웹 어플리케이션 보안

SNS

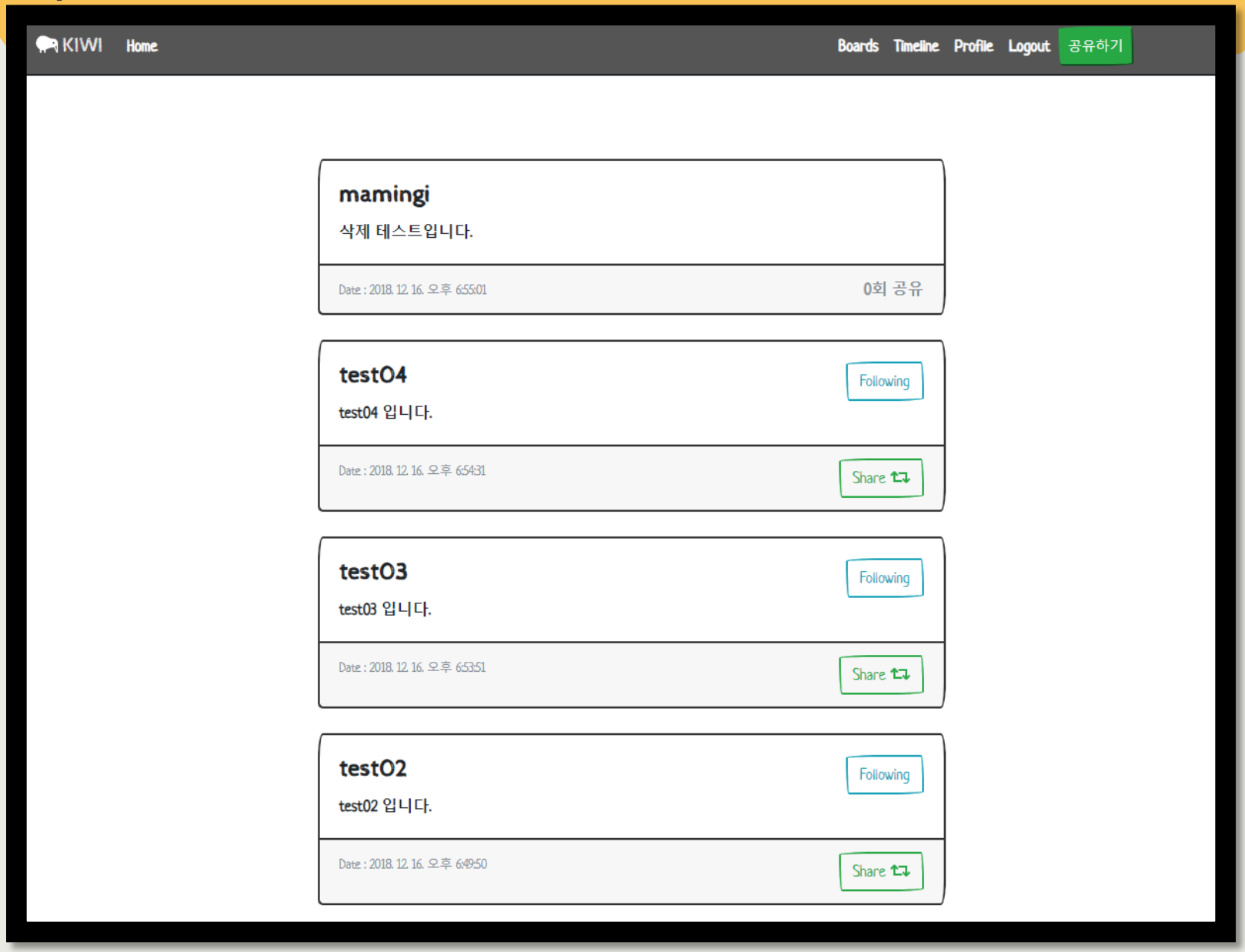
91416129
마민기

웹 어플리케이션 보안

목차

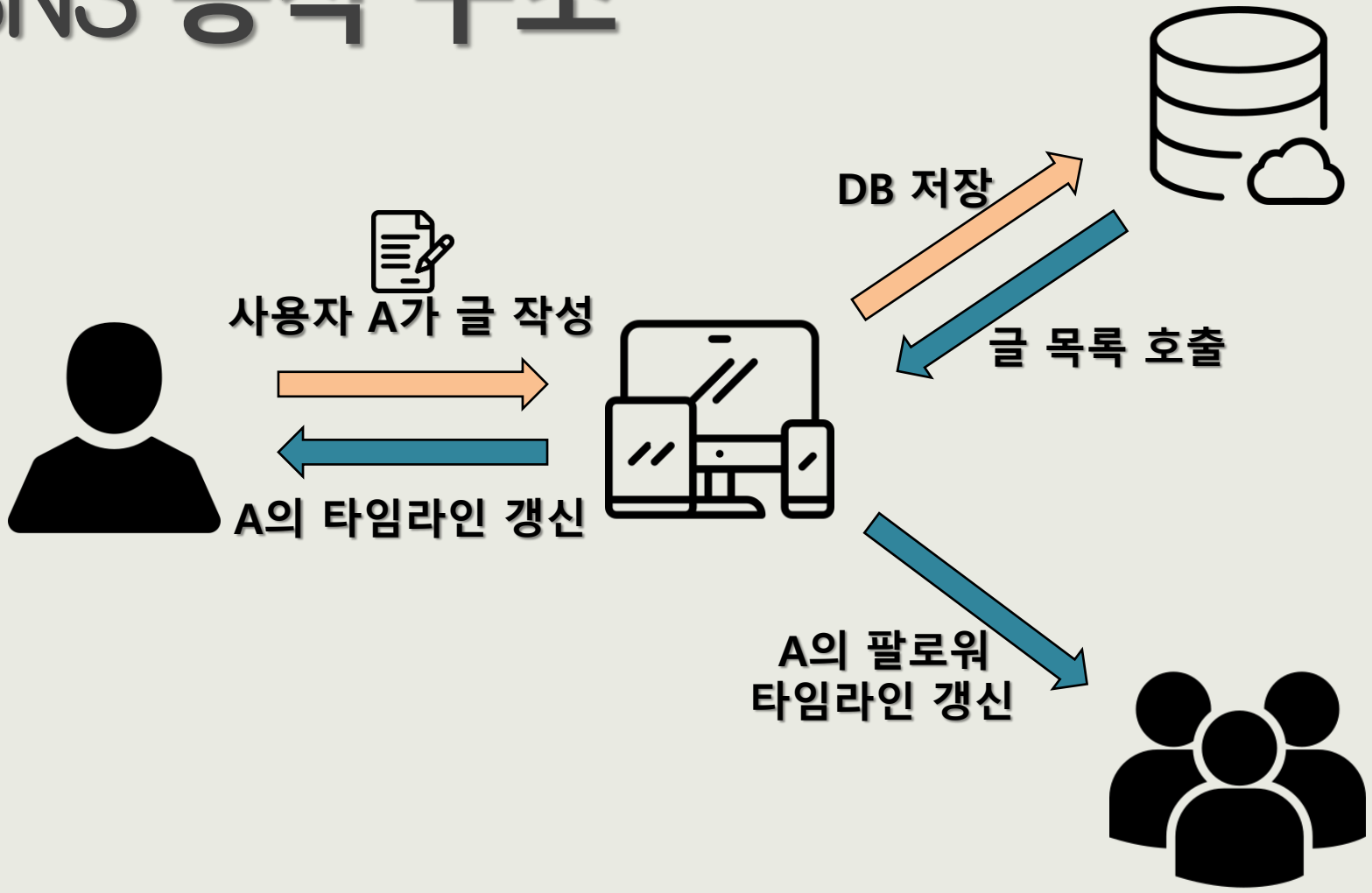
1. 소개

D'breed


KIWI
Home
Boards
Timeline
Profile
Logout
공유하기
mamingi
삭제 테스트입니다.
Date : 2018. 12. 16. 오후 6:55:01
0회 공유
test04
Following
test04 입니다.
Date : 2018. 12. 16. 오후 6:54:31
Share
test03
Following
test03 입니다.
Date : 2018. 12. 16. 오후 6:53:51
Share
test02
Following
test02 입니다.
Date : 2018. 12. 16. 오후 6:49:50
Share

Dbreed

SNS 동작 구조



Dbreed

로그인

난수화 토큰 인증을 사용한 인증 유지 기능

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

  if(!this.validateService.validateLogin(user)) {
    this.flashMessage.show('모든 필드를 입력하세요', { cssClass:
    'alert-danger', timeout: 3000 });
    return false;
  }

  this.authService.authenticateUser(user).subscribe(data => {
    if (data.success) {
      this.authService.storeUserData(data.ptoken, data.stoken
      data.user);
      this.flashMessage.show('로그인되었습니다',
        { cssClass: 'alert-success', timeout: 5000 });
```

```
storeUserData(ptoken, stoken, user){
  localStorage.setItem('id_ptoken', ptoken);
  localStorage.setItem('id_stoken', stoken);
  localStorage.setItem('user', JSON.stringify(user));
  this.pToken = ptoken;
  this.sToken = stoken;
  this.user = user;
```

```
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: "사용자 정보가 일치하지 않습니다.
      "});
    }
    User.comparePassword(password, user.password, (err, isMatch)=>{
      if(err) throw err;
      if(isMatch) {
        const ptoken = 'JWT '+jwt.sign({data: user}, config.secret, {
          expiresIn: 604800
        });
        const stoken = 'JWT '+jwt.sign({data: ptoken}, config.secret, {
          noTimestamp: true
        });
        res.json({
