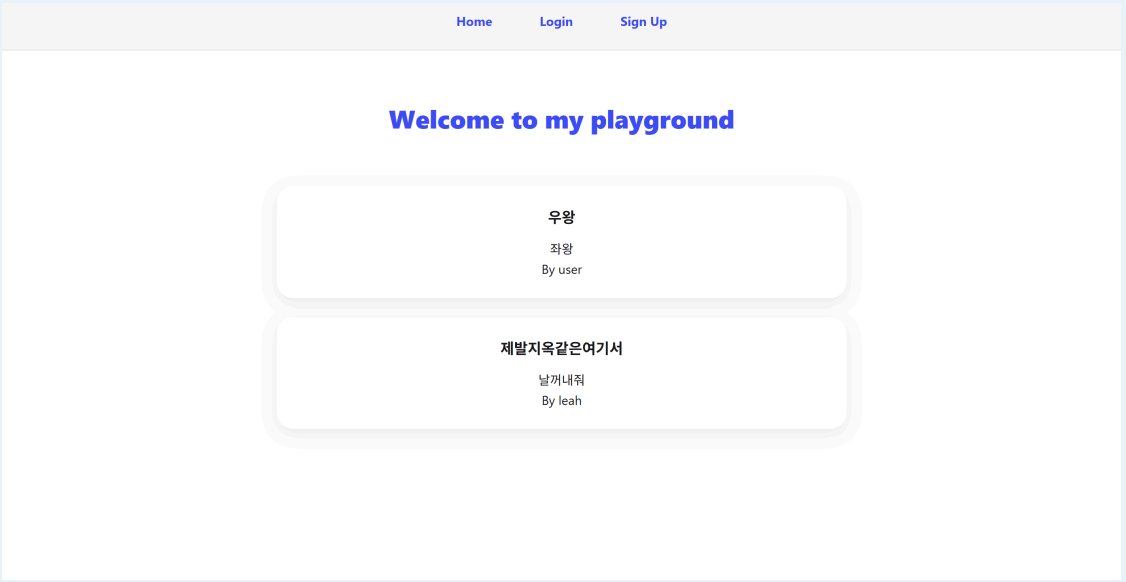
2025350202 전유정 보고서

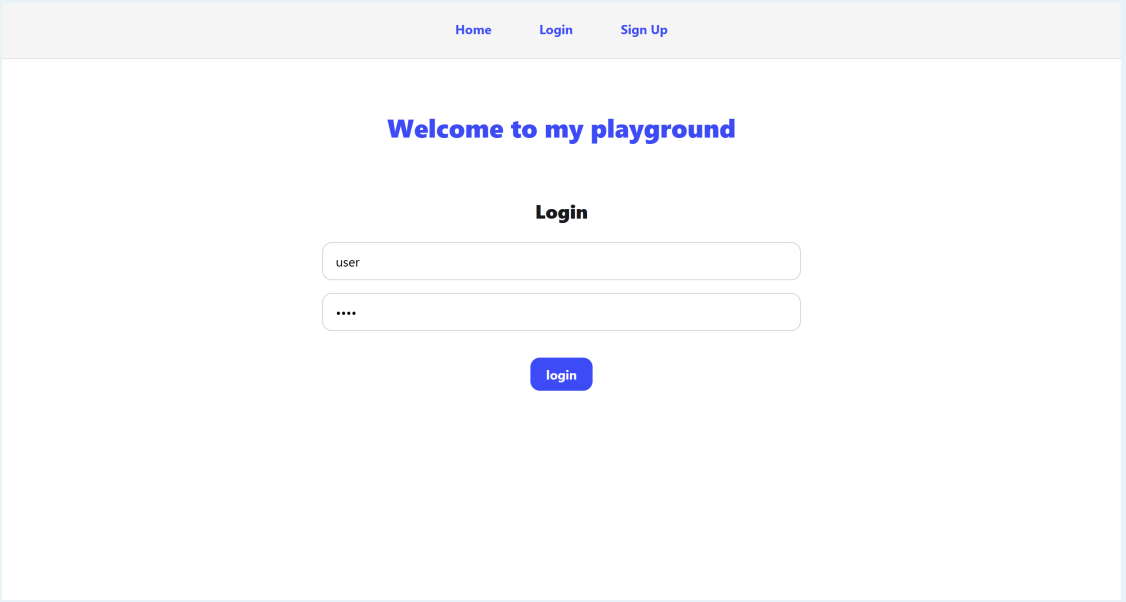
파일 구성

```
backend/
  - models/
      - Post.js
      - User.js
  - routes/
      - auth.js
      - posts.js
  - Dockerfile
  - server.js

frontend/
  - public/
       - index.html
  - src/
       - components/
            - CreatePost.js
            - EditPost.js
            - Layout.jsx
            - Login.jsx
            - PostDetail.js
            - PostList.js
            - SignUp.jsx
      - App.js
      - index.css
      - index.js
  - Dockerfile

docker-compose.yml
```

