확인

/app/api/topics/route.ts

```
import connectMongoDB from '@/libs/mongodb'
import Topic from '@/models/topic'
import { NextResponse } from 'next/server'

중략

export async function DELETE(request: NextRequest) {
  try {
    const id = request.nextUrl.searchParams.get('id')
    if (!id) {
      return NextResponse.json({ message: 'ID is required' }, { status: 400 })
    }
    await connectMongoDB()
    const deletedTopic = await Topic.findByIdAndDelete(id)
    if (!deletedTopic) {
      return NextResponse.json({ message: 'Topic not found' }, { status: 404 })
    }
    return NextResponse.json({ message: 'Topic deleted' }, { status: 200 })
  } catch (error) {
    console.error('Error in DELETE /api/topics:', error)
    return NextResponse.json(
      { message: 'Internal Server Error' },
      { status: 500 }
    )
  }
}
```

DELETE 기능 구현

파라메터가 추가된
주소 자동 생성



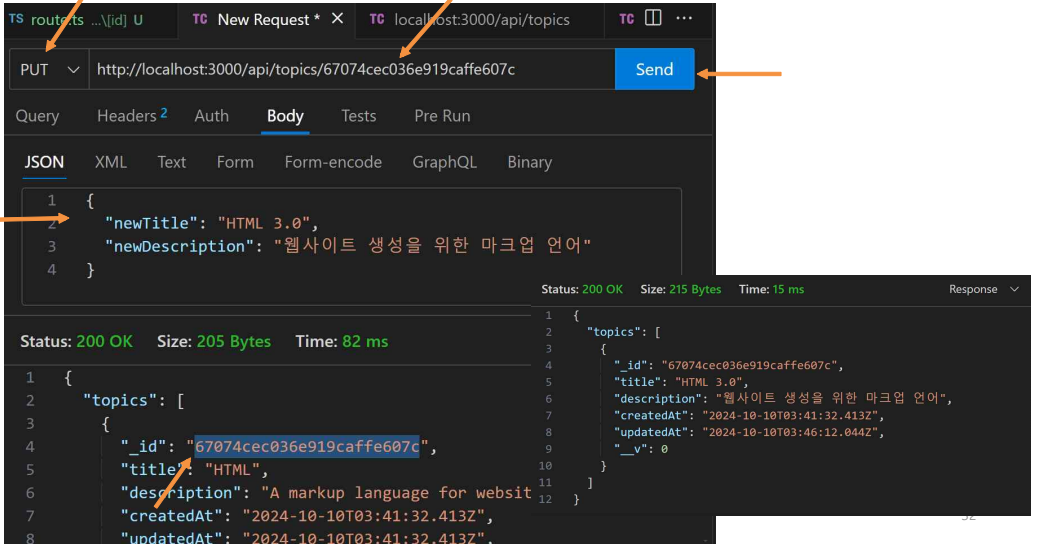
GET 실행하여 삭제 결과 확인

/app/api/topics/[id]/route.ts Dynamic route 활용

```
import connectMongoDB from '@/libs/mongodb'
import Topic from '@/models/topic'
import { NextRequest, NextResponse } from 'next/server'

export async function PUT(
  request: NextRequest,
  { params }: { params: { id: string } }
) {
  try {
    const { id } = params
    const { newTitle: title, newDescription: description } =
      await request.json()
    if (!title || !description) {
      return NextResponse.json(
        { message: 'Title and description are required' },
        { status: 400 }
      )
    }
    await connectMongoDB()
    const updatedTopic = await Topic.findByIdAndUpdate(
      id,
      { title, description },
      { new: true }
    )
    if (!updatedTopic) {
      return NextResponse.json({ message: 'Topic not found' }, { status: 404 })
    }
    return NextResponse.json(
      { message: 'Topic updated', topic: updatedTopic },
      { status: 200 }
    )
  } catch (error) {
    console.error('Error in PUT /api/topics/[id]:', error)
    return NextResponse.json(
      { message: 'Internal Server Error' },
      { status: 500 }
    )
  }
}
```

Update (PUT) 기능 구현

수정할 데이터의 id를 주소에 입력

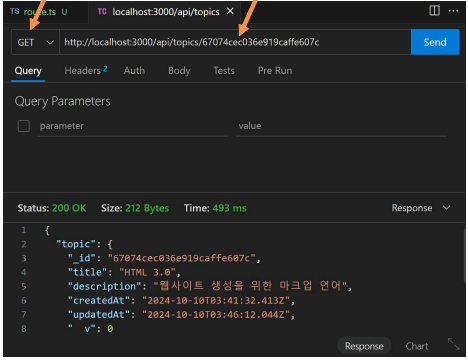


GET single data

/app/api/topics/[id]/route.ts

```
import connectMongoDB from '@/libs/mongodb'
import Topic from '@/models/topic'
import { NextRequest, NextResponse } from 'next/server'

...중략...

export async function GET(
  request: NextRequest,
  { params }: { params: { id: string } }
) {
  try {
    const { id } = params
    await connectMongoDB()
    const topic = await Topic.findOne({ _id: id })
    if (!topic) {
      return NextResponse.json({ message: 'Topic not found' }, { status: 404 })
    }
    return NextResponse.json({ topic }, { status: 200 })
  } catch (error) {
    console.error('Error in GET /api/topics/[id]:', error)
    return NextResponse.json(
      { message: 'Internal Server Error' },
      { status: 500 }
    )
  }
}
```

id를 지정하고 GET 요청



C-R-U-D

Create

```
export async function POST(request: NextRequest) {

 }
```

Read

```
export async function GET() {

}
```

Update

```
export async function PUT(
  request: NextRequest,
  { params }: { params: { id: string } }
) {

}
```

Delete

```
export async function DELETE(request: NextRequest) {

}
```

Frontend, Backend 연결

- 초기화면에 Topics 표시하기
- Add Topic 기능 구현
- Delete Topic 기능 구현
- Edit Topic 기능 구현

계속

초기화면에 Topics 표시

```
'use client'
import { useState, useEffect } from 'react'
import Link from 'next/link'
import { HiPencilAlt } from 'react-icons/hi'
import RemoveBtn from './RemoveBtn'
interface Topic {
  _id: string
  title: string
  description: string
  createdAt: string
  updatedAt: string
}

export default function TopicsList() {
  const [topics, setTopics] = useState<Topic[]>([])
  const [loading, setLoading] = useState(true)
  const [error, setError] = useState<string | null>(null)
  useEffect(() => {
    async function fetchTopics() {
      try {
        const res = await fetch('/api/topics')
        if (!res.ok) {
          throw new Error('Failed to fetch topics')
        }
        const data = await res.json()
        setTopics(data.topics)
      } catch (error) {
        console.error('Error loading topics: ', error)
        setError('Failed to load topics')
      } finally {
        setLoading(false)
      }
    }
    fetchTopics()
  }, [])

  if (loading) return <p>Loading topics...</p>
  if (error) return <p>Error: {error}</p>
  if (topics.length === 0) return <p>No topics found.</p>
```

/components/TolicsList.tsx

```
... 중략 ...

  return (
    <>
      {topics.map((topic: Topic) => (
        <div
          key={topic._id}
          className="p-4 border border-slate-300 my-3 flex justify-between gap-5 items-start"
        >
          <div>
            <h2 className="text-2xl font-bold">{topic.title}</h2>
            <div>{topic.description}</div>
            <div className="flex gap-4">
              <p>Created: {topic.createdAt} </p>
              <p>Updated: {topic.updatedAt} </p>
            </div>
          </div>
          <div className="flex gap-2">
            <RemoveBtn id={topic._id} />
            <Link href={`/editTopic/${topic._id}`}>
              <HiPencilAlt size={24} />
            </Link>
          </div>
        </div>
      ))}
    </>
  )
}
```

