웹 어플리케이션 보안

중부위키

윤준호 노인규 한상우 전예진 김강욱

목차

- 프로젝트 목표
- 위키란?
- 기능
  - 사용자 등록 / 로그인
  - 구글 리캡차
  - 게시판
- 결과 분석
- Q&A

프로젝트 목표

- 학교에 관련된 소식들을 위키형식으로 쉽고 간편하게 접근할 수 있도록 한다.
- 비로그인 사용자와 로그인 사용자 모두 읽고 쓰기가 가능하게 한다.
- 보안 요소를 추가한다.

위키란?

위키는 불특정 다수가 협업을 통해 직접 내용과 구조를 수정할 수 있는 웹사이트를 말한다.

위키피디아

나무위키

사용자 등록

Register.component.ts

```
onRegisterSubmit() {
  if (this.recapt == null) {
    this.flashMessage.show('체크박스를 체크해주세요', { cssClass: 'alert-danger', timeout: 3000 });
    return false;
  }

  this.authService.registerUser(user).subscribe(data => {
    if (data.success) {
      this.flashMessage.show(data.msg, { cssClass: 'alert-success', timeout: 3000 });
      this.router.navigate(['/login']);
    } else {
      this.flashMessage.show(data.msg, { cssClass: 'alert-danger', timeout: 3000 });
      this.router.navigate(['/register']);
    }
  });
```

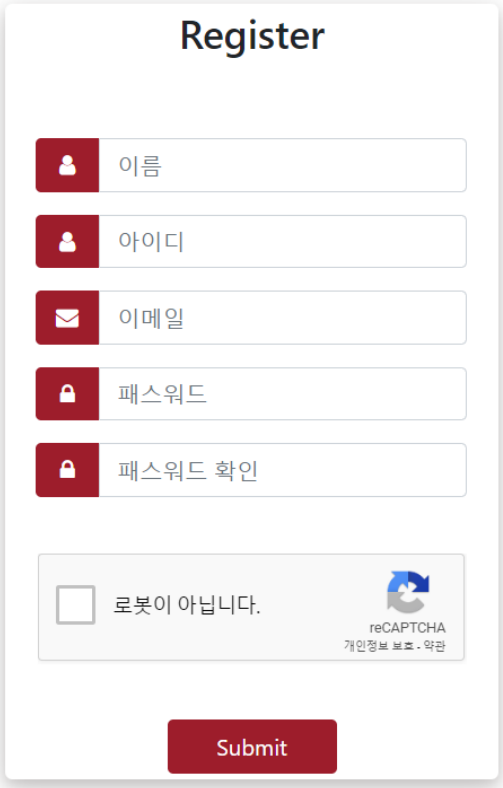
Auth.service.ts

```
registerUser(user): Observable<any> {
  const registerUrl = this.prepEndpoint('users/register');
  return this.http.post<User>(registerUrl, user, httpOptions);
}
```

Users.js

```
router.post('/register', (req, res, next) => {
  let newUser = new User({
    name: req.body.name,
    email: req.body.email,
    username: req.body.username,
    password: req.body.password
  });

  User.getUserByUsername(newUser.username, (err, user) => {
    if (err) throw err;
    if (user) {
      return res.json({ success: false, msg: "동일한 아이디가 존재합니다. 사용자 등록 실패." });
    } else {
      User.addUser(newUser, (err, user) => {
        if (err) {
          res.json({ success: false, msg: '사용자 등록 실패' });
        } else {
          res.json({ success: true, msg: '사용자 등록 성공' });
        }
      });
    }
  });
});
```



로그인

Login.component.ts

```
onLoginSubmit() {
  const login = {
    username: this.username,
    password: this.password
  };

  this.authService.authenticateUser(login).subscribe
    if (data.success) {
      this.authService.storeUserData(data.token, data
      this.flashMessage.show('로그인 성공', { cssCla
      this.router.navigate(['/list']);
    } else {
      this.flashMessage.show(data.msg, { cssClass: '
      this.router.navigate(['login']);
    }
  });
}
```