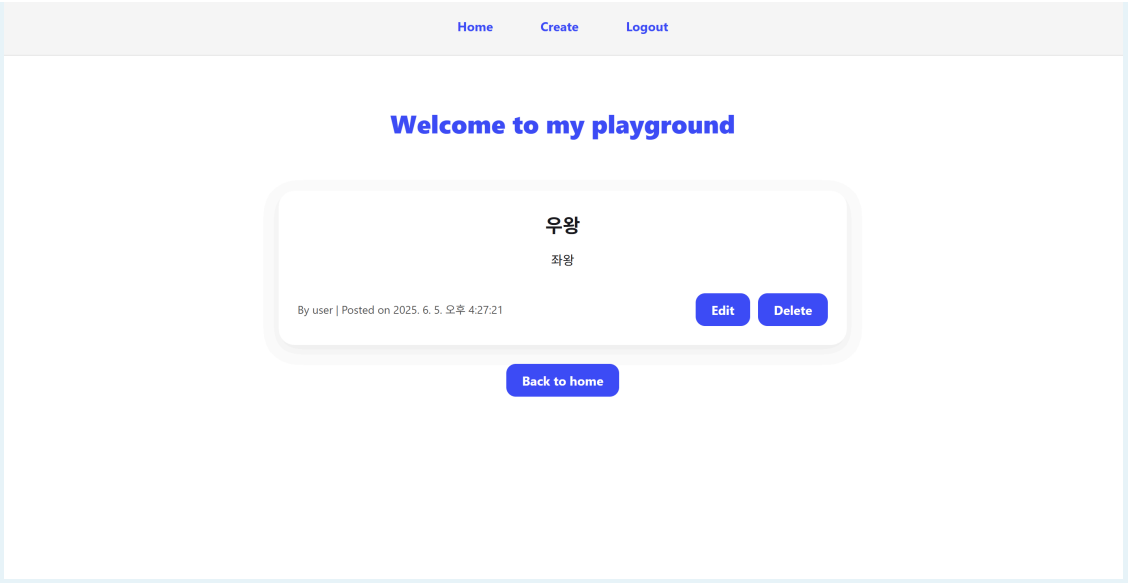
웹 구성



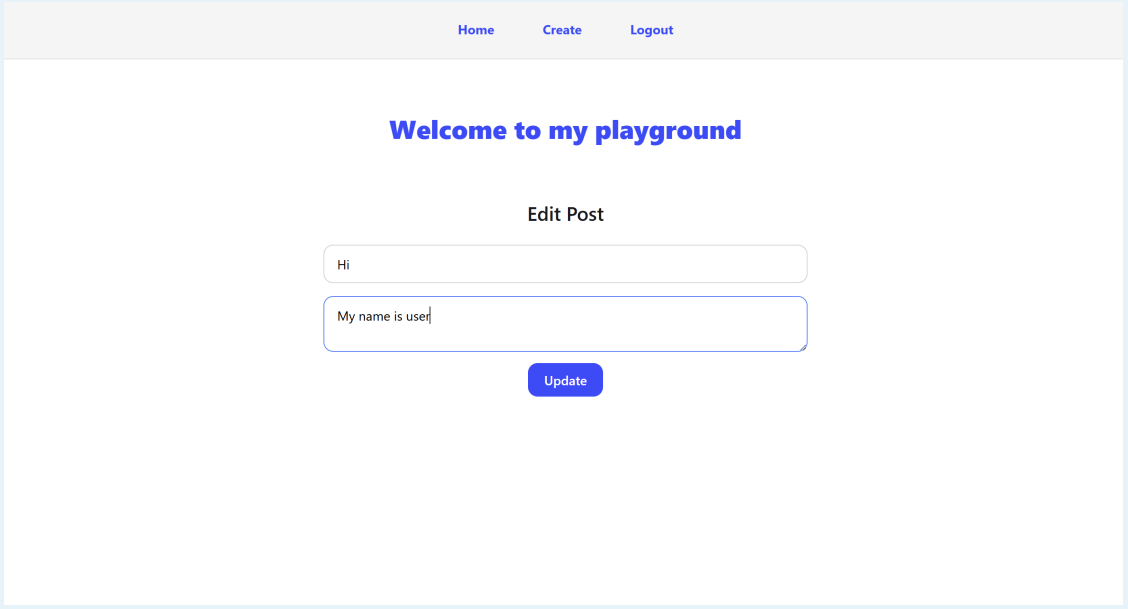
첫 화면이다. post list가 나와있다.