초기화면에 Topics 표시

Add Topic 기능 구현

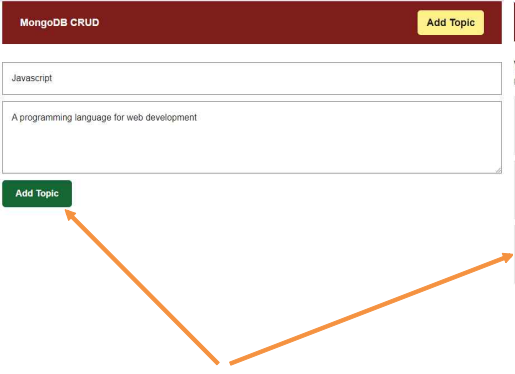
/app/addTopic/page.tsx

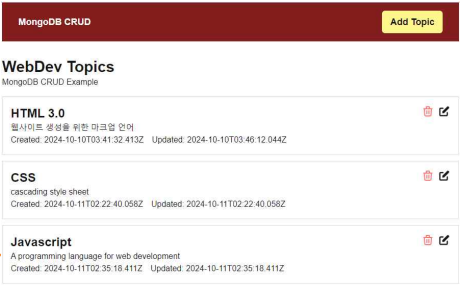
```
'use client'
import { useRouter } from 'next/navigation'
import { useState } from 'react'

export default function AddTopic() {
  const [title, setTitle] = useState('')
  const [description, setDescription] = useState('')
  const router = useRouter()

  const handleSubmit = async (e:
React.FormEvent<HTMLFormElement>) => {
    e.preventDefault()
    if (!title || !description) {
      alert('Title and description are required.')
    }
    try {
      const res = await fetch('/api/topics', {
        method: 'POST',
        headers: {
          'Content-type': 'application/json',
        },
        body: JSON.stringify({ title, description }),
      })
      if (res.ok) {
        router.push('/')
        router.refresh()
      } else {
        throw new Error('Failed to create a topic')
      }
    } catch (error) {
      console.log(error)
    }
  }
```

계속...

```
  return (
    <form className="flex flex-col gap-3" onSubmit={handleSubmit}>
      <input
        className="border border-slate-500 p-4"
        type="text"
        placeholder="Topic Title"
        onChange={(e: React.ChangeEvent<HTMLInputElement>) =>
          setTitle(e.target.value)
        }
        value={title}
      />
      <textarea
        className="border border-slate-500 p-4 h-32"
        placeholder="Topic Description"
        onChange={(e: React.ChangeEvent<HTMLTextAreaElement>) =>
          setDescription(e.target.value)
        }
        value={description}
      />
      <button
        className="bg-green-800 text-white font-bold px-6 py-3 w-fit rounded-md"
        type="submit"
      >
        Add Topic
      </button>
    </form>
  )
}
```

Add Topic 기능 테스트





Delete Topic 기능 구현

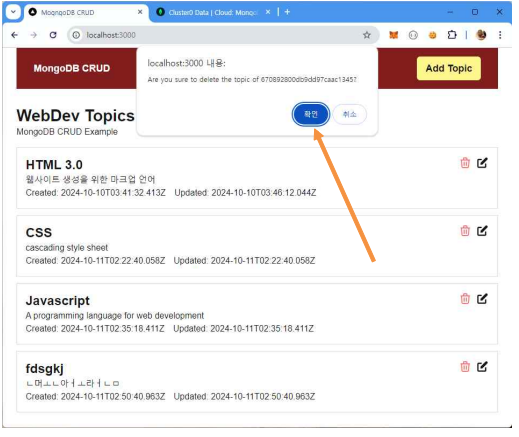
/components/TopicsList.tsx

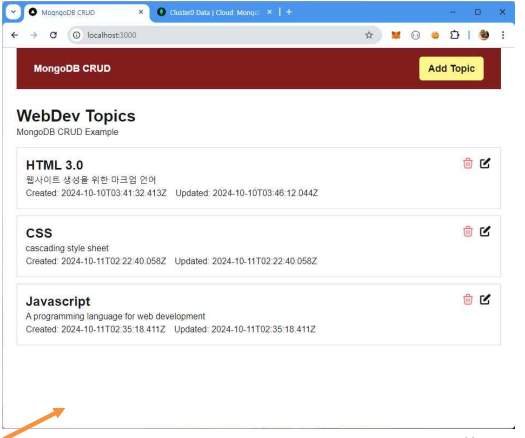
```
중략
        <div className="flex gap-2">
          <RemoveBtn id={topic._id} />
          <Link href={`/editTopic/${topic._id}`}>
            <HiPencilAlt size={24} />
          </Link>
        </div>
중략
```

/components/RemoveBtn.tsx

```
'use client'
import { useRouter } from 'next/navigation'
import { HiOutlineTrash } from 'react-icons/hi'

export default function RemoveBtn({ id }: { id: string }) {
  const router = useRouter()
  async function removeTopic() {
    const confirmed = confirm(`Are you sure to delete the topic of ${id}? `)
    if (confirmed) {
      const res = await fetch(`/api/topics?id=${id}`, {
        method: 'DELETE',
      })
      if (res.ok) {
        router.refresh()
      }
    }
  }
  return (
    <button className="text-red-400" onClick={removeTopic}>
      <HiOutlineTrash size={24} />
    </button>
  )
}
```

Delete Topic 기능 구현





Edit Topic 기능 구현

/app/editTopic/[id]/page.tsx

```
import EditTopicForm from '@/components/EditTopicForm'
const apiUrl = process.env.API_URL

const getTopicById = async (id: string) => {
  try {
    const res = await fetch(`${apiUrl}/api/topics/${id}`, {
      cache: 'no-store',
    })
    if (!res.ok) {
      throw new Error('Failed to fetch topic.')
    }
    return res.json()
  } catch (error) {
    console.log(error)
  }
}

export default async function EditTopic({
  params,
}: {
  params: { id: string }
}) {
  const { id } = params
  const { topic } = await getTopicById(id)
  const { title, description } = topic
  return <EditTopicForm id={id} title={title} description={description} />
}
```

현재 id의 topic을 읽어옴

/app/editTopic/[id]/page.tsx

MONGODB_URI=
API_URL=http://localhost:3000

개발중 서버주소
Deploy후 주소 변경 필요

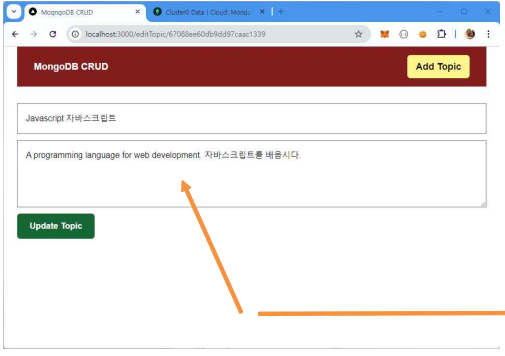
id, title, description을
EditTopicForm 컴포넌트에
파라메터로 전달