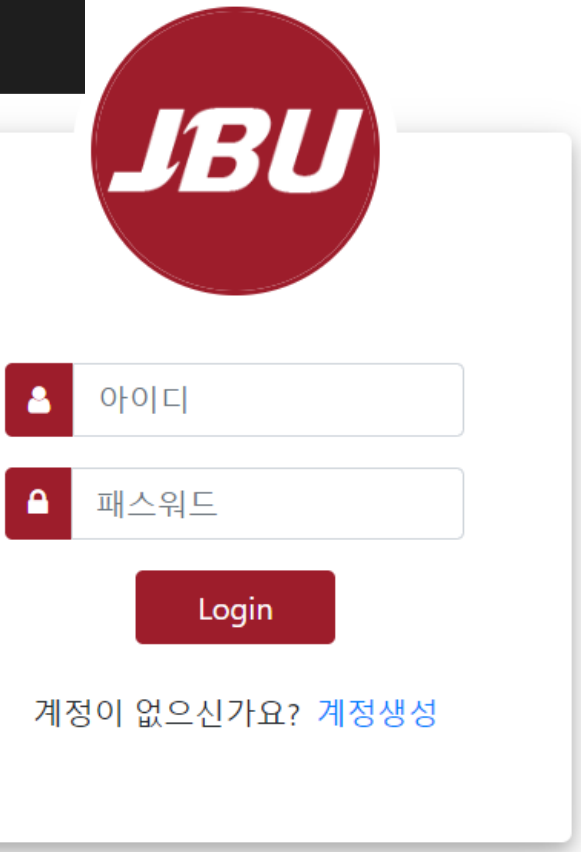
Auth.service.ts

```
authenticateUser(login): Observable<any> {
  const loginUrl = this.prepEndpoint('users/authenticate');
  return this.http.post<Login>(loginUrl, login, httpOptions);
}
```

```
storeUserData(token, userNoPW) {
  localStorage.setItem('id_token', token);
  localStorage.setItem('user', JSON.stringify(userNoPW));
  this.authToken = token;
  this.userNoPW = userNoPW;
}
```

Users.js

```
router.post('/authenticate', (req, res, next) => {
  const username = req.body.username;
  const password = req.body.password;
  User.getUserByUsername(username, (err, user) => {
    if (err) throw err;
    if (!user) {
      return res.json({ success: false, msg: "User not found! 등록된 사용자가 없습니다..." });
    }
    User.comparePassword(password, user.password, (err, isMatch) => {
      if (err) throw err;
      if (isMatch) {
        const token = jwt.sign({ data: user }, config.secret, {
          expiresIn: 604800
        });
        res.json({
          success: true,
          token: 'JWT ' + token,
          userNoPW: {
            name: user.name,
            username: user.username,
            email: user.email
          }
```



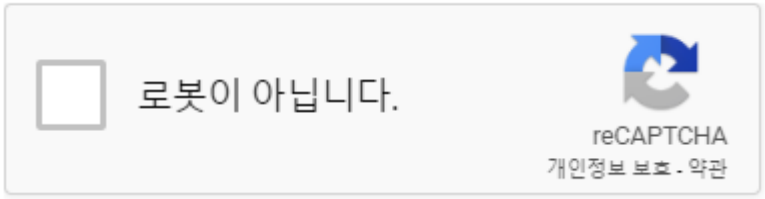
구글 리캡차

App.module.ts

```
import { RecaptchaModule } from 'ng-recaptcha';
```

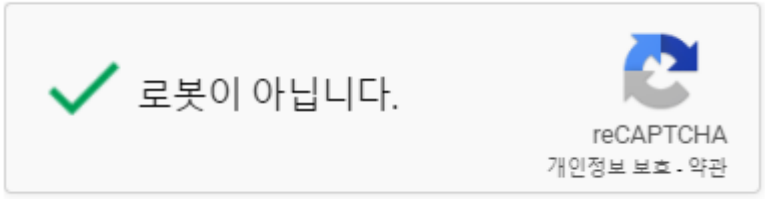
Register.component.html

```
<re-captcha (resolved)="resolved($event)"
siteKey="6Lf2wcQUAAAAAALsRs1t-1tt-S7rNgZRg9T6AKJ_
"></re-captcha>
```



