Assigment4 보고서 - 2024315325 서지은

1. mongoDB

1. Users collection은 id, password, roles를 포함한다.

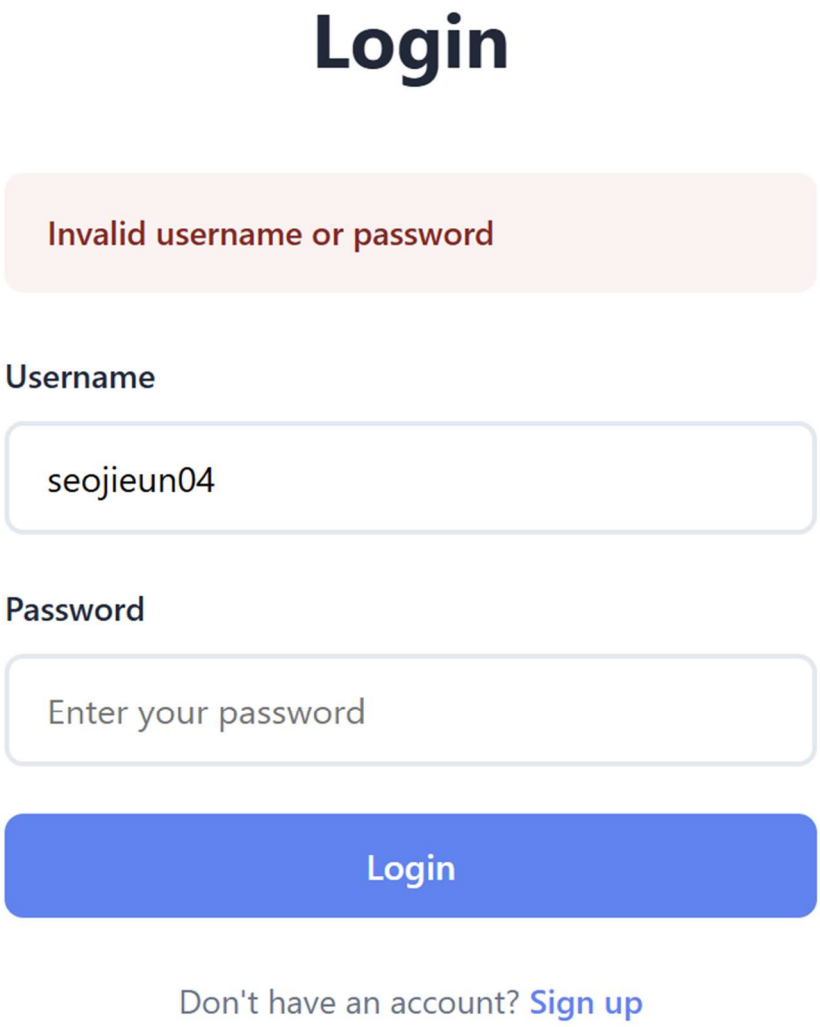
2. Posts collection은 author, content, likes, createdAt를 포함한다.

```
root@BOOK-V9TF0TTAPO:~/wp_pa2# mongosh
Current Mongosh Log ID: 694bc1aaeaa18d81b88de665
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTim
Using MongoDB:          7.0.28
Using Mongosh:          2.5.10

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-12-21T14:17:26.574+09:00: Using the XFS filesystem is strongly recommended with the
   2025-12-21T14:17:27.215+09:00: Access control is not enabled for the database. Read and
   2025-12-21T14:17:27.215+09:00: vm.max_map_count is too low
------

test> use nodejs
switched to db nodejs
nodejs> show collections
...
posts
sessions
users
nodejs> db.getCollectionNames()
...
[ 'users', 'sessions', 'posts' ]
nodejs> db.users.findOne({}, { _id: 0, id: 1, password: 1, roles: 1 })
...
{
  id: 'seojieun05',
  password: '$2a$10$.GX27zLjpW.8f1LVNO6ad.LRbsCs8OK8JQmibSj4oKysXxbrDuZc6',
  roles: [ 'user' ]
}
nodejs> db.posts.findOne({}, { _id: 0, author: 1, content: 1, liklikes: 1, createdAt: 1 })
...
{
  author: 'seojieun05',
  content: 'hi this is test for PA2',
  likes: 2,
  createdAt: ISODate('2025-12-21T05:38:31.189Z')
}
```