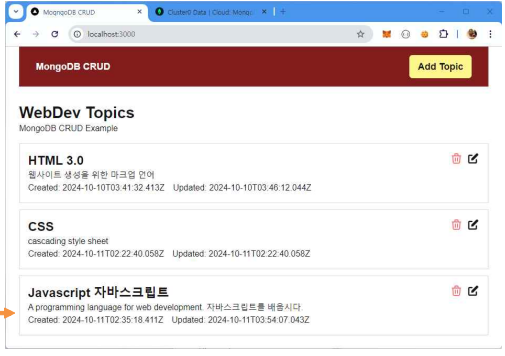
/components/EditTopicForm.tsx

```
'use client'
import { useRouter } from 'next/navigation'
import { useState } from 'react'
interface EditTopicFormProps {
  id: string
  title: string
  description: string
}
export default function EditTopicForm({
  id,
  title,
  description,
}: EditTopicFormProps) {
  const [newTitle, setNewTitle] = useState(title)
  const [newDescription, setNewDescription] = useState(description)
  const router = useRouter()
  const handleSubmit = async (e: React.FormEvent<HTMLFormElement>) => {
    e.preventDefault()
    try {
      const res = await fetch(`/api/topics/${id}`, {
        method: 'PUT',
        headers: {
          'Content-type': 'application/json',
        },
        body: JSON.stringify({ newTitle, newDescription }),
      })
      if (!res.ok) {
        throw new Error('Failed to update topic')
      }
      router.push('/')
      router.refresh()
    } catch (error) {
      console.log(error)
    }
  }
```

```
... 계속
  return (
    <form className="flex flex-col gap-3" onSubmit={handleSubmit}>
      <input
        className="border border-slate-500 p-4"
        type="text"
        placeholder="Topic Title"
        onChange={(e: React.ChangeEvent<HTMLInputElement>) =>
          setNewTitle(e.target.value)
        }
        value={newTitle}
      />
      <textarea
        className="border border-slate-500 p-4 h-32"
        placeholder="Topic Description"
        onChange={(e: React.ChangeEvent<HTMLTextAreaElement>) =>
          setNewDescription(e.target.value)
        }
        value={newDescription}
      />
      <button
        className="bg-green-800 text-white font-bold px-6 py-3 w-fit rounded-md"
        type="submit"
      >
        Update Topic
      </button>
    </form>
  )
}
```

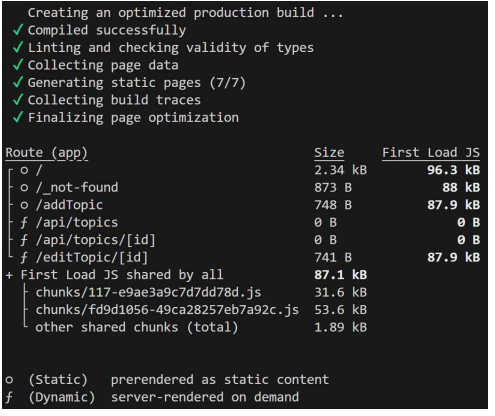
Edit Topic 기능 테스트





Deploy

- 빌드 테스트
  - 빌드시 에러 발생하면 그에 따라 수정
  - Typescript 사용으로 에러를 조기 확인하고 수정하기 쉬움
  - > npm run build



Deploy

- 서버주소 확인
  - 개발중 서버주소: http://localhost:3000
  - 서비스중 서버주소: https://*****.vercel.app 추후 결정.

- .env.local 파일에 설정된 API_URL 확인
  - Deploy후 주소가 결정되면 추후 변경 예정

```
API_URL=http://localhost:3000
```

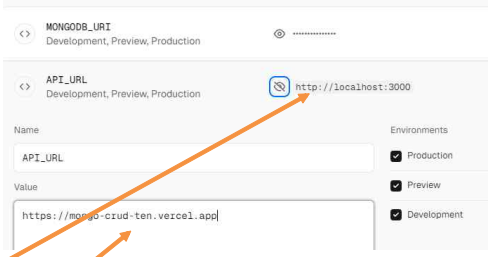
Deploy

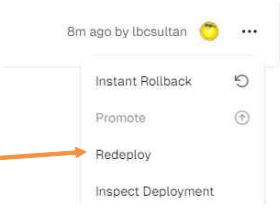
- 클라이언트측 페이지의 서버 주소
  - Client side: /api/*** (상대주소 사용 가능, 이미 적용됨)
  - /components/TopicsList.tsx (client side)
  - /app/addTopic/page.tsx (client side)
  - /components/EditTopicForm.tsx (client side)
  - /components/RemoveBtn.tsx (client side)

- 서버측 페이지의 서버 주소
  - Server side: ${apiUrl}/api/*** (절대주소 사용 필요)
  - /app/editTopic/[id]/page.tsx (server side)
  - Deploy후 주소가 결정되면 추후 변경

Deploy

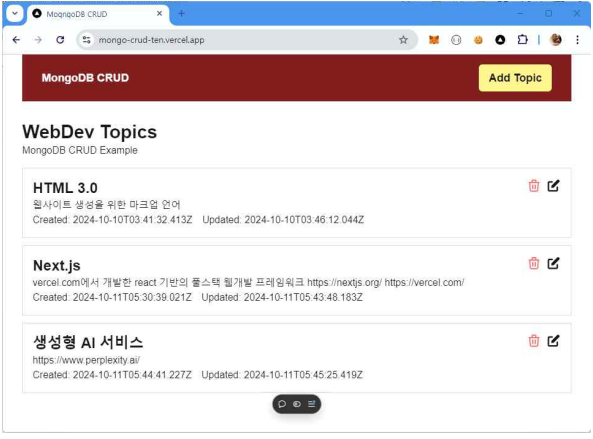
- Github.com에 소스 등록
  - https://github.com/lbcsultan/mongo-crud
- Vercel.com에 deploy
  - .env.local 의 내용을 환경변수로 입력
  - Deploy
  - https://mongo-crud-ten.vercel.app/
- EditTopic 부분의 주소 수정 후 redeploy
  - http://localhost:3000 을
  - https://mongo-crud-ten.vercel.app 로 변경
  - Redeploy





기능 테스트

완성


MongoDB CRUD
mongo-crud-ten.vercel.app
MongoDB CRUD
Add Topic
WebDev Topics
MongoDB CRUD Example
HTML 3.0
웹사이트 생성을 위한 마크업 언어
Created: 2024-10-10T03:41:32.413Z Updated: 2024-10-10T03:46:12.044Z
Next.js
vercel.com에서 개발한 react 기반의 풀스택 웹개발 프레임워크 https://nextjs.org/ https://vercel.com/
Created: 2024-10-11T05:30:39.021Z Updated: 2024-10-11T05:43:48.183Z
생성형 AI 서비스
https://www.perplexity.ai/
Created: 2024-10-11T05:44:41.227Z Updated: 2024-10-11T05:45:25.419Z

Server Actions

- Server actions란?
  - 서버에서 실행되는 비동기 함수
  - 서버컴포넌트, 클라이언트컴포넌트에서 call할 수 있음
  - API route를 만들지 않고도 서버에서 동작하는 비동기 함수를 생성 가능
  - https://nextjs.org/docs/app/building-your-application/data-fetching/server-actions-and-mutations

Server actions vs. API routes

서버사이드 기능 구현의 두가지 방식

특징	Server Actions	API Routes
타입 안정성	기본적으로 제공	추가 설정 필요
사용 위치	컴포넌트 내부 또는 별도 파일	별도의 API 폴더
엔드포인트	명시적 URL 없음	명확한 URL 경로
재사용성	Next.js 앱 내부에 제한	외부 앱에서도 사용 가능
웹훅 지원	제한적	우수
캐싱	자동으로 캐시되지 않음	설정에 따라 가능
HTTP 메소드	POST만 사용	모든 HTTP 메소드 지원