          success: true,
          ptoken: ptoken,
          stoken: stoken,
          user: {
            id: user._id,
            name: user.name,
            username: user.username,
            address: user.address,
            email: user.email
          }
        });
```

글쓰기

Bootstrap Modal을 사용한 글쓰기

글쓰기

1. 스키마 생성

```
const BoardsSchema = mongoose.Schema({
  writer: {
    type: String
  },
  contents: {
    type: String,
    required: true
  },
  date: {
    type: String
  },
  shared: {
    type: String
  },
  sharedcount: {
    type: Number,
    default: 0
  },
```

2. 작성한 글을 검증, 날짜 생성

```
onWriteSubmit() {

  this.ndate = new Date().toLocaleString('ko-KR', {
    timeZone: 'Asia/Seoul'
  });

  const board = { //board객체 선언
    writer: this.user,
    contents: this.contents,
    date: this.ndate
  }

  //모든 필드가 입력되었는지 확인
  if(!this.validateService.validateWrite(board)) {
    this.flashMessage.show('모든 필드를 입력하세요', { cssClass:
    'alert-danger', timeout: 3000 });
    return false;
  }

  this.authService.writeBoard(board).subscribe(data=>{
    if(data.success) {
      this.flashMessage.show('글 쓰기 성공.', { cssClass:
      'alert-success', timeout: 3000 });
      location.reload();
    } else {
      this.flashMessage.show('글 쓰기 실패.', { cssClass: 'alert-
      danger', timeout: 3000 });
    }
  });

}
```

3. DB에 글을 저장

```
router.post('/write', (req, res, next)=>{

  let newBoard = new Boards({
      writer: req.body.writer,
      contents: req.body.contents,
      date: req.body.date
  });

  Boards.addBoard(newBoard, (err, board)=>{
    if(err) {
      res.json({success: false, msg: '글 쓰기 실패.'});
    } else {
      res.json({success: true, msg:'글 쓰기 성공.'});
    }
  });
});
```