2. 로그인

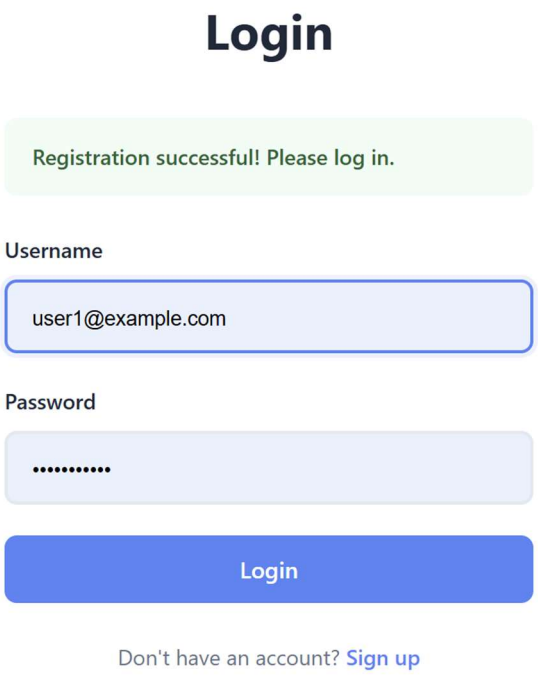
1. 기존 사용자가 로그인 할 수 있어야 한다

2. 아이디, 비밀번호가 올바르지 않으면 에러 메세지