Server actions vs. API routes

- Server Actions 적합한 경우:
  - 간단한 데이터 변경 작업
  - 폼 제출 처리
  - 사용자 인증 및 권한 부여
  - Next.js 앱 내부에서만 사용되는 로직
- API Routes 적합한 경우:
  - 복잡한 데이터 처리 및 조작
  - 외부 API나 서비스와의 통합
  - 공개 API 제공
  - 웹훅 구현
  - React Native 앱 등 외부 클라이언트 지원

기존 구현과 별도로 구현

- 기존 프로젝트를 복사하여 새 프로젝트로 저장
- Server action을 이용한 서버 기능 구현
  - /src/actions/topicActions.ts
  - 위 테스트 완료 후 /app/api/topics 폴더 삭제
- 프런트엔드 페이지를 적절하게 수정
- 별도의 프로젝트로 deploy

Server action을 이용한 CRUD 구현

/src/actions/topicActions.ts

```
'use server'

import connectMongoDB from '@/libs/mongodb'
import Topic from '@/models/topic'
import { revalidatePath } from 'next/cache'
import { convertDocToObj } from '@/libs/helpers'
// 1. 토픽 생성: Create (POST)
export async function createTopic(title: string, description: string) {
  try {
    await connectMongoDB()
    const doc = await Topic.create({ title, description })
    revalidatePath('/')
    return { success: true, topic: convertDocToObj(doc) }
  } catch (error) {
    throw new Error(`토픽 생성에 실패했습니다: ${error}`)
  }
}
// 2. 토픽 수정: Update (PUT)
export async function updateTopic(
  id: string,
  title: string,
  description: string
) {
  try {
    await connectMongoDB()
    const doc = await Topic.findByIdAndUpdate(
      id,
      { title, description },
      { new: true }
    )
    if (!doc) throw new Error('토픽을 찾을 수 없습니다')
    revalidatePath('/')
    return { success: true, topic: convertDocToObj(doc) }
  } catch (error) {
    throw new Error(`토픽 수정에 실패했습니다: ${error}`)
  }
}
// 3. 단일 토픽 조회 (GET)
export async function getTopic(id: string) {
  try {
    await connectMongoDB()
    const doc = await Topic.findById(id)
    if (!doc) throw new Error('토픽을 찾을 수 없습니다')
    return { success: true, topic: convertDocToObj(doc) }
  } catch (error) {
    throw new Error(`토픽 조회에 실패했습니다: ${error}`)
  }
}
// 4. 모든 토픽 조회 (GET)
export async function getAllTopics() {
  try {
    await connectMongoDB()
    const docs = await Topic.find({}).sort({ createdAt: -1 })
    const topics = docs.map((doc) => convertDocToObj(doc))
    return { success: true, topics }
  } catch (error) {
    throw new Error(`토픽 목록 조회에 실패했습니다: ${error}`)
  }
}
// 5. 토픽 삭제: DELETE
export async function deleteTopic(id: string) {
  try {
    await connectMongoDB()
    const doc = await Topic.findByIdAndDelete(id)
    if (!doc) throw new Error('토픽을 찾을 수 없습니다')
    revalidatePath('/')
    return { success: true }
  } catch (error) {
    throw new Error(`토픽 삭제에 실패했습니다: ${error}`)
  }
}
```

메인페이지
- 모든 토픽 보여주기

/app/page.tsx

```tsx
import { Metadata } from 'next'
import TopicsList from '@/components/TopicsList'
import { Suspense } from 'react'
import { getAllTopics } from '@/actions/topicActions'

export const metadata: Metadata = {
  title: 'WebDev Topics | MongoDB CRUD Example',
  description: 'A simple CRUD application using Next.js and MongoDB',
}

export default async function Home() {
  const { topics } = await getAllTopics()

  return (
    <main className="container mx-auto p-4">
      <h1 className="text-3xl font-bold">WebDev Topics</h1>
      <p className="mb-4">MongoDB CRUD Example</p>
      <Suspense fallback={<div>로딩 중...</div>}>
        <TopicsList topics={topics} />
      </Suspense>
    </main>
  )
}
```

/components/TopicList.tsx

```tsx
'use client'
import Link from 'next/link'
import { HiPencilAlt } from 'react-icons/hi'
import RemoveBtn from './RemoveBtn'
import { Topic } from '@/types/topic'
interface TopicsListProps {
  topics: Topic[]
}
export default function TopicsList({ topics }: TopicsListProps) {
  return (
    <>
      {topics.map((topic) => (
        <div
          key={topic._id}
          className="p-4 border border-slate-300 my-3 flex justify-between gap-5 items-start"
        >
          <div>
            <h2 className="text-2xl font-bold">{topic.title}</h2>
            <div>{topic.description}</div>
            <div className="flex gap-4">
              <p>Created: {new Date(topic.createdAt).toLocaleDateString()}</p>
              <p>Updated: {new Date(topic.updatedAt).toLocaleDateString()}</p>
            </div>
          </div>
          <div className="flex gap-2">
            <RemoveBtn id={topic._id} />
            <Link href={`/editTopic/${topic._id}`}>
              <HiPencilAlt size={24} />
            </Link>
          </div>
        </div>
      ))}
    </>
  )
}
```

토픽 생성

/app/addTopic/page.tsx

```
import AddTopicForm from '@/components/AddTopicForm'

export default function AddTopic() {
  return (
    <div className="max-w-3xl mx-auto mt-8">
      <h1 className="text-2xl font-bold mb-4">새 토픽 추가</h1>
      <AddTopicForm />
    </div>
  )
}
```

/components/AddTopicForm.tsx

```
'use client'

import { createTopic } from '@/actions/topicActions'
import { useState } from 'react'
import { useRouter } from 'next/navigation'

export default function AddTopicForm() {
  const [title, setTitle] = useState('')
  const [description, setDescription] = useState('')
  const router = useRouter()

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault()

    try {
      await createTopic(title, description)
      router.push('/') // 생성 후 메인 페이지로 이동
    } catch (error) {
      console.error('토픽 생성 중 오류:', error)
      alert('토픽 생성 중 오류가 발생했습니다.')
    }
  }

  return (
    <form onSubmit={handleSubmit} className="flex flex-col gap-3">

                중략 (동일)

    </form>
  )
}
```

토픽 수정

/app/editTopic/[id]/page.tsx

```tsx
import { getTopic } from '@/actions/topicActions'
import EditTopicForm from '@/components/EditTopicForm'

export default async function EditTopic({
  params,
}: {
  params: { id: string }
}) {
  const { topic } = await getTopic(params.id)

  return (
    <div className="max-w-3xl mx-auto mt-8">
      <h1 className="text-2xl font-bold mb-4">토픽 수정</h1>
      <EditTopicForm
        id={topic._id}
        initialTitle={topic.title}
        initialDescription={topic.description}
      />
    </div>
  )
}
```

/components/EditTopicForm.tsx

```tsx
'use client'

import { updateTopic } from '@/actions/topicActions'
import { useState } from 'react'
import { useRouter } from 'next/navigation'

interface EditTopicFormProps {
  id: string
  initialTitle: string
  initialDescription: string
}

export default function EditTopicForm({
  id,
  initialTitle,
  initialDescription,
}: EditTopicFormProps) {
  const [title, setTitle] = useState(initialTitle)
  const [description, setDescription] = useState(initialDescription)
  const router = useRouter()

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault()

    try {
      await updateTopic(id, title, description)
      router.push('/') // 수정 후 메인 페이지로 이동
    } catch (error) {
      console.error('토픽 수정 중 오류:', error)
      alert('토픽 수정 중 오류가 발생했습니다.')
    }
  }

  return (
    <form onSubmit={handleSubmit} className="flex flex-col gap-3">
      중략 (동일)
    </form>
  )
```

