PoiemaWeb

13.22 Angular JWT Authentication

Token 기반 인증

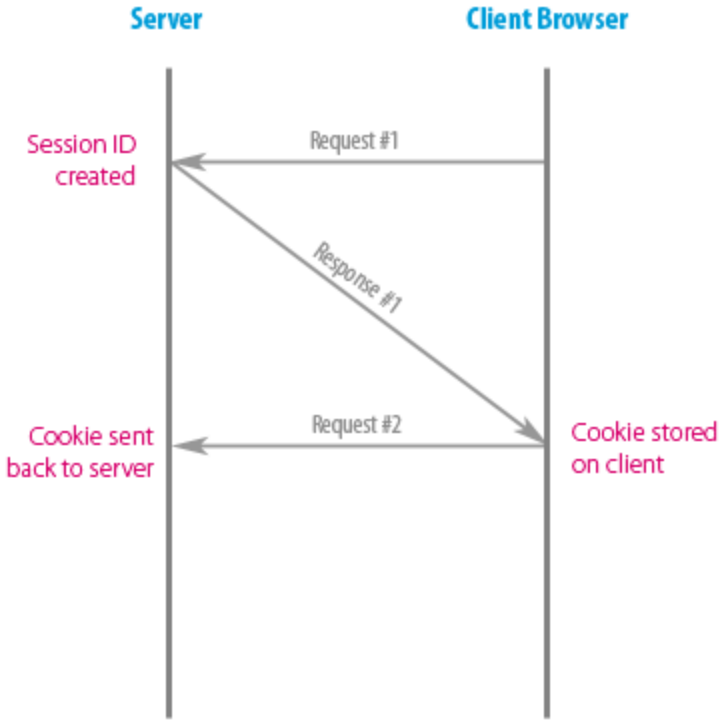
#1. JWT(JSON Web Token)이란?

http 프로토콜은 상태(state)를 유지하지 않는다. 이를 stateless protocol이라 한다.

HTTP 프로토콜은 요청(request)를 전송하고 응답(response)를 전송받은 시점에서 통신이 종료되며 어떠한 상태 정보도 남지 않는다. 즉, 특정 클라이언트에서 동일 서버에 반복하여 접속하여도 각각의 접속은 독립적인 트랜잭션으로 취급된다.

따라서 로그인 화면에서 아이디, 패스워드를 입력하여 사용자 인증 과정을 거친 이후에 재차 웹사이트에 접근하면 로그인 상태임을 인식(유지)할 수 없으므로 매번 사용자 인증 과정을 반복해야 하는 문제가 발생한다.

http 프로토콜의 상태 비유지(stateless) 문제를 보완하여 클라이언트와 서버 간의 논리적 연결을 위한 방법에는 Session과 Cookie 그리고 Cookie-Based Session 방식과 같은 stateful 서버가 있다.