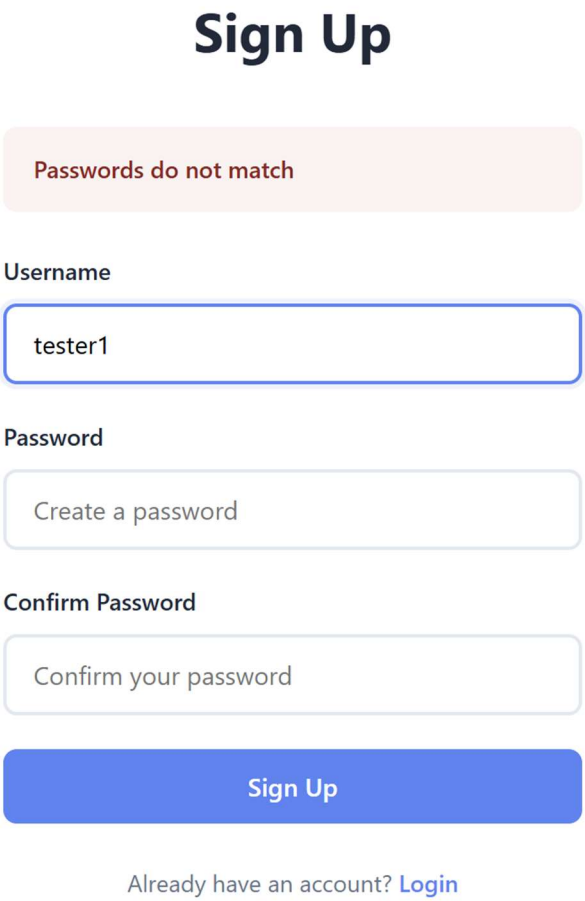


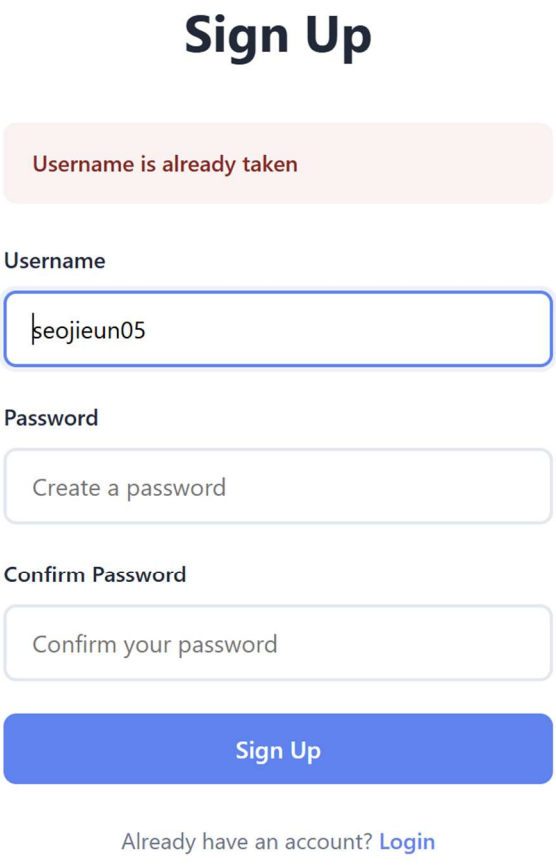
3. 회원가입

1. 회원가입 기능 구현



2. 비밀번호 일치하지 않거나 아이디가 이미 사용중이면 에러를 표시한다