토픽 삭제

/components/RemoveBtn.tsx

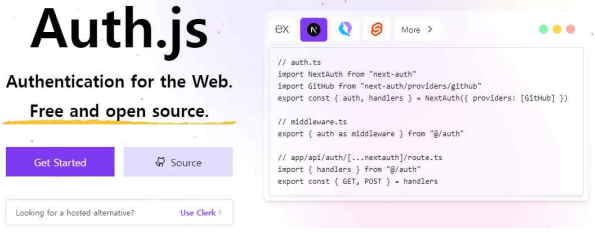
```
'use client'

import { deleteTopic } from '@/actions/topicActions'
import { HiOutlineTrash } from 'react-icons/hi'

export default function RemoveBtn({ id }: { id: string }) {
  const handleDelete = async () => {
    const confirmed = confirm('이 토픽을 삭제하시겠습니까?')

    if (confirmed) {
      try {
        await deleteTopic(id)
      } catch (error) {
        console.error('삭제 중 오류 발생:', error)
        alert('삭제 중 오류가 발생했습니다.')
      }
    }
  }

  return (
    <button onClick={handleDelete} className="text-red-400">
      <HiOutlineTrash size={24} />
    </button>
  )
}
```

Deploy

- Github
  - https://github.com/lbcsultan/mongo-crud-action
- Vercel
  - https://mongo-crud-action.vercel.app/

2. NextAuth 인증

- Auth.js
  - https://authjs.dev/
  - 다양한 인증 provider, database adapter 제공
  - 개발자가 다양하게 customize 가능한 확장성
  - 계정정보를 직접 관리 가능
  - Google, github 인증 추가 예정



시작하기

- Getting started
  - https://authjs.dev/getting-started
  - Next.js 선택

- 1. 패키지 설치
  - > npm install next-auth@beta

- 2. 환경변수 AUTH_SECRET 생성
  - > npx auth secret
  - .env.local 파일에 AUTH_SECRET 변수가 자동 추가됨

시작하기

- 3. 환경설정
  - ./auth.ts 파일 생성
  - ./app/api/auth/[...nextauth]/route.ts 파일 생성
  - ./middleware.ts 파일 생성

./auth.ts

```
import NextAuth from 'next-auth'

export const { handlers, signIn, signOut, auth } = NextAuth({
  providers: [],
})
```

./app/api/auth/[...nextauth]/route.ts

```
import { handlers } from '@/auth' // Referring to the auth.ts we just created
export const { GET, POST } = handlers
```

./middleware.ts

```
export { auth as middleware } from '@/auth'
```

시작하기

- 4. 인증 방법 설정
  - Oauth: 80가지 이상의 소셜로그인 방식 제공
  - Magic links: 이메일로 검증코드를 제공하여 입력하는 로그인
  - Credentials: id/password 방식의 로그인
  - WebAuthn (Passkeys): 새로 개발되고 있는 웹인증 방식
  - Google, github 로그인 방식 사용 예정

Google 인증, Github 인증 추가

- ./auth.ts의 providers 배열에 Google, Github 추가

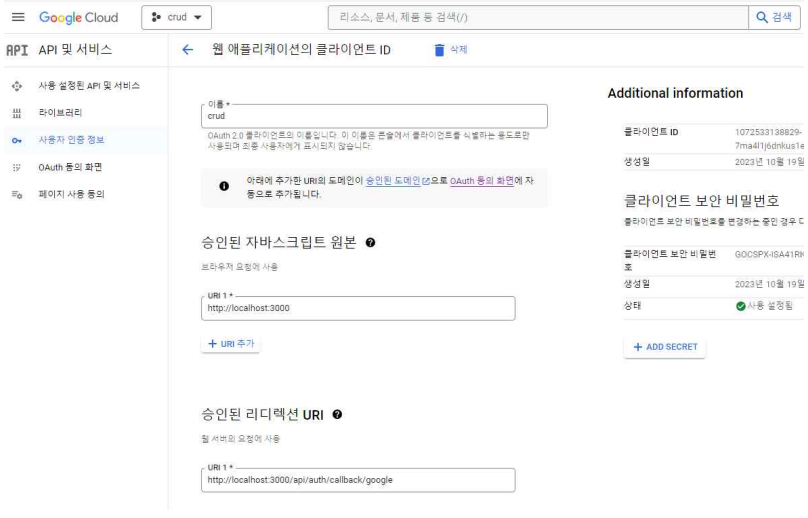
```
import NextAuth from 'next-auth'
import GitHub from 'next-auth/providers/github'
import Google from 'next-auth/providers/google'

export const { handlers, signIn, signOut, auth } = NextAuth({
  providers: [Google, GitHub],
})
```

Google Provider 앱 등록

- **Documentation**
  - **https://developers.google.com/identity/protocols/oauth2?hl=ko**
- **Configuration**
  - **https://console.developers.google.com/apis/credentials**

OAuth 동의 화면 만들기
프로젝트 생성/선택
사용자 인증정보 만들기
OAuth 클라이언트 ID 만들기
애플리케이션 유형 → 웹 애플리케이션
승인된 자바스크립트 원본 – http://localhost:3000
승인된 리디렉션 URI – http://localhost:3000/api/auth/callback/google

생성된 클라이언트 ID, 클라이언트 보안 비밀번호를 .env.local에 등록

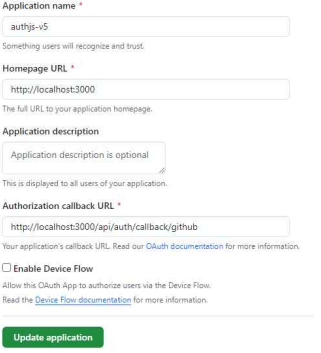

Google Cloud
crud
리소스, 문서, 제품 등 검색(/)
검색
API 및 서비스
웹 애플리케이션의 클라이언트 ID
삭제
사용 설정된 API 및 서비스
라이브러리
사용자 인증 정보
OAuth 동의 화면
페이지 사용 동의
이름 *
crud
OAuth 2.0 클라이언트의 이름입니다. 이 이름은 콘솔에서 클라이언트를 식별하는 용도로만 사용되며 최종 사용자에게 표시되지 않습니다.
아래에 추가한 URI와 도메인이 승인된 도메인으로 OAuth 동의 화면에 자동으로 추가됩니다.
승인된 자바스크립트 원본
브라우저 요청에 사용
URI 1 *
http://localhost:3000
+ URI 추가
승인된 리디렉션 URI
웹 서버의 요청에 사용
URI 1 *
http://localhost:3000/api/auth/callback/google
Additional information
클라이언트 ID
1072533138829-7ma4l1j6dnkus1e
생성일
2023년 10월 19일
클라이언트 보안 비밀번호
클라이언트 보안 비밀번호를 변경하는 중인 경우 다
클라이언트 보안 비밀번호
GOCSPX-ISA41RK
생성일
2023년 10월 19일
상태
사용 설정됨
+ ADD SECRET

Github Provider 앱 등록

- **Documentation**
  - **https://developer.github.com/apps/building-oauth-apps/authorizing-oauth-apps**
- **Configuration**
  - **https://github.com/settings/apps**