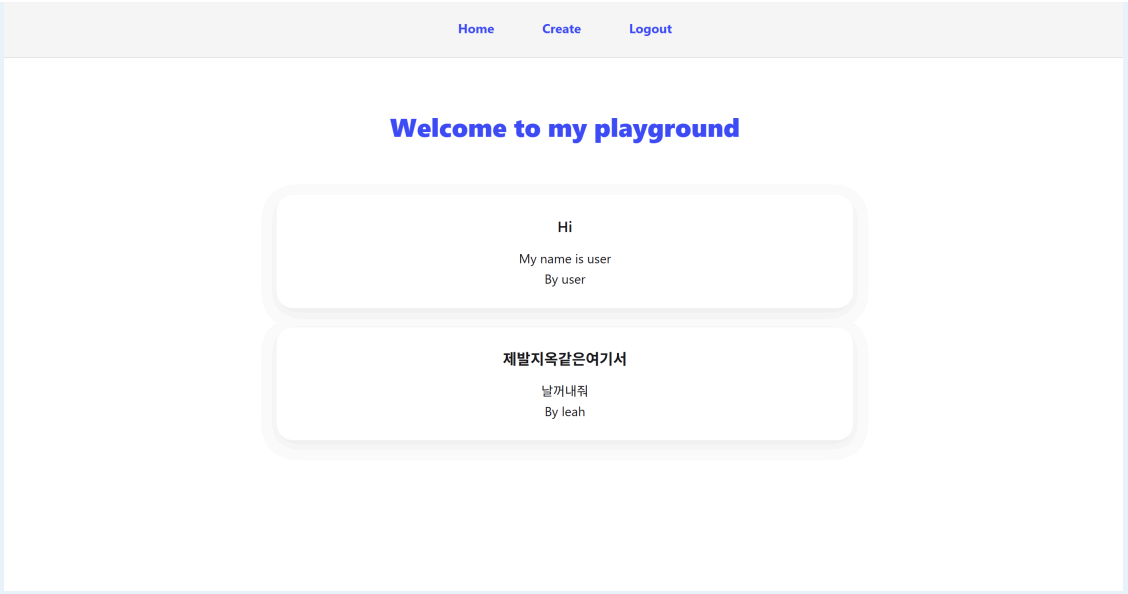


user로 로그인을 하면

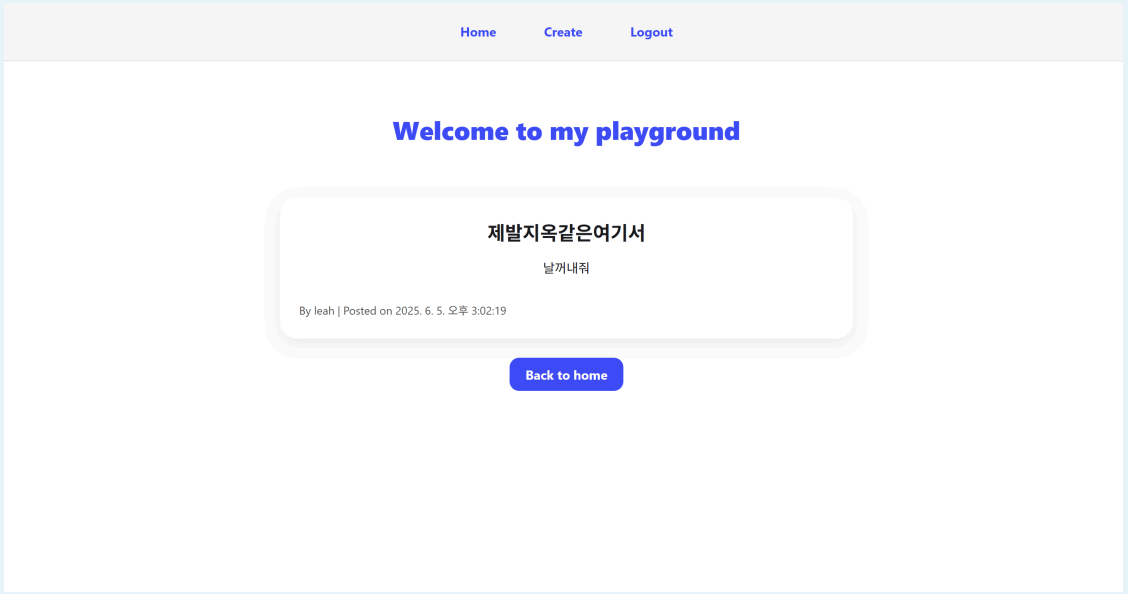


글쓴이가 user = 로그인한 유저이므로 edit, delete 버튼이 뜬다.