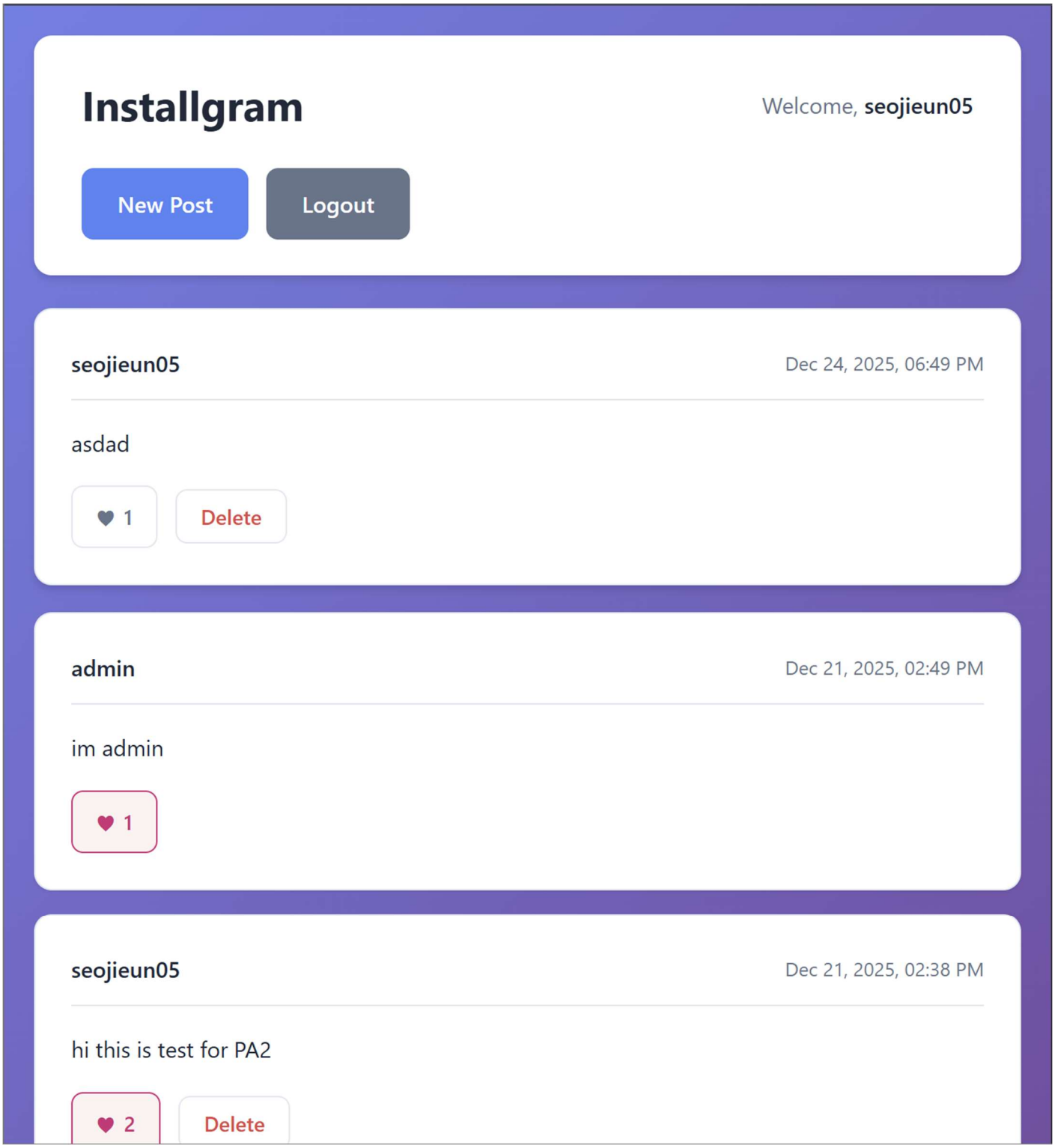




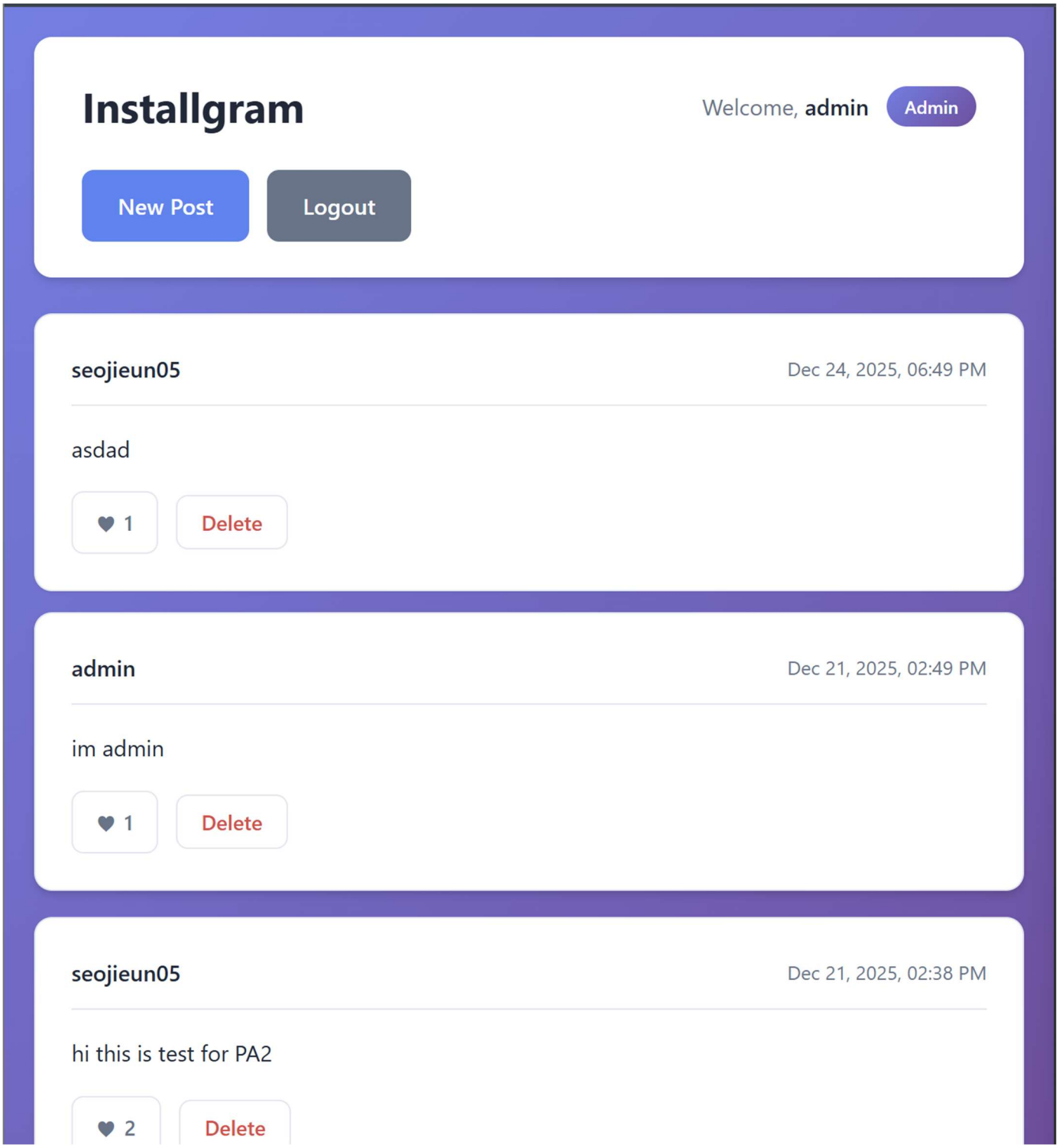
4. 메인페이지

1. 페이지 상단에 유저 이름 표시, 새 게시글 작성할 수 있는 버튼