```
module.exports.addBoard = function(newBoard, callback){
  newBoard.save(callback);
```

글 출력

```
this.authService.getBoard().subscribe(post => {
  this.board = post.board;
}, err => {
  console.log(err);
  return false;
});
```

```
getBoard() {
  let headers = new Headers();
  headers.append('Content-Type', 'application/json');
  let ep = this.prepEndpoint('contents/boards');
  return this.http.post(ep, { headers: headers })
  .pipe(map(res => res.json()));
```

```
router.post('/boards', (req, res, next)=>{
  Boards.find({}).sort({date:-1}).exec(function(err, board){
  // db에서 최신 날짜부터 글들을 정렬해서 가져옴
    if(err) throw err;
    res.json({board: board});
  });
});
```

ngFor문을 사용, 모든 글을 출력

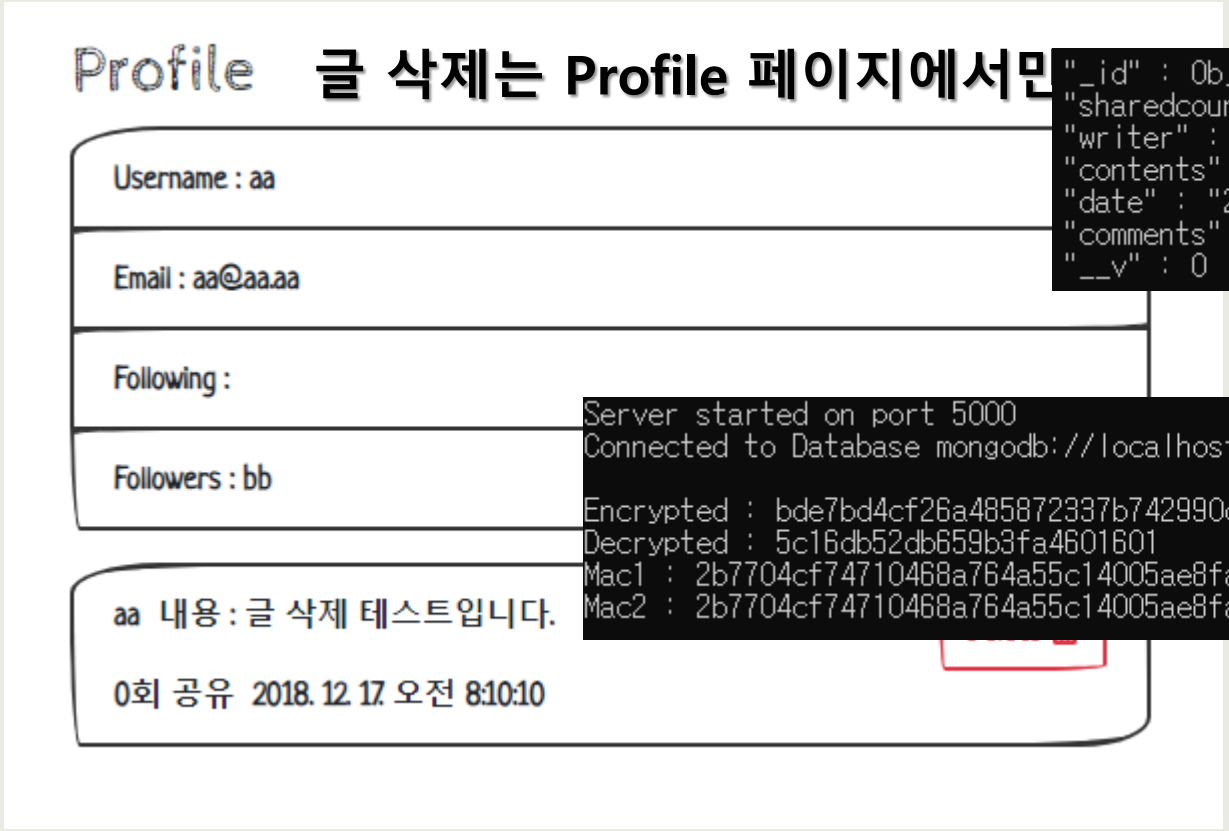
```
<div class="card-deck" *ngFor="let i of board">
 <div class="card">
   <div class="card-body">
     <button *ngIf = "!(i.writer==user)" class="btn btn-
     outline-info float-right" type="button"
     value="Following"
     (click)="onFollowingSubmit(i.writer)">Following</
     button>


     <h4 class="card-title"><strong>{{i.writer}}</strong></
     h4>
     <p class="card-text">{{i.contents}}</p>
   </div>
   <div class="card-footer">
     <small class="text-muted">Date : {{i.date}}</small>
     <a *ngIf="user==i.writer" class="text-muted float-
     right">{{i.sharedcount}}회 공유</a>
     <button *ngIf="!(i.writer==user)" class="btn btn-
     outline-success float-right" type="button"
     value="Share" (click)="onSharedSubmit(i._id)">Share
     <a *ngIf="!i.sharedcount==0">{{i.sharedcount}} </a>
     <i class="fas fa-retweet"></i></button>
```

글 삭제

난수화 토큰 인증을 사용한 일회용 비밀키를 생성

이를 이용한 메시지 암호화, 메시지 인증을 적용

서버는 복호화, 복호화된 메시지를 검증



글 삭제

```
<button *ngIf="i.writer==user.username" class="btn btn-
outline-danger float-right" type="button" value="Delete"
(click)="onDeleteSubmit(i. id)">Delete <i class="fas fa-
trash-alt"></i></button>
```