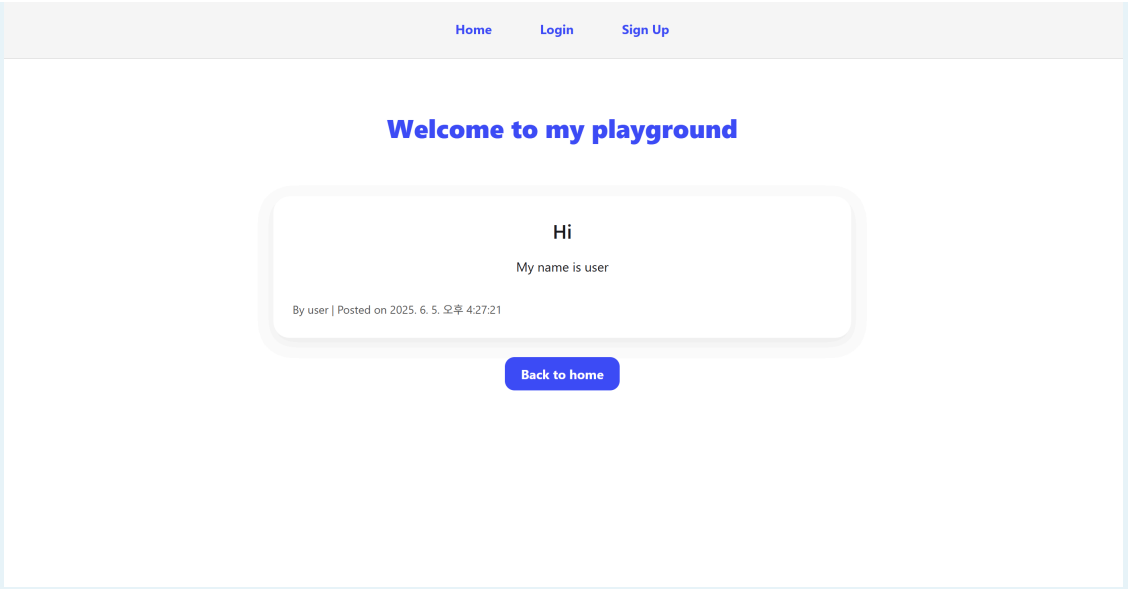




수정이 된 것을 볼 수 있다.