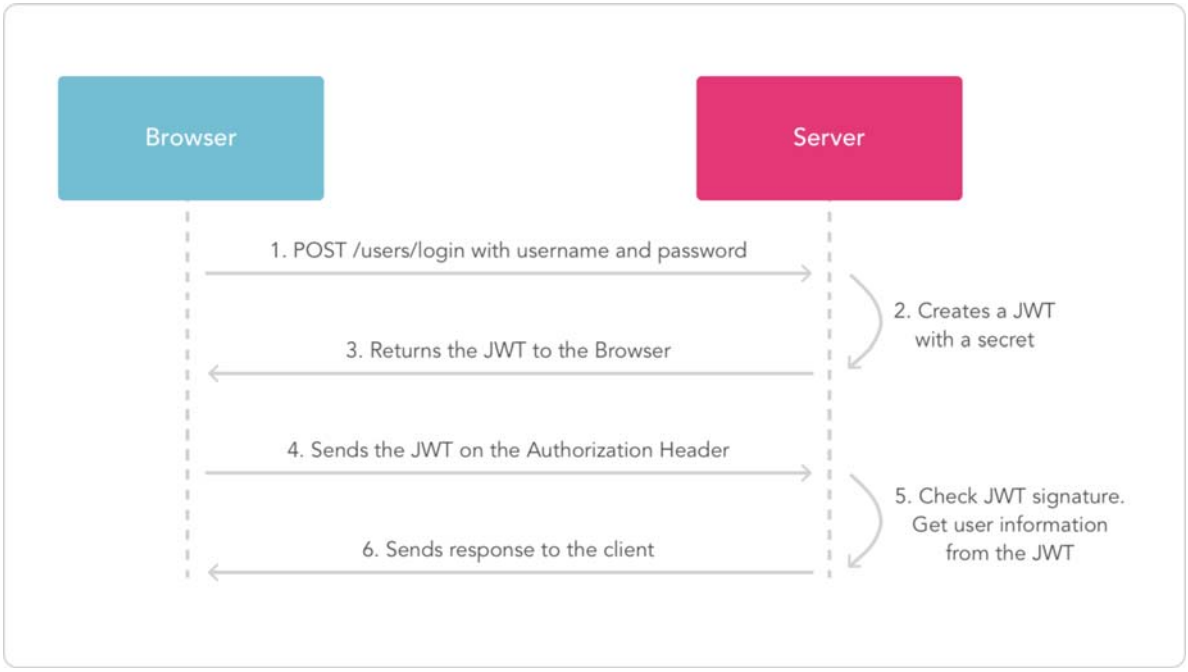
Cookie-Based Session

stateful 서버는 클라이언트로부터의 요청이 있을 때 마다 클라이언트의 상태를 유지한다. 예를 들어 사용자가 로그인 화면에서 아이디, 패스워드를 입력하여 사용자 인증을 요청하면 stateful 서버는 인증에 성공하였을 때 그 결과 즉 인증 상태 정보를 서버의 메모리 또는 데이터베이스에 유지하고 서비스 제공에 사용한다.

JWT(JSON Web Token) 기반 인증(토큰 기반 인증)은 상태를 서버에 유지하지 않는다. 즉, stateful 서버와 반대적 개념인 stateless 서버를 사용하며 상태 정보를 유지하지 않는다.

토큰 기반 인증은 일반적으로 아래의 방식으로 구현한다.

1. 클라이언트는 아이디와 패스워드로 사용자 인증을 요청한다.
2. 서버는 사용자 인증을 수행한다. 이때 인증에 성공하면 정상적으로 발급된 토큰임을 증명하는 signature를 갖는 토큰을 클라이언트에 발급한다.
3. 클라이언트는 토큰을 저장하고 서버 요청시 해당 토큰을 요청 헤더(Request header)에 담아 서버에 전달한다.
4. 서버는 토큰을 검증한 후, 요청에 응답한다.