```
onDeleteSubmit(delid) {
  this.currT = this.authServ.ce.getCurrTime();
  this.key = this.authServic.getKey(this.currT, this. stoken);
  this.mac = this.authServic.getMac(delid, this.key);
  const encrypted = this.aut Service.getEncrypt(delid,
  this.key);

  this.authService.deleteEncMacMessage(encrypted, this.currT,
  this.ptoken, this.mac).subscribe(data => {
    if(data.success) {
      this.flashMessage.show('삭제 성공', { cssClass: 'alert-
      success', timeout: 3000 });
      location.reload();
    } else {
      this.flashMessage.show('삭제 실패. '+data.msg, {
      cssClass: 'alert-danger', timeout: 3000 });
    }
  });
```

```
getKey(currT, stoken){
  var md = forge.md.sha256.create();
  md.update(currT+stoken);
  return md.digest().toHex();
}
```

메시지 인증코드 생성, 암호화

```
getMac(plaintext, key) {
  var md = forge.md.sha256.create();
  md.update(plaintext+key);
  return md.digest().toHex();
}

getEncrypt(plaintext, key) {
  var key1 = forge.util.hexToBytes(key);
  var cipher = forge.cipher.createCipher('AES-ECB', key1);
  cipher.start();
  cipher.update(forge.util.createBuffer(plaintext, 'binary'));
  cipher.finish();
  var ciphertext = cipher.output;
  return forge.util.bytesToHex(ciphertext);
```

```
deleteEncMacMessage(encrypted, currT, ptoken, mac) {
  let headers = new Headers();
  headers.append('Authorization', ptoken);
  headers.append('Ctime', currT);
  headers.append('Enc', encrypted);
  headers.append('Mac', mac);
  headers.append('Content-Type','application/json');

  let ep = this.prepEndpoint('contents/delete');
  return this.http.get(ep, {headers: headers})
  .pipe(map(res => res.json()));
```

글 삭제

```
router.get('/delete', (req, res, next)=>{

  const ptoken = req.headers.authorization;
  const currT = req.headers.ctime;
  const encrypted = req.headers.enc;
  const mac1 = req.headers.mac;
  const stoken = 'JWT '+jwt.sign({data: ptoken}, config.secret, {
    noTimestamp: true
  });

  var md = forge.md.sha256.create();
  md.update(currT+stoken);
  var auth = forge.util.bytesToHex(md.digest());
  var key = forge.util.hexToBytes(auth);
  var encrypted1 = forge.util.hexToBytes(encrypted);
  var decipher = forge.cipher.createDecipher('AES-ECB', key);
  decipher.start();
  decipher.update(forge.util.createBuffer(encrypted1, 'binary'));
  decipher.finish();
  var decrypted = decipher.output;
```

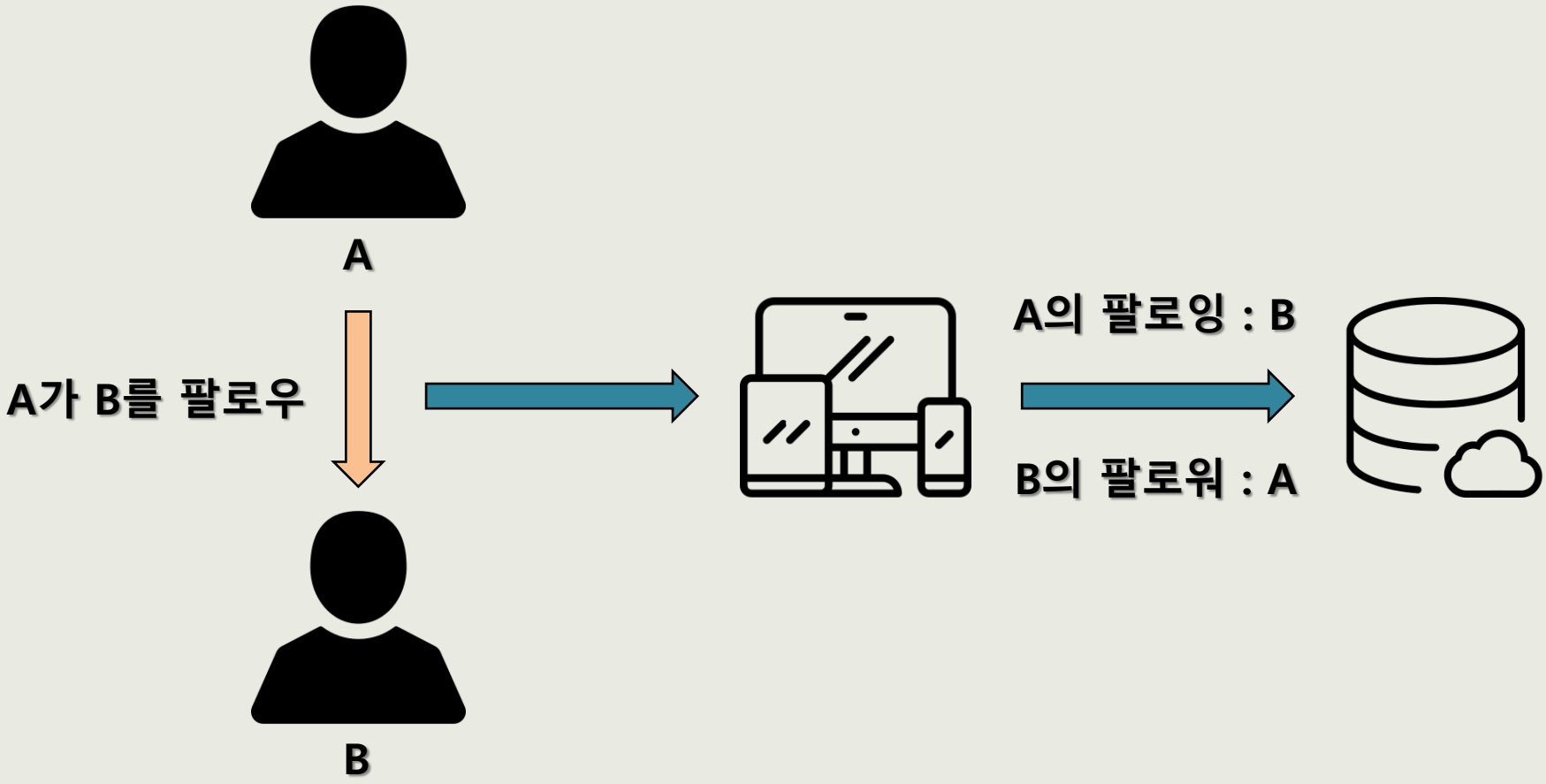
```
var md1 = forge.md.sha256.create();
md1.update(decrypted+auth);
var mac2 = forge.util.bytesToHex(md1.digest());

if (mac1 == mac2) {
  let dBoard = new Boards({
    _id: decrypted
  });