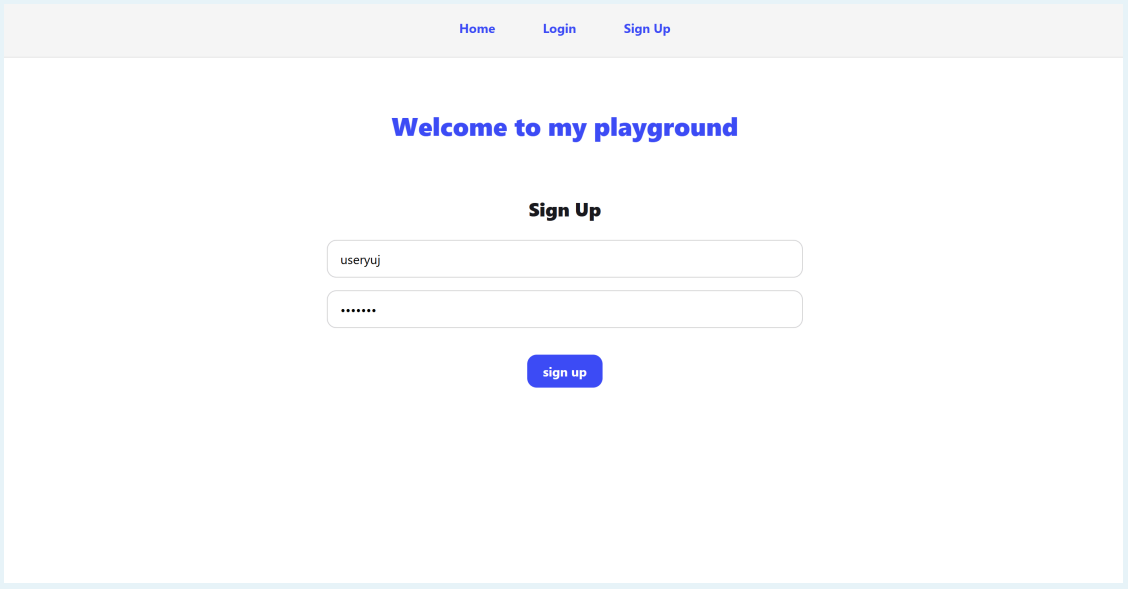


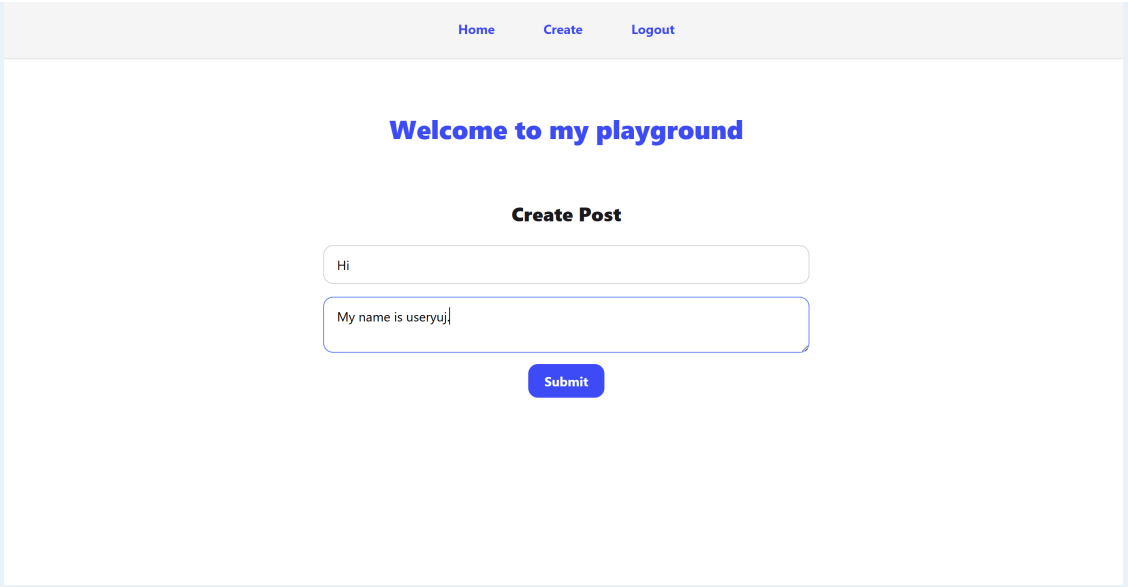
반면 leah가 쓴 글을 보면 edit delete 할 수 없다.

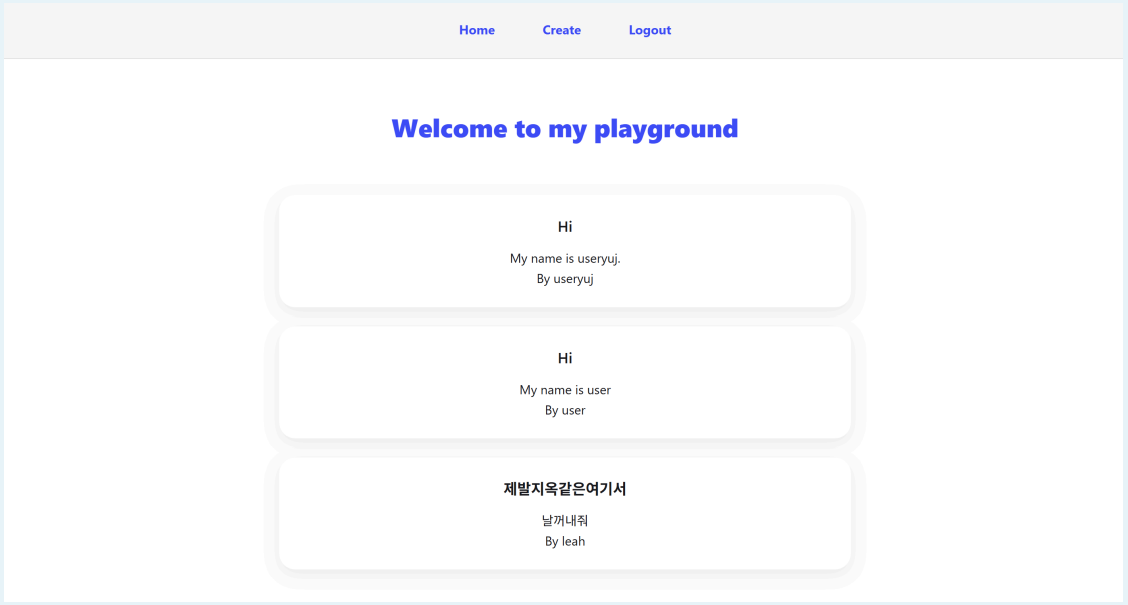


로그아웃을 하면 아무 글도 수정할 수 없고 작성할 수도 없다.

새로 가입해서 글을 써보자.







잘 작성이 되는 것을 볼 수 있다.

Backend

server.js

```
// backend 진입점
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const postsRoute = require('./routes/posts');
const authRoute = require('./routes/auth');

const app = express();

app.use(cors());
app.use(express.json());

app.use('/posts', postsRoute);
app.use('/auth', authRoute);

const mongoUri = process.env.MONGO_URI || 'mongodb://mongo:27017/mern-crud';

mongoose.connect(mongoUri)
  .then(() => {
    app.listen(5000, () => console.log('Backend running on http://localhost:5000'));
  })
  .catch((err) => console.error('MongoDB 연결 실패:', err));
```

백엔드 서버의 진입점이자, Express 앱을 초기화하고 MongoDB와의 연결을 설정한다. 게시글 및 사용자 인증 관련 API 라우팅을 정의가 담겼다. CORS 및 JSON 요청 처리를 위한 미들웨어도 적용되었다.