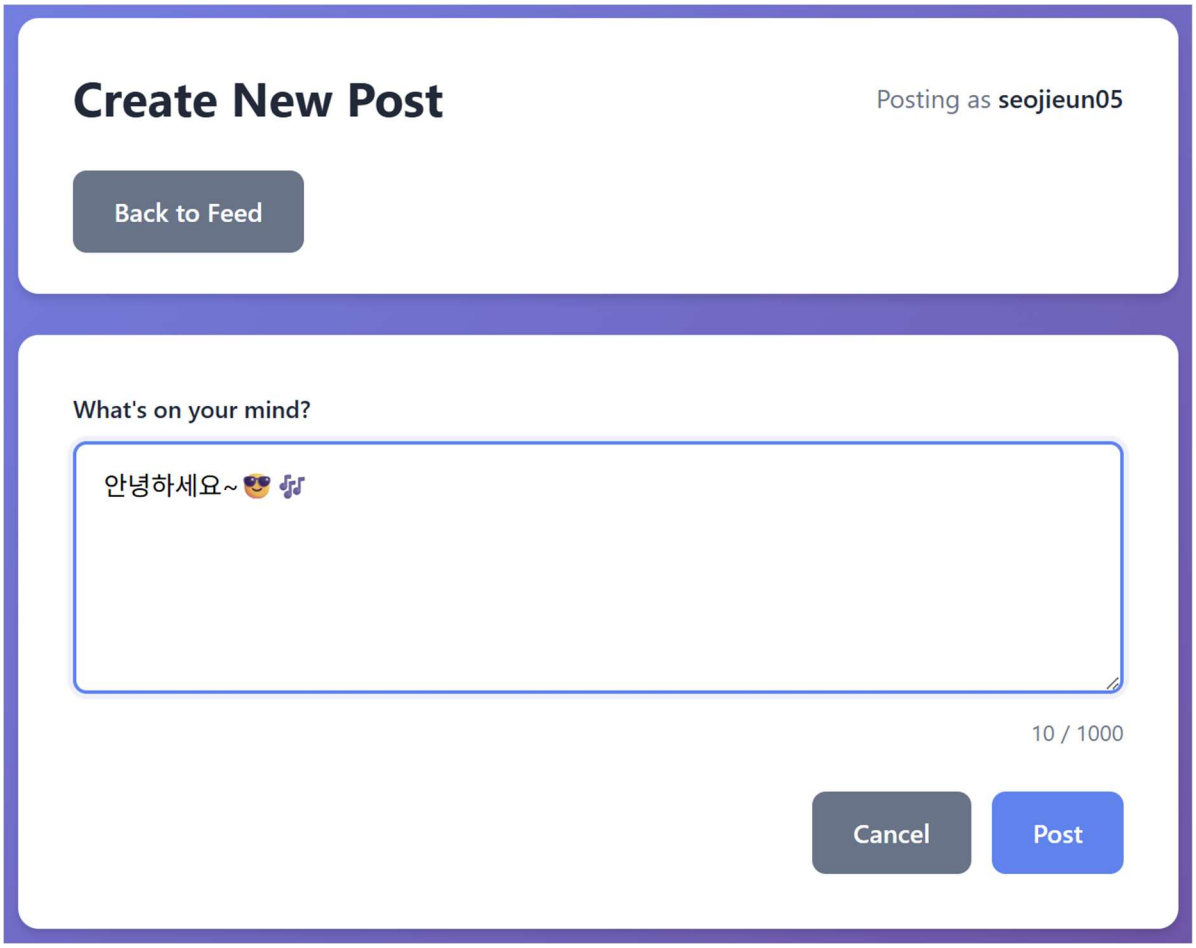
2. 각 게시글에 작성자와 내용, 좋아요 버튼, 게시글을 시간 순으로 정렬, 작성자는 본인 게시글을 삭제할 수 있다.