  Boards.delBoard(dBoard, (err, board)=>{
    if(err) {
      res.json({success: false, msg: '글 삭제 실패'});
    } else {
      res.json({success: true, msg:'글 삭제 성공'});
    }
  });
} else {
  res.json({success: false, msg: '삭제 인증 실패'})
}
```

```
module.exports.delBoard = function(board, callback){
  Boards.deleteOne({ _id : board._id }, callback);
```

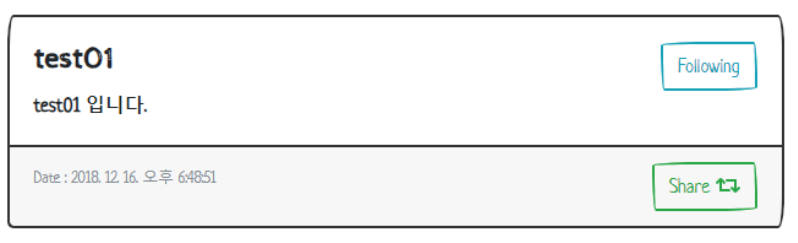
1. 암호화된 메시지를 복호화 진행

2. 복호화 후, 메시지 인증코드 생성
3. 두 메시지 인증코드를 검증

팔로우



팔로우




```
<button *ngIf = "!(i.writer==user)" class="btn btn-
outline-info float-right" type="button"
value="Following"
(click)="onFollowingSubmit(i.writer)">Following</
button>
```

로그인되지 않은 사용자,
자기자신은 팔로우 할 수 없다.

```
onFollowingSubmit(w) {

  if(this.authService.loggedIn() == false) {
    this.router.navigate(['/login']);
    this.flashMessage.show('로그인 후 사용할 수 있습니다.', {
    cssClass: 'alert-danger', timeout: 3000 });
    return false;
  }

  const follow = {
    username: this.user,
    following: w
  }

  if(follow.username == follow.following) {
    this.flashMessage.show('팔로우 실패. SelfFollow', {
    cssClass: 'alert-danger', timeout: 3000 });
    return false;
  }

  this.authService.followingUser(follow).subscribe(data => {
    if(data.success) {
      this.flashMessage.show('팔로우 성공', { cssClass: 'alert-
      success', timeout: 3000 });
      this.router.navigate(['/profile']);
    } else {
      this.flashMessage.show('팔로우 실패. ' +data.msg, {
      cssClass: 'alert-danger', timeout: 3000 });
```

Dbreed

팔로우

A가 B를 팔로우 시
A의 folloing에 B가 추가
B의 followers에 A가 추가

```
const UserSchema = mongoose.Schema({
  name: {
    type: String
  },
  email: {
    type: String,
    required: true //사용자에게 입력을
  },
  address: {
    type: String
  },
  username: {
    type: String,
    required: true
  },
  password: {
    type: String,
    required: true
  },
  following: { //내가 구독하는 사람
    type: String
  },
  followers: { //나를 구독하는 사람
    type: String
```

```
module.exports.addfollowingUser = function(newfollowingUser,
callback){
  User.updateOne({ username : newfollowingUser.username }, {
  $push : { following : newfollowingUser.following }}, callback);
}

module.exports.addfollowerUser = function(newfollowerUser,
callback){
  User.updateOne({ username : newfollowerUser.username }, { $push
  : { followers : newfollowerUser.followers }}, callback);
```

```
router.post('/following', (req, res, next)=>{
  let newfollowingUser = new User({ //let은 변경 가능한 변수.
  const는 상수
    username: req.body.username,
    following: req.body.following
  });

  let newfollowerUser = new User({
    username: req.body.following,
    followers: req.body.username
  });