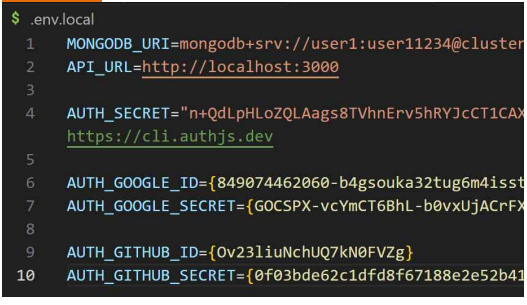
https://github.com/settings/developers
New OAuth App
Application Name
Homepage URL
http://localhost:3000
Authorization callback URL
http://localhost:3000/api/auth/callback/github

생성된 Client ID, Client secrets을 .env에 등록



환경변수 등록 .env.local

.env.local

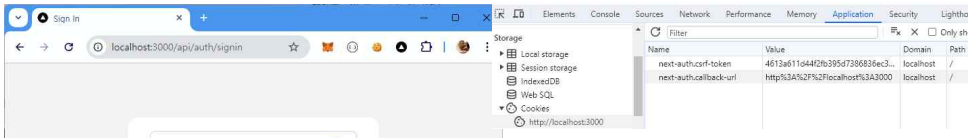


환경변수 변경 후에는 개발서버 재시작 필요
> npm run dev

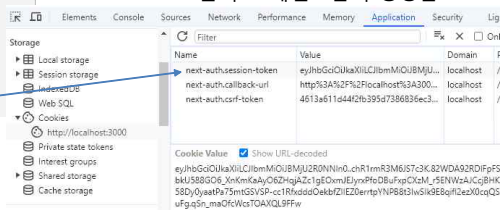
Next-auth 기본 로그인 페이지

- http://localhost:3000/api/auth/signin

로그인 전



로그인 후 – 세션토큰이 생성됨



세션 활용하기

- SessionProvider 설정
  - 전체 어플리케이션에서 로그인 세션정보를 활용할 수 있도록 환경 설정
  - 컴포넌트 생성 : /components/Providers.tsx
  - 레이아웃에 적용 : /app/layout.tsx

/components/Providers.tsx

```
import { SessionProvider } from 'next-auth/react'

export const NextAuthProvider = ({
  children,
}: {
  children: React.ReactNode
}) => {
  return <SessionProvider>{children}</SessionProvider>
}
```

/app/layout.tsx

```
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body className={inter.className}>
        <NextAuthProvider>
          <div className="max-w-3xl mx-auto p-4">
            <Navbar />
            <div className="mt-8">{children}</div>
          </div>
        </NextAuthProvider>
      </body>
    </html>
  )
}
```

메뉴에 세션 적용

- 로그인된 경우
  - 세션정보가 존재
  - 사용자 정보 표시, Sign Out 버튼 표시
- 로그인 안된 경우
  - 세션정보가 없음
  - Sign In 버튼 표시
- 세션정보 확인하기
  - Client 측: useSession() 사용
  - Server 측: auth() 사용

메뉴에 세션 적용

/components/Navbar.tsx

```
'use client'
import { signOut, useSession } from 'next-auth/react'
import Image from 'next/image'
import Link from 'next/link'

export default function Navbar() {
  const { status, data: session } = useSession()

  return (
    <nav className="flex justify-between items-center bg-red-900 px-8 py-4">
      <Link className="text-white text-lg font-bold " href="/">
        MongoDB CRUD
      </Link>
      <Link
        className="bg-yellow-200 text-lg font-bold px-4 py-2 rounded-md"
        href="/addTopic"
      >
        Add Topic
      </Link>
      <div className="flex gap-4">

        조건부 메뉴 표시 <----------------------------

      </div>
    </nav>
  )
}
```

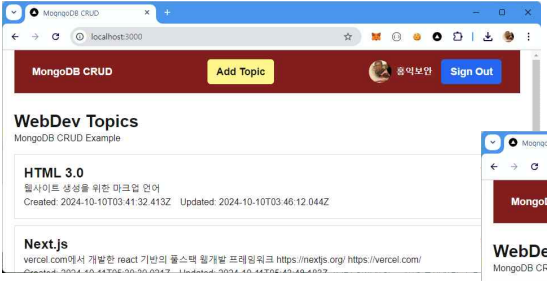
로그인 상태

로그인 안된 상태

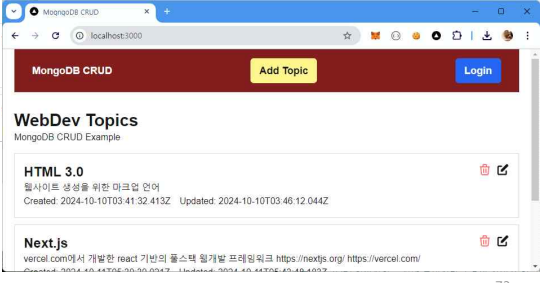
```
{status === 'authenticated' ? (
          <>
            <div className="flex gap-2 items-center">
              <Image
                alt={session?.user?.name ?? 'user'}
                src={session?.user?.image ?? '/default-avatar.png'}
                width={40}
                height={40}
                alt={session?.user?.name ?? 'user'}
              />
              <span className="text-white font-bold">
                {session?.user?.name}
              </span>
            </div>
            <button
              onClick={() => signOut()}
              className="bg-blue-600 hover:bg-blue-700 text-white px-4 py-2
rounded-md text-lg font-bold"
            >
              Sign Out
            </button>
          </>
        ) : (
          <Link
            href="/login"
            className="bg-blue-600 hover:bg-blue-700 text-white px-4 py-2
rounded-md text-lg font-bold"
          >
            Login
          </Link>
        )}
```

메뉴에 세션 적용

로그인 상태



로그인 안된 상태



Sign In 페이지 만들기

- 직접 만드는 로그인 페이지
  - Nextauth의 기본 로그인 페이지를 사용하지 않고 직접 생성하는 로그인 페이지 이용

/auth.ts

```
import NextAuth from 'next-auth'
import GitHub from 'next-auth/providers/github'
import Google from 'next-auth/providers/google'

export const { handlers, signIn, signOut, auth } = NextAuth({
  providers: [Google, GitHub],
  pages: {
    signIn: '/login',
  },
})
```

/login 위치에 만드는 페이지를
signIn으로 사용하겠다고 설정
http://localhost:3000/api/auth/signin 를 요청하면
/login 페이지로 리다이렉트함

이것이 설정되지 않으면 기본 로그인 페이지를 사용
http://localhost:3000/api/auth/signin

Login 페이지 만들기

/components/SignInButton.tsx

```
import { signIn } from '@/auth'
import Image from 'next/image'

export default function SignInButton() {
  return (
    <div className="flex flex-col gap-4 mt-10 items-center">
      <form
        action={async () => {
          'use server'
          await signIn('google', { redirectTo: '/dashboard' })
        }}
      >
        <button
          type="submit"
          className="flex items-center justify-center gap-4 rounded-lg pl-3 mb-4"
        >
          <Image src="/google-logo.png" height={30} width={30} alt="google" />
          <span className="bg-blue-500 text-white px-4 py-3">
            Sign in with Google
          </span>
        </button>
      </form>
      <form
        action={async () => {
          'use server'
          await signIn('github', { redirectTo: '/dashboard' })
        }}
      >
        <button
          type="submit"
          className="flex items-center justify-center gap-4 rounded-lg pl-3 mb-4"
        >
          <Image src="/github-logo.png" height={30} width={30} alt="github" />
          <span className="bg-blue-500 text-white px-4 py-3">
            Sign in with Github
          </span>
        </button>
      </form>
    </div>
  )
}
```