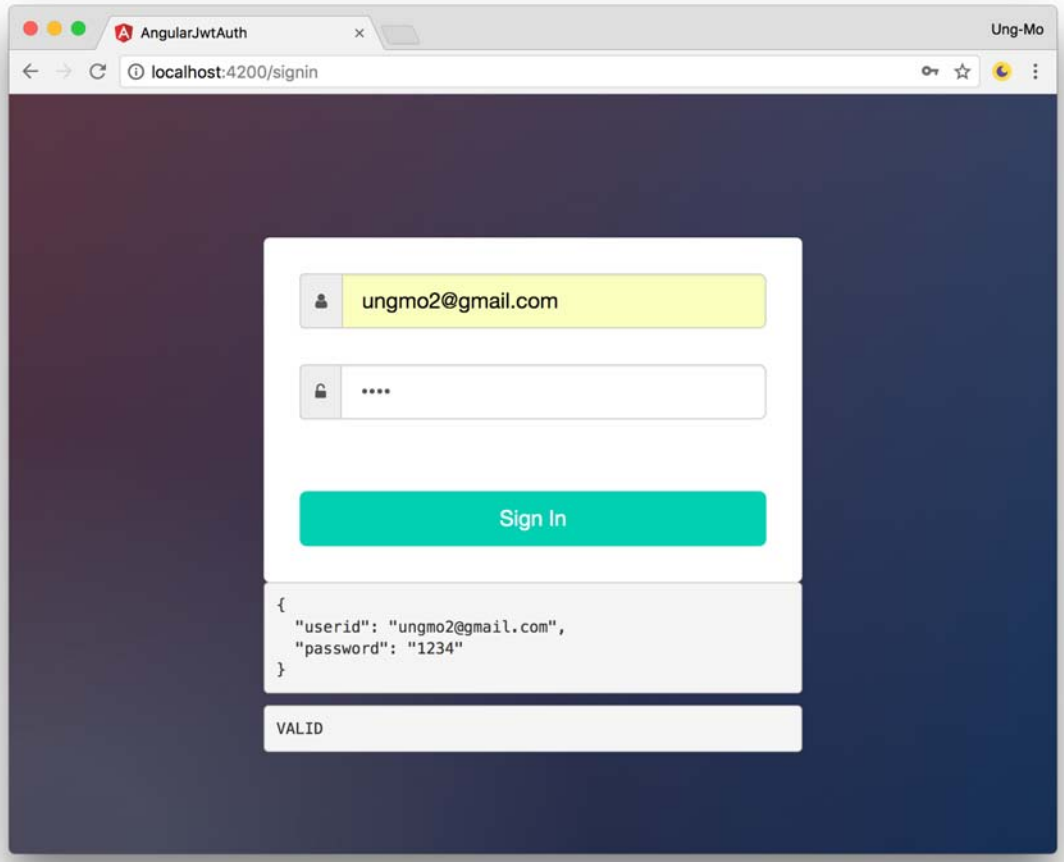


JWT(JSON Web Token) 기반 인증

토큰 기반 인증은 상태 정보를 유지하지 않기 때문에 서버 확장성(Scalability)이 좋아지고 클라이언트는 쿠키를 전달할 필요가 없으므로 보안성이 강화되며 다른 서버의 서비스에 정보를 공유할 수 있는 장점이 있다.

2. Angular JWT 인증

Angular에서 토큰 기반 인증 방식으로 간단한 로그인 애플리케이션을 구현해보자. 완성된 애플리케이션의 실행결과는 아래와 같다.



2.1 Backend

Backend는 Express와 MongDB를 사용하여 구현하였다. 전체 소스코드는 이곳에서 참조할 수 있다.

- MongDB 설치

BASH

```
$ git clone https://github.com/ungmo2/jwt-express.git server
$ cd server
$ npm install
```

파일 구성은 아래와 같다.

CODE

```
server/
 ├── lib/
```

```
|    └──── token.js
├──── middlewares/
|    └──── auth.js
├──── models/
|    └──── user.js
├──── routes/
|    ├──── auth.js
|    └──── users.js
├──── .env
├──── package.json
└──── server.js
```

환경변수를 관리하기 위해 .env 파일을 생성한다.

CODE

```
# 서버 포트
PORT=5500

# 패스워드 암호화 시크릿키
SECRET_KEY=<Your-secret-key>

# JWT 시크릿키
JWT_SECRET=<Your-jwt-secret-key>

# Database URI
MONGO_URI=<Your-MongoDB-URI>
```

2.2 Frontend

로그인 뷰(signin.component.ts)와 로그인 성공시 이동할 대시보드 뷰(dashboard.component.ts)으로 구성되어 있다. 처리의 흐름은 아래와 같다.

1. 로그인 뷰는 인증 서비스(auth.service.ts)를 사용하여 서버에 인증을 요청한다.
2. 서버 인증이 성공한 경우, 서버는 토큰을 발행하고 클라이언트로 토큰을 응답한다.

3. 서버 인증이 성공한 경우, 인증 서비스는 서버가 응답한 토큰을 로컬 스토리지에 저장하고 대시보드 뷰로 이동한다. 이때 가드(auth.guard.ts)를 사용하여 토큰을 검증한다.
4. 대시보드 컴포넌트는 사용자 서비스(users.service.ts)를 사용하여 서버에 사용자 정보를 요청한다. 이때 요청 헤더(Request Header)에 토큰을 담아 전송한다.
5. 대시보드 컴포넌트는 서버의 응답을 받아 뷰에 표시한다.

파일 구성은 아래와 같다. 전체 소스코드는 이곳에서 참조할 수 있다.

CODE

```
app/
├── dashboard/
│   └── dashboard.component.ts
├── guards/
│   └── auth.guard.ts
├── models/
│   ├── token.ts
│   └── user.ts
├── services/
│   ├── auth.service.ts
│   └── users.service.ts
├── signin/
│   └── signin.component.ts
├── app-routing.module.ts
├── app.component.ts
└── app.module.ts
```

추가로 설치할 의존성은 아래와 같다.

BASH

```
$ npm install font-awesome bootstrap angular2-jwt
```

font-awesome과 bootstrap을 적용하기 위해 angular.json을 아래와 같이 수정한다.

JSON

```
...
  "styles": [
```

```
      "../node_modules/font-awesome/css/font-awesome.min.css",
      "../node_modules/bootstrap/dist/css/bootstrap.min.css",
      "styles.css"
    ],
  ...
```

2.2.1 로그인 컴포넌트 (signin.component.ts)

로그인 컴포넌트는 사용자 아이디와 패스워드를 입력받는 폼이다. 리액티브 폼을 사용하여 유효성을 검증하고 인증 서비스로 서버에 사용자 인증을 요청한다. 인증 성공시에는 대시보드로 이동한다.