  User.getfollowingByUsername(newfollowingUser.username,
  newfollowingUser.following, (err, user)=>{
    if(user) {
      res.json({success: false, msg: "이미 팔로우 중인 유저입니다.
      "});
    }
    else {
      User.addfollowingUser(newfollowingUser, (err, user)=>{
        if(err) {
          res.json({success: false, msg: '팔로잉 실패. Err 1 '});
        } else {
          User.addfollowerUser(newfollowerUser, (err, user)=>{
            if(err) {
              res.json({success: false, msg: '팔로잉 실패. Err 2
              '});
            } else {
              res.json({success: true, msg:'팔로잉 성공. '});
```

팔로우 취소

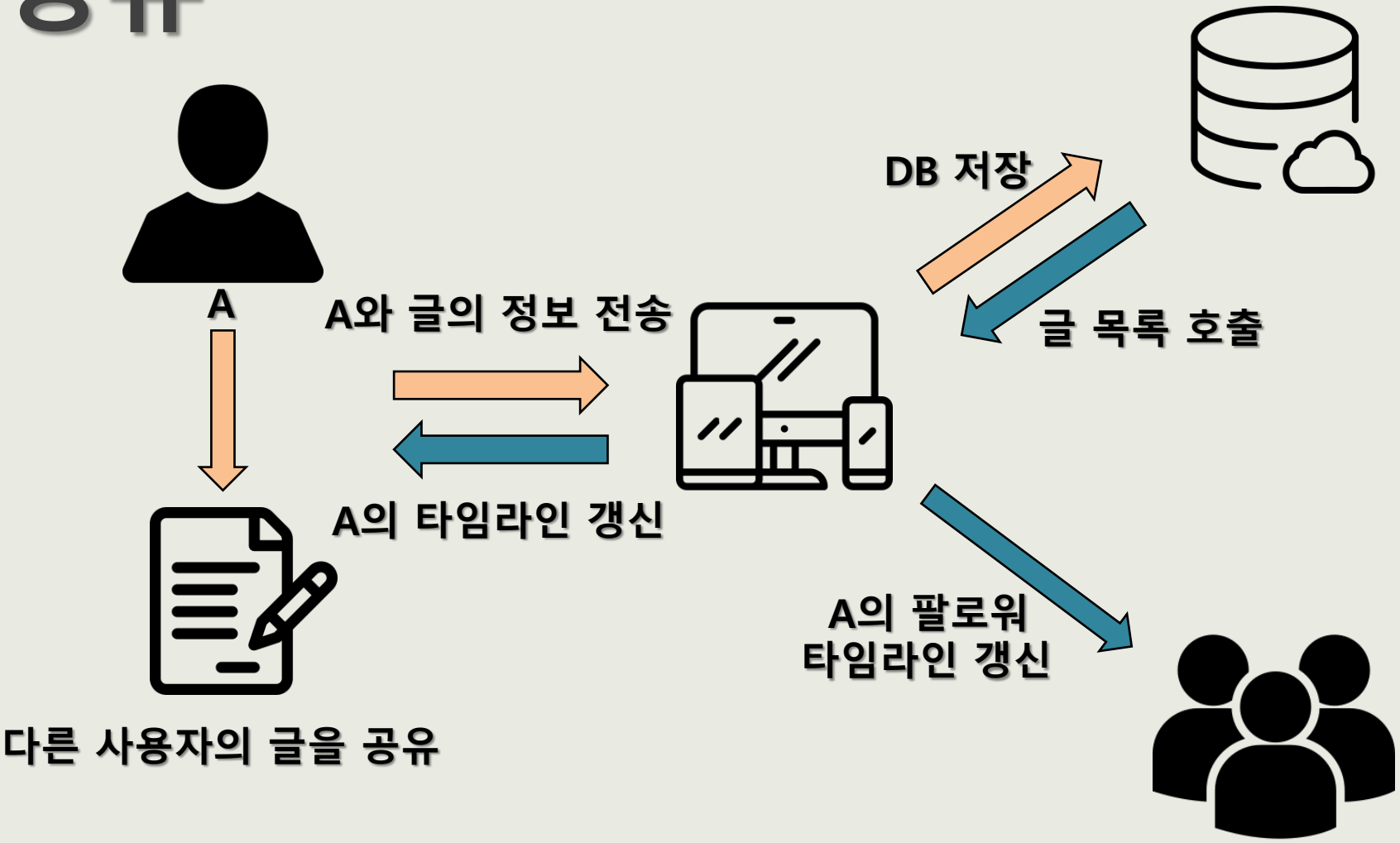
```
<button class="dropdown-item" type="button"
(click)="onUnfollowingSubmit(user.username,
i.writer)">Unfollowing</button>
```

```
onUnfollowingSubmit(usr, un) {
  const unfollow = {
    username: usr,
    following: un
  }

  if(unfollow.username == unfollow.following) {
    this.flashMessage.show('언팔로우 실패. SelfUnfollow', {
    cssClass: 'alert-danger', timeout: 3000 });
    return false;
  }

  this.authServic.unfollowingUser(unfollow).subscribe(data =>
  {
    if(data.success) {
      this.flashMessage.show('언팔로우 성공', { cssClass:
      'alert-success', timeout: 3000 });
      location.reload();
      this.router.navigate(['/profile']);
    } else {
      this.flashMessage.show('언팔로우 실패. ' +data.msg, {
      cssClass: 'alert-danger', timeout: 3000 });
```

```
router.post('/unfollowing', (req, res, next)=>{
  let unfollowingUser = new User({
    username: req.body.username,
    following: req.body.following
  });

  let unfollowerUser = new User({
    username: req.body.following,
    followers: req.body.username
  });

  User.getfollowingByUsername(unfollowingUser.username,
  unfollowingUser.following, (err, user)=>{
    if(user) {
      User.delfollowingUser(unfollowingUser, (err, user)=>{
        if(err) {
          res.json({success: false, msg: '언팔로잉 실패. Err 1
          '});
        } else {
          User.delfollowerUser(unfollowerUser, (err, user)=>{
            if(err) {
              res.json({success: false, msg: '언팔로잉 실패. Err
              2 '});
            } else {
              res.json({success: true, msg:'언팔로잉 성공. '});
```

```
module.exports.delfollowingUser = function(unfollowingUser,
callback){
  User.updateOne({ username : unfollowingUser.username }, { $pull
  : { following : unfollowingUser.following }}, callback);
}

module.exports.delfollowerUser = function(unfollowerUser,
callback){
  User.updateOne({ username : unfollowerUser.username }, { $pull
  : { followers : unfollowerUser.followers }}, callback);
```

공유



공유

```
<button *ngIf="!(i.writer==user)" class="btn btn-
outline-success float-right" type="button"
value="Share" (click)="onSharedSubmit(i. id)">Share
<a *ngIf="!i.sharedcount==0">{{i.sharedcount}} </a>
<i class="fas fa-retweet"></i></button>
```