Register.component.ts

```
resolved(captchaResponse: string) {
  console.log(`Resolved captcha with response: ${captchaResponse}`);
  this.recapt = captchaResponse;
}
if (this.recapt == null) {
  console.log('체크박스를 체크해주세요');
  this.flashMessage.show('체크박스를 체크해주세요',
   { cssClass: 'alert-danger', timeout: 3000 });
  return false;
}
```



* 통상적인 경우 사람 특유의 마우스 포인팅, 클릭/터치 패턴, 쿠키 값, 기타 알려지지 않은 구분방법을 통해 구별해 낸다

Create

Create-update.component.ts

```
createOrUpdate() {
  if (this.checkLoggedIn()) {
    if (this.board._id == undefined) {
      this.userString = localStorage.getItem('user');
      this.user = JSON.parse(this.userString);
      this.username = this.user.username;
      this.board.name = this.username;
      this.boardService.createBoard(this.board).subscribe(
        data => {
          console.log(data);
          this.router.navigate(['/list']);
        },
        error => {
          console.log(error);
        }
      )
} else {
  if (this.board._id == undefined) {
    this.board.name = "unknown";
    this.boardService.createBoard(this.board).subscribe(
      data => {
        console.log(data);
        this.router.navigate(['/list']);
      },
      error => {
        console.log(error);
      }
    )
```

글쓰기

appRoutes.js

```
router.post('/create', (req, res, next) => {
    let number = new Date();
    number = Date.now();
    const newBoard = new Board({
        name: req.body.name,
        title: req.body.title,
        text: req.body.text,
        date: number
    });
    newBoard.save((err, board) => {
        if (err)
            res.status(500).json({ errmsg: err });
        res.status(200).json({ msg: board });
    });


});
```

Board.service.ts

```
createBoard(board: Board) {
  return this.http.post(this.baseUri + 'create', board, {
  headers: this.headers });
}
```

Read

List.component.ts

```
ngOnInit() {
  this.readBoards();
}

readBoards() {
  this._boardService.readBoards().subscribe(
    data => {
      this.boards = data['msg'];
    },
    error => {
      console.log(error);
    }
  )
}
```

Board.component.ts

```
ngOnInit() {
  this.readBoards(this.board);
}
readBoards(board) {
  const id = this.route.snapshot.params['id'];
  this._boardService.readOneBoard(id).subscribe(
    data => {
      this.board = data['msg'];
    },
    error => {
      console.log(error);
    }
  )
}
```

Board.service.ts

```
readBoards() {
  return this.http.get(this.baseUri + 'read', { headers:
  this.headers });
}
```

appRoutes.js

```
router.get('/read', (req, res, next) => {
    Board.find({}, (err, boards) => {
        if (err)
            res.status(500).json({ errmsg: err });
        res.status(200).json({ msg: boards });
    }).sort({ date: -1 });
});
```

Board.service.ts

```
readOneBoard(id: string): Observable<any> {
  return this.http.get(this.baseUri + 'board/' + id, {
  headers: this.headers });
}
```

appRoutes.js

```
router.get('/board/:id', (req, res, next) => {
    Board.findOne({ _id: req.params.id }, (err, board) => {
        if (err)
            res.status(500).json({ errmsg: err });
        res.status(200).json({ msg: board });
    });
});
```

Update

Create-update.component.ts

```
ngOnInit() {
  this.board = this.boardService.getter();
}
```

Board.service.ts

```
getter() {
  return this.board;
}
```

수정

Create-update.component.ts

```
this.userString = localStorage.getItem('user');
    this.user = JSON.parse(this.userString);
    this.username = this.user.username;
    this.board.name = this.username;
this.boardService.updateBoard(this.board).subscribe(
  data => {

    console.log(data);
    this.router.navigate(['/list']);
  },
  error => {
    console.log(error);
  }
)
} else if(this.board.name == "unknown") {
  this.boardService.updateBoard(this.board).subscribe(
    data => {
      console.log(data);
      this.router.navigate(['/list']);
    },
    error => {
      console.log(error);
    }
  )
} else {
  this.flashMessage.show('잘못된 접근입니다.', {
  cssClass: 'alert-danger', timeout: 5000 });
}
```