Server action

Server action

/app/login/page.tsx

```
import SignInButton from '@/components/SignInButton'
import React from 'react'

export default function LoginPage() {
  return (
    <div className="mt-16 max-w-xl mx-auto text-center">
      <SignInButton />
    </div>
  )
}
```

이미지 준비하기

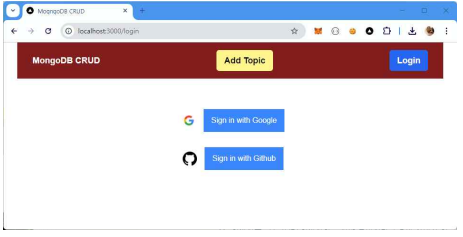
- public 폴더 생성
  - 프로젝트 루트에 public 폴더 생성
  - 프로젝트 전체에서 공유하는 정적 파일들을 관리
- public 폴더에 이미지 저장
  - google-logo.png
  - github-logo.png
  - default-avatar.png
- Next.config.mjs 파일 설정
  - Images에 remotePatterns 에 이미지를 제공하는 외부 서버 주소를 등록해야 함.
  - 등록되지 않은 서버에서 제공하는 이미지를 렌더링 시도하는 경우 에러 발생

/next.config.mjs

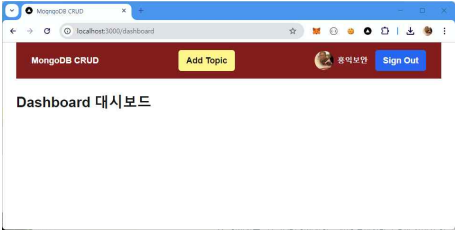
```
/** @type {import('next').NextConfig} */
const nextConfig = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: 'lh3.googleusercontent.com',
      },
      {
        protocol: 'https',
        hostname: 'avatars.githubusercontent.com',
      },
    ],
  },
}
```

Login 페이지 만들기

직접 만든 /login 페이지 표시



로그인되면 /dashboard로 리다이렉트



루트 보호하기 (route protection)

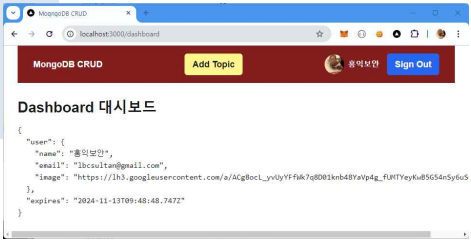
- 특정 페이지에는 로그인된 상태에서만 접근할 수 있도록 설정
  - 로그인 여부는 세션정보 존재 여부로 확인
  - 보호하고자 하는 페이지: 대시보드 페이지, addTopic, editTopic 페이지
  - 세션 정보가 있으면 요청한 페이지 표시, 없으면 로그인 페이지로 리다이렉트

- 세션 정보 읽어오기 방법
  - 클라이언트 세션 : useSession() 사용
  - 서버 세션 : auth() 사용

루트 보호하기 – 대시보드 페이지

/app/dashboard/page.tsx

```
import { auth } from '@/auth'

export default async function DashboardPage() {
  const session = await auth()
  if (!session) return <div className="text-2xl">Not authenticated...</div>

  return (
    <div>
      <h1 className="text-3xl font-bold">Dashboard 대시보드</h1>
      <pre className="mt-4">{JSON.stringify(session, null, 2)}</pre>
    </div>
  )
}
```



세션으로부터 사용자 정보 표시

루트 보호하기 – addTopic 페이지

/app/addTopic/page.tsx

```
'use client'
import { useSession } from 'next-auth/react'
import { redirect, useRouter } from 'next/navigation'

중략

  const { status, data: session } = useSession()

  if (!session) {
    redirect('/login')
  }

중략
```

클라이언트 페이지에서는 useSession() 훅을 사용하여 브라우저의 쿠키에 저장된 세션정보 가져오기

루트 보호하기 – editTopic 페이지

/app/editTopic/[id]/page.tsx

```
import { auth } from '@/auth'
import { redirect } from 'next/navigation'

중략

  const session = await auth()
  if (!session) {
    redirect('/login')
  }

중략
```

서버 페이지에서는 auth() 함수를 이용하여
서버에 요청하여 세션정보를 받아옴

사용자 정보를 DB로 관리하기

- 현재까지는 DB를 사용하지 않고 로그인 처리
- MongoDB, mongoose를 사용하여 사용자등록, 로그인정보, 세션정보 등을 관리할 예정임

User 모델 생성

/models/user.ts

```
import mongoose, { Model, Schema } from 'mongoose'

interface IUser {
  name: string
  email: string
}

const userSchema = new Schema<IUser>({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
})

let User: Model<IUser>

try {
  User = mongoose.model<IUser>('User')
} catch {
  User = mongoose.model<IUser>('User', userSchema)
}

export default User
```

새로운 사용자를 DB에 저장

- Callback 함수 설정
  - signIn 함수가 실행되면 그 이후 추가적으로 수행되는 함수
  - 사용자 정보를 DB에 기록
  - 등록되지 않은 email 주소에 대해서만 기록

백엔드 루트를 통해 DB에 저장

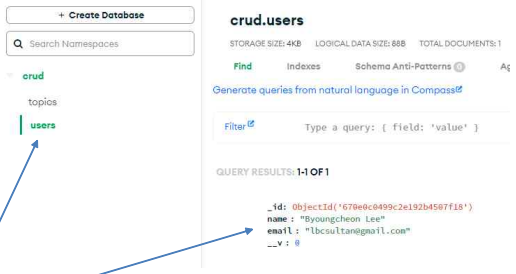
/auth.ts

```
import NextAuth from 'next-auth'
import GitHub from 'next-auth/providers/github'
import Google from 'next-auth/providers/google'
import connectMongoDB from './libs/mongodb'
import User from './models/user'

export const { handlers, signIn, signOut, auth } = NextAuth({
  providers: [Google, GitHub],
  pages: {
    signIn: '/login',
  },
  callbacks: {
    async signIn({ user, account }) {
      const apiUrl = process.env.API_URL
      const { name, email } = user
      if (account?.provider === 'google' || account?.provider === 'github') {
        try {
          await connectMongoDB()
          const userExists = await User.findOne({ email })
          if (!userExists) {
            const res = await fetch(`${apiUrl}/api/user`, {
              method: 'POST',
              headers: {
                'Content-Type': 'application/json',
              },
              body: JSON.stringify({ name, email }),
            })
            if (res.ok) {
              return true
            }
          }
        } catch (error) {
          console.log(error)
          return false
        }
      }
      return true
    },
  },
})
```

/app/api/user/route.ts

백엔드 API 루트

```
import User from '@/models/user'
import connectMongoDB from '@/libs/mongodb'
import { NextRequest, NextResponse } from 'next/server'

export async function POST(request: NextRequest) {
  try {
    const { name, email } = await request.json()
    // 입력 유효성 검사
    if (!name || !email) {
      return NextResponse.json(
        { error: 'Name and email are required' },
        { status: 400 }
      )
    }
    // 이메일 형식 검사
    if (!/^₩S+@₩S+₩.₩S+$/.test(email)) {
      return NextResponse.json(
        { error: 'Invalid email format' },
        { status: 400 }
      )
    }
    await connectMongoDB()
    const user = await User.create({ name, email })

    return NextResponse.json(
      { message: 'User registered', user },
      { status: 201 }
    )
  } catch (error) {
    console.error('Error registering user:', error)
    return NextResponse.json(
      { error: 'Failed to register user' },
      { status: 500 }
    )
  }
}
```