```
onSharedSubmit(sh) {

  if(this.authService.loggedIn() == false) {
    this.router.navigate(['/login']);
    this.flashMessage.show('로그인 후 사용할 수 있습니다.'
    cssClass: 'alert-danger', timeout: 3000 });
    return false;
  }

  const share = {
    _id: sh,
    shared: this.user
  }

  this.authService.shareBoard(share).subscribe(data => {
    if(data.success) {
      this.flashMessage.show('공유 성공. '+data.msg, {
      cssClass: 'alert-success', timeout: 3000 });
      location.reload();
      //this.router.navigate(['/boards']);
    } else {
      this.flashMessage.show('공유 실패. '+data.msg, {
      cssClass: 'alert-danger', timeout: 3000 });
```

```
router.post('/share', (req, res, next)=>{
  let sBoard = new Boards({
    _id: req.body._id,
    shared: req.body.shared
  });

  Boards.getsharedbyBoardid(sBoard._id, sBoard.shared (err,
  board)=>{
    if(board) {
      res.json({success: false, msg: "이미 공유된 글입니다. "});
    } else {
      Boards.sharedBoard(sBoard, (err, board)=>{
        if(err) {
          res.json({success: false, msg: '공유 실패. Crr1. '});
        } else {
          Boards.updateOne({ _id : sBoard._id }, { $inc : {
          sharedcount : +1 }}, (err, user) => {
            if(err) {
              res.json({success: false, msg: '공유 실패. Err2.
              '});
            }
            else {
              res.json({success: true, msg:'공유 성공. '});
```

```
module.exports.getsharedbyBoardid = function(boardid, shareduser,
callback){
  Boards.findOne({ _id : boardid, shared : shareduser },
  callback);
}

module.exports.sharedBoard = function(board, callback){
  Boards.updateOne({ _id : board._id }, { $push : { shared :
  board.shared }}, callback);
```

공유 취소

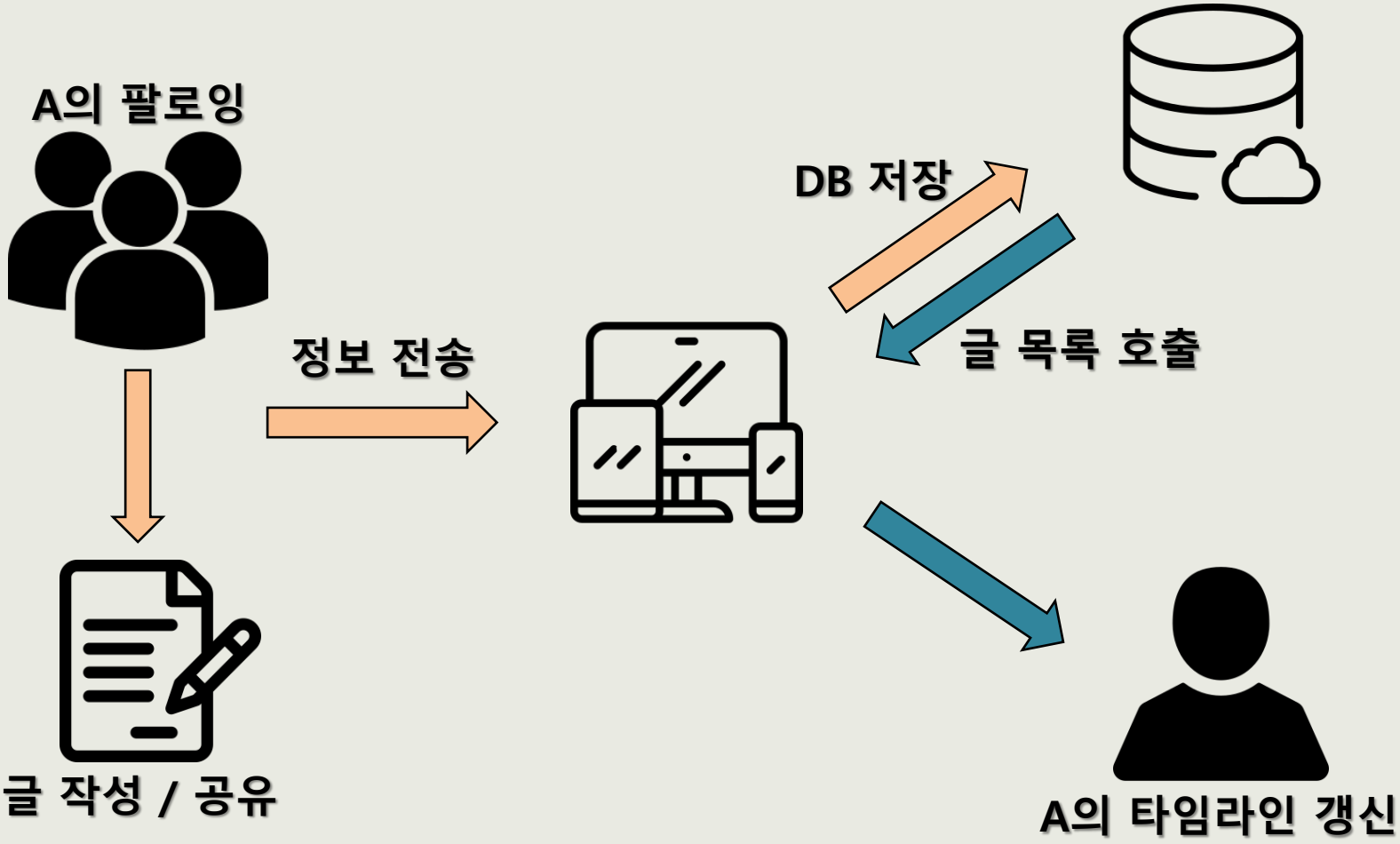
```
<button class="dropdown-item" type="button"
(click)="onUnsharedSubmit(i._id,
user.username)">Unshare <i class="fas fa-retweet"></
i></button>
```