Board.service.ts

```
updateBoard(board: Board) {
  return this.http.put(this.baseUri + 'update', board, {
  headers: this.headers });
}
```

appRoutes.js

```
router.put('/update', (req, res, next) => {
    let number = new Date();
    number = Date.now();
    Board.findById(req.body._id, (err, board) => {
        if (err)
            res.status(500).json({ errmsg: err });
        board.name = req.body.name;
        board.title = req.body.title;
        board.text = req.body.text;
        board.date = number;
        board.save((err, board) => {
            if (err)
                res.status(500).json({ errmsg: err });
            res.status(200).json({ msg: board });
        });
    });
});
```

Delete

Board.component.ts

```
doDelete(board) {
  if(this.checkLoggedIn()) {
    const id = this.route.snapshot.params['id'];
    this._boardService.deleteBoards(id).subscribe(
      data => {
        this.router.navigate(['/list']);
      },
      error => {
        console.log(error);
      }
    )
  } else {
    this.flashMessage.show('잘못된 접근입니다.', { cssClass:
    'alert-danger', timeout: 5000 });
  }

}
```

삭제

Board.service.ts

```
deleteBoards(id: string) {
  return this.http.delete(this.baseUri + 'delete/' + id, {
  headers: this.headers });
}
```

appRoutes.js

```
router.delete('/delete/:id', (req, res, next) => {
    Board.findOneAndRemove({ _id: req.params.id }, (err, board)
     => {
        if (err)
            res.status(500).json({ errmsg: err });
        res.status(200).json({ msg: board });
        console.log(req.params.id);

    });
});
```

Search

Navbar.component.ts

```
ngOnInit() {
  this.board = this._boardService.getter();
  this.readBoards(this.boards);
}

readBoards(board) {
  const title = this.route.snapshot.params['title'];
  this._boardService.readBoard(title).subscribe(
    data => {
      this.boards = data['msg'];
    },
    error => {
      console.log(error);
    }
  )
}
```

Board.service.ts

```
readBoard(title: string) {
  return this.http.get(this.baseUri + 'search/' + title, {
  headers: this.headers });
}
```

appRoutes.js

```
router.get('/search/:title', (req, res, next) => {
    Board.find({ title: req.params.title }, (err, boards) => {
        if (err)
            res.status(500).json({ errmsg: err });
        res.status(200).json({ msg: boards });
    }).sort({ date: -1 });

});
```

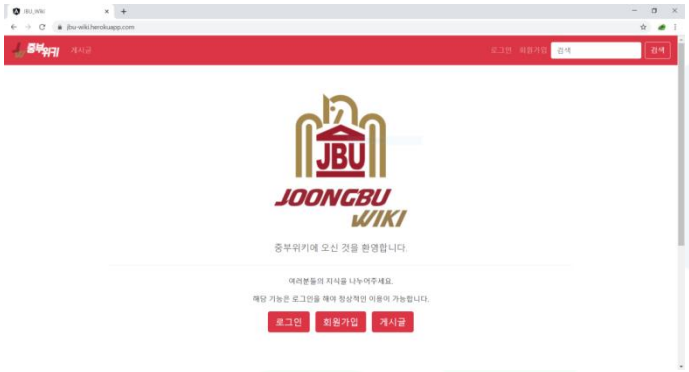
Search.component.ts

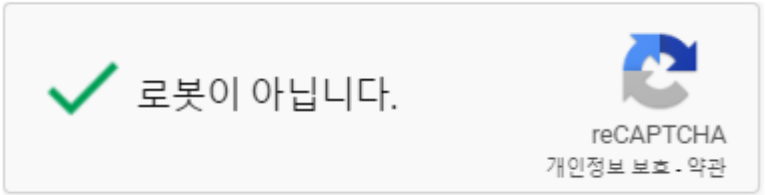
```
ngOnInit() {
  this.board = this._boardService.getter();
  this.readBoards(this.boards);
}

readBoards(board) {
  const title = this.route.snapshot.params['title'];
  this._boardService.readBoard(title).subscribe(
    data => {
      this.boards = data['msg'];
    },
    error => {
      console.log(error);
    }
  )
}
```

결과 분석

- 기존에 목표로한 위키 구현 성공
- 리캡차 기능 구현 성공
- 검색기능 외 CRUD기능 구현 성공
- 보안 부분 미흡
- https://jbu-wiki.herokuapp.com/





Q & A

THANK YOU