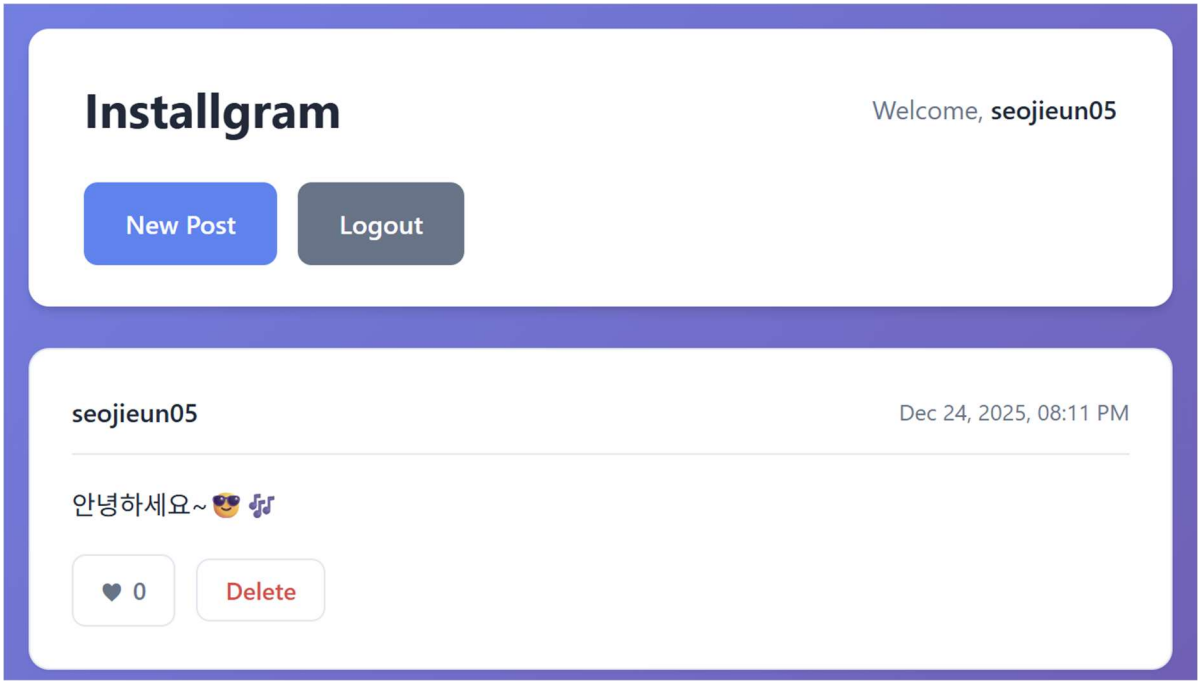
3. 관리자는 모든 게시글을 삭제할 수 있다. (id: admin, pw: admin123)



4. 사용자가 새 게시글을 작성할 수 있다. 게시글 작성을 성공하면 메인페이지로 리다이렉트한다.



(바로 메인화면으로 이동한다)



5. 서버의 컨트롤러 함수들

1. Authentication Controllers

1-1. signupUser(req, res): 사용자 회원가입을 처리하는 함수. users 컬렉션에 저장하기 전에 비밀번호를 해시한다. username이 유일한지 검증한다.

```
const User = require("../models/User")

exports.signupUser = async (req, res) => {
  try {
    const { username, password, confirmPassword } = req.body

    // Validation
    if (!username || !password || !confirmPassword) {
      return res.render("signup", {
        error: "All fields are required",
        username: username || "",
      })
    }

    if (password !== confirmPassword) {
      return res.render("signup", {
        error: "Passwords do not match",
        username,
      })
    }

    if (password.length < 4) {
      return res.render("signup", {
        error: "Password must be at least 4 characters",
        username,
      })
    }

    // Check if username already exists
    const existingUser = await User.findOne({ id: username })
    if (existingUser) {
      return res.render("signup", {
        error: "Username is already taken",
        username,
      })
    }

    // Create new user (password is hashed in pre-save hook)
    const newUser = new User({
      id: username,
      password: password,
      roles: ["user"],
    })

    await newUser.save()

    // Redirect to login page
    res.redirect("/login?registered=true")
  } catch (error) {
    console.error("Signup error:", error)
    res.render("signup", {
      error: "An error occurred during registration",
      username: req.body.username || "",
    })
  }
}
```