```
onUnsharedSubmit(unid, usr) {
  const unshare = {
    _id: unid,
    shared: usr
  }

  this.authService.unshareBoard(unshare).subscribe(data => {
    if(data.success) {
      this.flashMessage.show('공유취소 성공. '+data.msg, {
      cssClass: 'alert-success', timeout: 3000 });
      location.reload();
      //this.router.navigate(['/boards']);
    } else {
      this.flashMessage.show('공유취소 실패. '+data.msg, {
      cssClass: 'alert-danger', timeout: 3000 });
```

```
router.post('/unshare', (req, res, next)=>{
  let usBoard = new Boards({
    _id: req.body._id,
    shared: req.body.shared
  });

  Boards.getsharedbyBoardid(usBoard._id, usBoard.shared, (err,
  board)=>{
    if(board) {
      Boards.unsharedBoard(usBoard, (err, board)=>{
        if(err) {
          res.json({success: false, msg: '공유취소 실패. Err1.
          '});
        } else {
          Boards.updateOne({ _id : usBoard._id }, { $inc : {
          sharedcount : -1 }}, (err, user) => {
            if(err) {
              res.json({success: false, msg: '공유취소 실패.
              Err2. '});
            }
            else {
              res.json({success: true, msg:'공유취소 성공. '});
```

```
module.exports.unsharedBoard = function(board, callback){
  Boards.updateOne({ _id : board._id }, { $pull : { shared :
  board.shared }}, callback);
```

타임라인



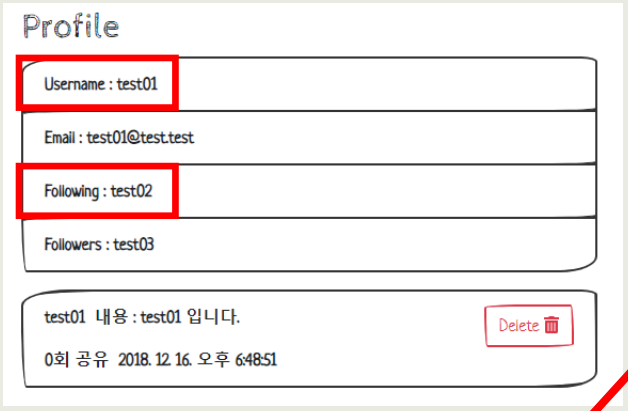
타임라인

```
ngOnInit() {

  this.timeline().subscribe(result => {
    this.user = result.user;
    const followtime = {
      username: result.user.username
    }
    this.authService.getTimeline(followtime).subscribe(post =>
    {
      this.board = post.board;
    }, err => {
      console.log(err);
      return false;
    });
  }, err => {
    console.log(err);
    this.authService.logout();
    this.router.navigate(['/login']);
    return false;
  });
}

timeline() {
  this.currT = this.authService.getCurrTime();
  this.ptoken = this.authService.getPubToken();
  this.stoken = this.authService.getSecToken();
  this.auth = this.authService.getKey(this.currT, this.stoken);
  return this.authService.getProfile(this.ptoken, this.currT,
  this.auth);
```

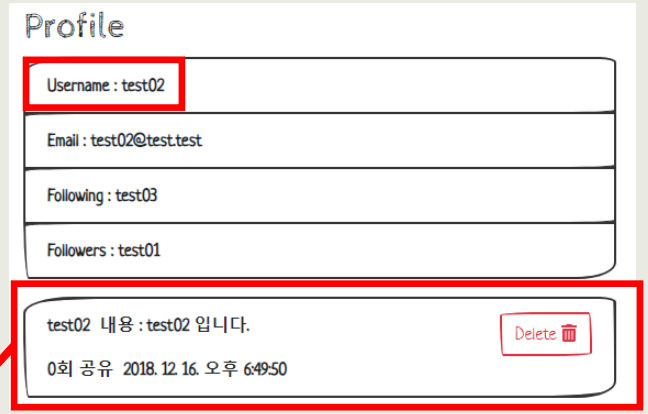
```
router.post('/timeline', (req, res, next) => {

  let newTimelineUser = new User({
    username: req.body.username
  });

  User.find( { followers : newTimelineUser.username }
  ).exec(function(err, user) {
    if (err) throw err;
    var uname = new Array();
    for(var i = 0; i<user.length; i++){
      uname[i] = user[i].username;
    }
    uname[i] = newTimelineUser.username;

    Boards.find({ $or : [{ writer : uname }, { shared : uname }]
    }).sort({date: -1}).exec(function(err, board) {
      if (err) throw err;
      res.json({board: board});
    });
  });
```

모든 사용자들 중에서 팔로워에 A가 있는 사용자의 목록을 받아서(A의 팔로잉과 동일) 배열을 생성.

타임라인 예시



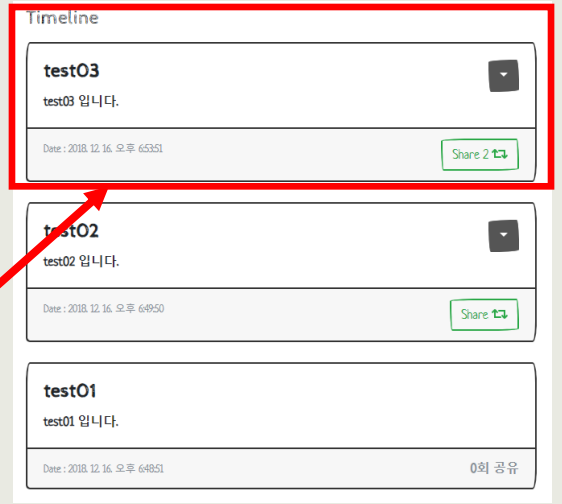
test01



test02

test01 타임라인

test02 타임라인



test01 타임라인

구현되지 못한 기능들

- 누군가가 공유한 글이 자신의 타임라인에 노출될 때 누가 공유한 글인지 표시하는 기능
- 이미지 업로드 기능
- 인증서
- 표시되는 글을 페이징 기능으로 나눠서 출력

Q&A

THANK YOU