Post.js

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

const PostSchema = new mongoose.Schema({
  title: String,
  content: String,
  author: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

}, { timestamps: true });

module.exports = mongoose.model('Post', PostSchema);
```

게시글 정보를 저장하는 Mongoose 모델로, 제목(title), 내용(content), 작성자(author) 정보를 포함하며, 작성자는 User 모델을 참조한다. timestamps 옵션을 통해 생성 및 수정 시간을 자동으로 기록한다.

User.js

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

const UserSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});

module.exports = mongoose.model('User', UserSchema);
```

사용자 정보를 저장하는 Mongoose 모델로, 사용자 이름(username)과 비밀번호(password)를 필수로 포함한다. 이 모델은 Post.js의 author 필드와 참조 관계를 맺는다.

auth.js

```
// signup
router.post('/signup', async (req, res) => {
  const { username, password } = req.body;

  try {
    const userExists = await User.findOne({ username });
    if (userExists) return res.status(400).json({ msg: 'User already exists' });

    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ username, password: hashedPassword });
    await newUser.save();

    res.json({ msg: 'Signup successful' });
  } catch (err) {
    res.status(500).json({ error: 'Server error' });
  }
});

// login
router.post('/login', async (req, res) => {
  const { username, password } = req.body;

  try {
    const user = await User.findOne({ username });
    if (!user) return res.status(400).json({ msg: 'User not found' });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ msg: 'Invalid credentials' });

    const token = jwt.sign({ id: user._id }, JWT_SECRET, { expiresIn: '1h' });
    res.json({ token });
  } catch (err) {
    res.status(500).json({ error: 'Server error' });
  }
});

module.exports = router;
```

사용자 인증을 담당하는 라우터이다. signup 부분은 새로운 사용자를 등록하고, login 부분은 비밀번호를 검증한 뒤 유효하면 JWT 토큰을 발급한다. 비밀번호는 bcrypt를 사용해 암호화되며, 보안을 위해 토큰은 1시간 뒤 만료된다.

posts.js

```
router.get('/', async (req, res) => {
  try {
    const posts = await Post.find().sort({ createdAt: -1 }).populate('author', '_id username');
    res.json(posts);
  } catch (err) {
    res.status(500).json({ message: 'Server error' });
  }
});

router.post('/', async (req, res) => {
  const { title, content } = req.body;
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ message: 'Unauthorized: no token provided' });
  }

  const token = authHeader.split(' ')[1];

  try {
    const payload = jwt.verify(token, JWT_SECRET);
    const authorId = payload.id;

    if (!authorId) {
      return res.status(401).json({ message: 'Invalid token: no user id' });
    }

    const post = new Post({
      title,
      content,
      author: new mongoose.Types.ObjectId(authorId),
    });

    await post.save();
    res.status(201).json(post);

  } catch (err) {
    console.error('Error in POST /posts:', err);
    if (err.name === 'JsonWebTokenError') {
      return res.status(401).json({ message: 'Invalid token' });
    } else if (err.name === 'TokenExpiredError') {
      return res.status(401).json({ message: 'Token expired' });
    }
    res.status(500).json({ message: 'Server error' });
  }
});
```

```
router.get('/:id', async (req, res) => {
  try {
    const post = await Post.findById(req.params.id).populate('author', '_id username');
    if (!post) return res.status(404).json({ message: 'Post not found' });
    res.json(post);
  } catch (err) {
    res.status(500).json({ error: 'Server error' });
  }
});

router.put('/:id', async (req, res) => {
  try {
    const post = await Post.findByIdAndUpdate(req.params.id, req.body, { new: true });
    res.json(post);
  } catch (err) {
    res.status(500).json({ message: 'Server error' });
  }
});

router.delete('/:id', async (req, res) => {
  try {
    await Post.findByIdAndDelete(req.params.id);
    res.json({ message: 'Deleted' });
  } catch (err) {
    res.status(500).json({ message: 'Server error' });
  }
});

module.exports = router;
```

게시글을 생성, 조회, 수정, 삭제하는 REST API이다. 글 작성 시 JWT 인증을 통해 작성자를 식별하며, 글 목록은 작성일 기준으로 최신순 정렬된다. JWT 오류 및 서버 에러에 대한 예외 처리도 포함되어 있다.

Frontend

App.js

```
function App() {
  const [auth, setAuth] = useState(false);

  useEffect(() => {
  const token = localStorage.getItem('token');
  if (token) {
    const payload = JSON.parse(atob(token.split('.')[1]));
    if (Date.now() / 1000 > payload.exp) {
      localStorage.removeItem('token');
    }
  }
  }, []);

  return (
      <Router>
      <Layout auth={auth} setAuth={setAuth}>
        <Routes>
          <Route path="/" element={<PostList />} />
          <Route path="/create" element={<CreatePost />} />
          <Route path="/edit/:id" element={<EditPost />} />
          <Route path="/posts/:id" element={<PostDetail />} />
          <Route path="/signup" element={<SignUp />} />
          <Route path="/login" element={<Login setAuth={setAuth} />} />
        </Routes>
      </Layout>
      </Router>
  );
}

export default App;
```

React 라우팅을 구성하며, localStorage의 JWT를 이용해 로그인 상태를 확인하고 관리한다. 로그인 상태는 login 컴포넌트로 전달되며, 전체 앱의 인증 흐름을 제어한다.