1-2. loginUser(req, res): 사용자 로그인을 처리하는 함수. 저장된 해시 비밀번호와 비교하여 자격 증명을 검증한다. 로그인 성공 시 세션을 초기화한다.

```
58  exports.loginUser = async (req, res) => {
59    try {
60      const { username, password } = req.body
61
62      // Validation
63      if (!username || !password) {
64        return res.render("login", {
65          error: "Username and password are required",
66          username: username || "",
67        })
68      }
69
70      // Find user
71      const user = await User.findOne({ id: username })
72      if (!user) {
73        return res.render("login", {
74          error: "Invalid username or password",
75          username,
76        })
77      }
78
79      // Check password
80      const isMatch = await user.comparePassword(password)
81      if (!isMatch) {
82        return res.render("login", {
83          error: "Invalid username or password",
84          username,
85        })
86      }
87
88      // Initialize session
89      req.session.user = {
90        id: user.id,
91        roles: user.roles,
92      }
93
94      res.redirect("/main")
95    } catch (error) {
96      console.error("Login error:", error)
97      res.render("login", {
98        error: "An error occurred during login",
99          username: req.body.username || "",
100       })
101     }
102   }
103
```

1-3. logoutUser(req, res): 세션을 파기하여 로그아웃을 처리한다.

```
exports.logoutUser = (req, res) => {
  req.session.destroy((err) => {
    if (err) {
      console.error("Logout error:", err)
    }
    res.redirect("/login")
  })
}
```

2. Post Controllers

2-1. createPost(req, res): 새 게시글 생성을 처리하는 함수. posts 컬렉션에 게시글 내용과 작성자를 저장한다. 사용자가 인증되었는지 확인한다.

```
exports.createPost = async (req, res) => {
  try {
    const { content } = req.body

    if (!content || content.trim() === "") {
      return res.render("newpost", {
        error: "Post content cannot be empty",
        user: req.session.user,
      })
    }

    const newPost = new Post({
      author: req.session.user.id,
      content: content.trim(),
      likes: 0,
      createdAt: new Date(),
    })

    await newPost.save()
    res.redirect("/main")
  } catch (error) {
    console.error("Create post error:", error)
    res.render("newpost", {
      error: "Failed to create post",
      user: req.session.user,
    })
  }
}
```

2-2. getPosts(req, res): posts 컬렉션에서 모든 게시글을 가져와 반환하는 함수. main.ejs 페이지에 게시한다.

```
exports.getPosts = async (req, res) => {
  try {
    const posts = await Post.find().sort({ createdAt: -1 }) // Newest first

    res.render("main", {
      posts,
      user: req.session.user,
    })
  } catch (error) {
    console.error("Get posts error:", error)
    res.render("main", {
      posts: [],
      user: req.session.user,
      error: "Failed to load posts",
    })
  }
}
```

2-3. deletePost(req, res): 사용자가 인증되어 있고, 해당 게시글의 작성자이거나 admin인지 검증하는 함수. 지정된 게시글을 posts 컬렉션에서 삭제한다. 성공적으로 삭제하면 main.ejs로 리다이렉트한다.

```
exports.deletePost = async (req, res) => {
  try {
    const postId = req.params.id
    const user = req.session.user

    const post = await Post.findById(postId)

    if (!post) {
      return res.redirect("/main")
    }

    // Check if user is author or admin
    const isAuthor = post.author === user.id
    const isAdmin = user.roles && user.roles.includes("admin")

    if (!isAuthor && !isAdmin) {
      return res.redirect("/main")
    }

    await Post.findByIdAndDelete(postId)
    res.redirect("/main")
  } catch (error) {
    console.error("Delete post error:", error)
    res.redirect("/main")
  }
}
```

6. 해시

```
userSchema.pre("save", async function (next) {
  if (!this.isModified("password")) return next()

  try {
    const salt = await bcrypt.genSalt(10)
    this.password = await bcrypt.hash(this.password, salt)
    next()
  } catch (error) {
    next(error)
  }
})
```

DB에 저장하기 전, 해시해서 저장함.