{name, email}을 User 컬렉션에 저장

로그인 이벤트 로그 남기기

- Log 모델 생성

/models/log.ts

```
import mongoose, { Schema } from 'mongoose'
const logSchema = new Schema(
  {
    email: { type: String, required: true },
  },
  {
    timestamps: true,
  }
)
const Log = mongoose.models.Log || mongoose.model('Log', logSchema)
export default Log
```

로그인 이벤트 DB에 저장

- Callback 함수 설정
  - signIn 함수가 실행되면 모든 로그인 이벤트를 DB에 저장

새로운 사용자를 DB에 저장

모든 로그인 이벤트를 DB에 저장

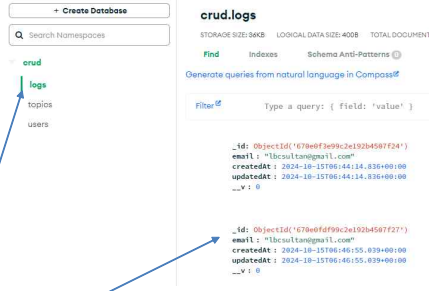
/auth.ts

```
try {
  await connectMongoDB()
  const userExists = await User.findOne({ email })
  if (!userExists) {
    const res = await fetch(`${apiUrl}/api/user`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ name, email }),
    })
    if (res.ok) {
      return true
    }
  }
  const res1 = await fetch(`${apiUrl}/api/log`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({ email }),
  })
  if (res1.ok) {
    return true
  }
 } catch (error) {
  console.log(error)
  return false
 }
```

백엔드 API 루트

/app/api/log/route.ts

```
import connectMongoDB from '@/libs/mongodb'
import Log from '@/models/log'
import { NextRequest, NextResponse } from 'next/server'

export async function POST(request: NextRequest) {
  try {
    const { email } = await request.json()
    await connectMongoDB()
    await Log.create({ email })
    return NextResponse.json({ message: 'Login event logged' },
{ status: 201 })
  } catch (error) {
    console.error('Error logging login event:', error)
  }
}
```



{email}을 Log 컬렉션에 저장

3. Deploy

- Build test
  - > npm run build

- 여러가지 에러들을 수정해야 함
  - Auth.ts 파일에서 MongoDB 연결을 직접 수행할 수 없음 => mongoDB 연결을 별도 API 루트로 분리하여 사용
  - 각종 파일에서 선언만 하고 사용하지는 않음 => 선언을 지움
  - Perplexity AI의 도움을 받음

- Build test 성공
  - > npm run build

/auth.ts

```
import NextAuth from 'next-auth'
import GitHub from 'next-auth/providers/github'
import Google from 'next-auth/providers/google'

export const { handlers, signIn, signOut, auth } = NextAuth({
  providers: [Google, GitHub],
  pages: {
    signIn: '/login',
  },
  callbacks: {
    async signIn({ user, account }) {
      if (account?.provider === 'google' || account?.provider === 'github') {
        try {
          const apiUrl = process.env.API_URL || ''

          const res = await fetch(`${apiUrl}/api/user-auth`, {
            method: 'POST',
            headers: {
              'Content-Type': 'application/json',
            },
            body: JSON.stringify({ user, account }),
          })
          if (res.ok) {
            return true
          }
          return false
        } catch (error) {
          console.log(error)
          return false
        }
      }
      return true
    },
  },
})
```

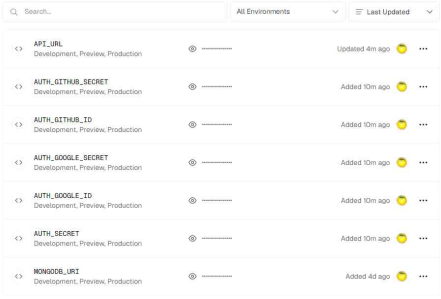
/app/api/user-auth/route.ts

```
import { NextRequest, NextResponse } from 'next/server'
import connectMongoDB from '@/libs/mongodb'
import User from '@/models/user'

export async function POST(req: NextRequest) {
  const { user } = await req.json()
  const { name, email } = user

  try {
    await connectMongoDB()
    const userExists = await User.findOne({ email })

    if (!userExists) {
      await User.create({ name, email })
    }

    // Log the login event
    const apiUrl = process.env.API_URL
    await fetch(`${apiUrl}/api/log`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ email }),
    })
    return NextResponse.json({ success: true })
  } catch (error) {
    console.error(error)
    return NextResponse.json(
      { success: false, error: 'Internal Server Error' },
      { status: 500 }
    )
  }
}
```

mongoDB 연결을 별도 API 루트로 분리하여 사용

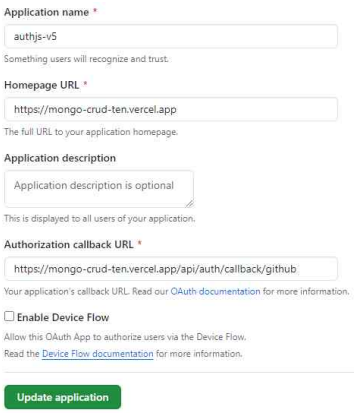
Redeploy

- Github에 Commit & Push
  - https://github.com/lbcsultan/mongo-crud
- Vercel에 Deploy
  - .env.local 파일의 변경된 내용을 환경변수에 설정
  - 동일한 주소로 Redeploy
  - https://mongo-crud-ten.vercel.app/



Google, github에 등록된 인증 URL을 변경

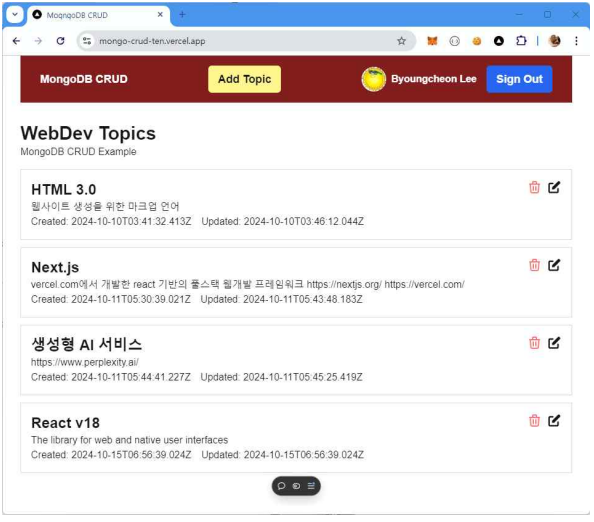
Github URL 변경



Google URL 변경

이렇게 바꾸면 http://localhost:3000 에서는 인증에 사용하지 못함

성공


MongoDB CRUD
mongo-crud-ten.vercel.app
MongoDB CRUD
Add Topic
Byoungcheon Lee
Sign Out
WebDev Topics
MongoDB CRUD Example
HTML 3.0
웹사이트 생성을 위한 마크업 언어
Created: 2024-10-10T03:41:32.413Z Updated: 2024-10-10T03:46:12.044Z
Next.js
vercel.com에서 개발한 react 기반의 풀스택 웹개발 프레임워크 https://nextjs.org/ https://vercel.com/
Created: 2024-10-11T05:30:39.021Z Updated: 2024-10-11T05:43:48.183Z
생성형 AI 서비스
https://www.perplexity.ai/
Created: 2024-10-11T05:44:41.227Z Updated: 2024-10-11T05:45:25.419Z
React v18
The library for web and native user interfaces
Created: 2024-10-15T06:56:39.024Z Updated: 2024-10-15T06:56:39.024Z

축하합니다