CreatePost.js

```
const CreatePost = () => {
  const [title, setTitle] = useState('');
  const [content, setContent] = useState('');
  const navigate = useNavigate();

  useEffect(() => {
    const token = localStorage.getItem('token');
    if (!token) {
      alert('You should login to create post.');
      navigate('/');
    }
  }, [navigate]);

  const submitPost = async () => {
    const token = localStorage.getItem('token');
    if (!token) {
    alert('You are not authorized to create a post.');
    return;
    }

    try {
      await axios.post('http://localhost:5000/posts', { title, content },
        {
          headers: {
            Authorization: `Bearer ${token}`,
            'Content-Type': 'application/json',
          },
        }
      );
      navigate('/');
    } catch (err) {
      alert('Post creation failed.');
      console.error(err);
    }
  };
```

React 컴포넌트로, 로그인 토큰이 없으면 홈으로 이동시켜서 접근을 제한한다. 제목과 내용을 입력받아 서버에 게시글을 작성하는 POST 요청을 보낸다. 작성 완료 후에는 홈(이자 리스트)로 이동한다.

EditPost.js

```
const EditPost = () => {
  const [title, setTitle] = useState('');
  const [content, setContent] = useState('');
  const [authorId, setAuthorId] = useState(null);
  const { id } = useParams();
  const navigate = useNavigate();

  useEffect(() => {
    const token = localStorage.getItem('token');
    if (!token) {
      alert('You need to login.');
      navigate('/login');
      return;
    }

    const fetchPost = async () => {
      try {
        const res = await axios.get(`http://localhost:5000/posts/${id}`, {
          headers: { Authorization: `Bearer ${token}` }
        });

        setAuthorId(res.data.author);
        setTitle(res.data.title);
        setContent(res.data.content);

        const payload = JSON.parse(atob(token.split('.')[1]));
        const currentUserId = payload.id;
        const authorId = typeof res.data.author === 'object' ? res.data.author._id : res.data.author;

        if (authorId !== currentUserId) {
          alert('You are not allowed to edit.');
          navigate('/');
        }
      } catch (error) {
        alert('Failed to load post.');
        navigate('/');
      }
    };

    fetchPost();
  }, [id, navigate]);
```

```
  const updatePost = async () => {
    try {
      const token = localStorage.getItem('token');
      await axios.put(`http://localhost:5000/posts/${id}`,
        { title, content },
        { headers: { Authorization: `Bearer ${token}` } }
      );
      navigate('/');
    } catch (error) {
      alert('Failed to edit.');
    }
  };

  return (
    <div className="content-container">
      <h2>Edit Post</h2>
      <input value={title} onChange={(e) => setTitle(e.target.value)} /><br />
      <textarea value={content} onChange={(e) => setContent(e.target.value)} /><br />
      <button onClick={updatePost}>Update</button>
    </div>
  );
};

export default EditPost;
```

로그인된 사용자가 특정 게시글을 불러와 제목과 내용을 수정할 수 있게한다. 작성자와 로그인한 사용자가 같지 않으면 수정 권한이 없다고 알리고 홈으로 이동시킨다. (근데 사실 작성자 본인이 아니라면 edit 버튼이 뜨지도 않는다..) 수정한 내용은 서버에 PUT 요청으로 저장하고, 완료되면 홈으로 이동한다.

Layout.jsx

```jsx
const Layout = ({ auth, setAuth, children }) => {
  const navigate = useNavigate();

  const handleLogout = () => {
    localStorage.removeItem('token');
    setAuth(false);
    navigate('/');
  };

  return (
    <>
      <header className="header">
        <nav className="nav">
          <button><Link to="/"><strong>Home</strong></Link></button>
          {auth ? (
            <>
              <button><Link to="/create"><strong>Create</strong></Link></button>
              <button onClick={handleLogout}><strong>Logout</strong></button>
            </>
          ) : (
            <>
              <button><Link to="/login"><strong>Login</strong></Link></button>
              <button><Link to="/signup"><strong>Sign Up</strong></Link></button>
            </>
          )}
        </nav>
      </header>
      <main className="content-container">
        <h1 className="welcome"><strong>Welcome to my playground</strong></h1>
        {children}
      </main>
    </>
  );
};

export default Layout;
```

페이지의 레이아웃과 네비게이션 바를 담당한다. Home 버튼은 고정이고, 로그인 상태에 따라 Login + Sign Up or Create + Logout 을 보여준다. Logout을 누르면 토큰을 삭제하고 홈으로 이동한다.

SignUp.jsx

```jsx
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import '../index.css';

const SignUp = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  const handleSignUp = async () => {
    try {
      await axios.post('http://localhost:5000/auth/signup', { username, password });
      alert('Now you are the member!');
      navigate('/login');
    } catch (err) {
      alert('Failed to signup. (already exists)');
    }
  };

  return (
    <div className="content-container">
      <h2><strong>Sign Up</strong></h2>
      <input placeholder="Username" onChange={(e) => setUsername(e.target.value)} /><br />
      <input type="password" placeholder="Password" onChange={(e) => setPassword(e.target.value)} /><br /><br />
      <button onClick={handleSignUp}><strong>sign up</strong></button>
    </div>
  );
};

export default SignUp;
```

회원가입 페이지를 담당한다. 사용자가 아이디와 비밀번호를 입력하면 서버에 회원가입 요청을 보내고, 성공시 알림을 띄운 후 로그인 페이지로 이동한다.

Login.jsx

```jsx
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import '../index.css';

const Login = ({ setAuth }) => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  const handleLogin = async () => {
    try {
      const res = await axios.post('http://localhost:5000/auth/login', { username, password });
      localStorage.setItem('token', res.data.token);
      setAuth(true);
      navigate('/');
    } catch (err) {
      alert('Failed to login.');
    }
  };

  return (
    <div className="content-container">
      <h2><strong>Login</strong></h2>
      <input placeholder="Username" onChange={(e) => setUsername(e.target.value)} /><br />
      <input type="password" placeholder="Password" onChange={(e) => setPassword(e.target.value)} /><br /><br />
      <button onClick={handleLogin}><strong>login</strong></button>
    </div>
  );
};

export default Login;
```

로그인 페이지를 담당한다. 사용자가 아이디와 비밀번호를 입력하면 서버에 로그인 요청을 보내고, 성공 시 토큰을 저장하고 로그인 상태 setAuth를 true로 설정한 뒤 홈으로 이동한다. 실패하면 알림을 띄운다.

PostList.js

```jsx
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { Link } from 'react-router-dom';
import '../index.css';

const PostList = () => {
  const [posts, setPosts] = useState([]);

  const fetchPosts = async () => {
    const res = await axios.get('http://localhost:5000/posts');
    setPosts(res.data);
  };

  useEffect(() => {
    fetchPosts();
  }, []);

  return (
    <div className="content-container">
      {posts.map((post) => (
        <div className="post-card" key={post._id}>
          <Link to={`/posts/${post._id}`}>
            <h3>{post.title}</h3>
            <p>{post.content}</p>
            <p>By {post.author?.username}</p>
          </Link>
        </div>
      ))}
    </div>
  );
};

export default PostList;
```

게시글 목록을 보여준다. useEffect로 처음 렌더링될 때 서버에서 게시글 데이터를 가져와 posts 에 저장한다. list에서 게시글은 title, content, author를 보여주고 클릭하면 상세페이지(PostDetail)로 이동한다.

PostDetail.js

```
const PostDetail = () => {
  const { id } = useParams();
  const navigate = useNavigate();
  const [post, setPost] = useState(null);
  const [currentUserId, setCurrentUserId] = useState(null);

  useEffect(() => {
    const token = localStorage.getItem('token');
    if (token) {
        const payload = JSON.parse(atob(token.split('.')[1]));
        setCurrentUserId(payload.id);
    }

    const fetchPost = async () => {
        try {
            const res = await axios.get(`http://localhost:5000/posts/${id}`);
            console.log('Fetched post:', res.data);
            setPost(res.data);
        } catch (err) {
            alert('Failed to load post.');
            navigate('/');
        }
    };
    fetchPost();
  }, [id, navigate]);

  const deletePost = async () => {
    try {
      const token = localStorage.getItem('token');
      await axios.delete(`http://localhost:5000/posts/${id}`, {
        headers: { Authorization: `Bearer ${token}` }
      });
      alert('Deleted.');
      navigate('/');
    } catch (err) {
      alert('Failed to delete.');
    }
  };
```

```
  if (!post) return <div>Loading...</div>;

  return (
    <div className="content-container">
        <div className="post-card">
            <h2>{post.title}</h2>
            <p>{post.content}</p>

            <div className="post-footer">
                <p className="author-text">
                By {post.author?.username} | Posted on {new Date(post.createdAt).toLocaleString()}
                </p>

                {currentUserId && String(post.author?._id || post.author) === String(currentUserId) && (
                <div className="post-button-group">
                    <Link to={`/edit/${post._id}`}>
                    <button><strong>Edit</strong></button>
                    </Link>
                    <button onClick={deletePost}><strong>Delete</strong></button>
                </div>
                )}
            </div>
        </div>
        <Link to={`/`}><button><strong>Back to home</strong></button></Link>
    </div>
  );
};

export default PostDetail;
```

게시글 상세(?)페이지를 보여준다. 게시글 ID를 받아와 서버에서 해당 게시글을 불러온다. 로그인된 사용자 = 해당 게시글 작성자 라면 edit delte 버튼이 보인다. 삭제 시 서버에 요청을 보내고 완료되면 홈으로 이동한다. + 작성자와 게시 날짜, home으로 가는 버튼은 로그인 여부와 상관없이 존재한다.