TYPESCRIPT

```
// signin/signin.component.ts
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl, Validators, FormBuilder } from '@angular/fo
rms';
import { Router } from '@angular/router';

import { User } from '../models/user';
import { AuthService } from '../services/auth.service';

@Component({
  selector: 'app-signin',
  template: `
    <div class="container">
      <div class="col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
        <form [formGroup]="signinForm" (ngSubmit)="signin()" novalidate>
          <div id="userid" class="input-group">
            <div class="input-group-addon">
              <i class="fa fa-user"></i>
            </div>
            <input formControlName="userid" type="text" class="form-control
 input-lg" name="userid" placeholder="email">
          </div>

          <div class="alert-box">
            <em *ngIf="userid.errors?.required && userid.touched" class="tex
```

```
t-danger">
              userid를 입력하세요!</em>
            <em *ngIf="userid.errors?.pattern && userid.touched" class="text
-danger">
              userid를 이메일 형식에 맞게 입력하세요!</em>
          </div>

          <div id="password" class="input-group">
            <div class="input-group-addon">
              <i class="fa fa-unlock-alt"></i>
            </div>
            <input formControlName="password" type="password" class="form-co
ntrol input-lg" name="password" placeholder="password">
          </div>

          <div class="alert-box">
            <em *ngIf="password.errors?.required && password.touched" class
="text-danger">
              password를 입력하세요!</em>
            <em *ngIf="password.errors?.pattern && password.touched" class
="text-danger">
              password는 영문 또는 숫자로 입력하세요!</em>
            <em *ngIf="password.errors?.minlength && password.touched" class
="text-danger">
              password는 4자리 이상으로 입력하세요!</em>
            <em *ngIf="password.errors?.maxlength && password.touched" class
="text-danger">
              password는 10자리 이하로 입력하세요!</em>
          </div>

          <div *ngIf="message" class="alert alert-danger">{{ message }}</div
>

          <button [disabled]="signinForm.invalid" type="submit" class="btn b
tn-lg btn-success btn-block">Sign In</button>
        </form>

        <pre>{{ signinForm.value | json }}</pre>
        <pre>{{ signinForm.status }}</pre>
      </div>
    </div>
```

```
    `,
    styles: [`...`]
  })
  export class SigninComponent implements OnInit {
    signinForm: FormGroup;
    message: string;

    constructor(
      private fb: FormBuilder,
      private auth: AuthService,
      private router: Router) { }

    ngOnInit() {
      this.signinForm = this.fb.group({
        userid: ['', [
          Validators.required,
          Validators.pattern(/^[0-9a-zA-Z]([-_\.]?[0-9a-zA-Z])*@[0-9a-zA-Z]([-
_\.]?[0-9a-zA-Z])*\.[a-zA-Z]{2,3}$/)
        ]],
        password: ['', [Validators.required,
          Validators.pattern(/[a-zA-Z0-9]/),
          Validators.minLength(4),
          Validators.maxLength(10)
        ]]
      });
    }

    signin() {
      console.log('[payload]', this.signinForm.value);
      this.auth.signin(this.signinForm.value)
        .subscribe(
          () => this.router.navigate(['dashboard']),
          ({error}) => {
            console.log(error.message);
            this.message = error.message;
          }
        );
    }

    get userid() {
      return this.signinForm.get('userid');
```

```
  }

  get password() {
    return this.signinForm.get('password');
  }
}
```

2.2.2 대시보드 컴포넌트 (dashboard.component.ts)

대시보드 컴포넌트는 사용자 인증 성공시 이동하는 화면으로 가드에 의해 보호된다. 즉, 사용자 인증 성공시 서버로 부터 전달받은 토큰의 유효성을 검증하여 대시보드로의 이동을 허가한다. 대시보드로 이동하면 사용자 서비스를 사용하여 서버로 부터 사용자 정보를 취득한다.

TYPESCRIPT

```
// dashboard/dashboard.component.ts
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { AuthService } from '../services/auth.service';
import { UserService } from '../services/user.service';
import { User } from '../models/user';

interface Header {
  title: string;
  subtitle: string;
}

@Component({
  selector: 'app-dashboard',
  template: `
    <div class="jumbotron">
      <div class="container">
        <h1>{{ header.title }}</h1>
        <p>{{ header.subtitle }}</p>
        <p>{{ userid }}</p>
        <a class="btn btn-default" (click)="signout()">Signout</a>
      </div>
    </div>
```

```
      <div class="container">
        <div class="row">
          <h2>Users infomation</h2>
          <div class="table-responsive">
            <table class="table table-bordred table-hover">
              <thead>
                <th>#</th>
                <th>ID</th>
                <th>Admin</th>
              </thead>
              <tbody>
                <tr *ngFor="let user of users; let i = index">
                  <td>{{ i }}</td>
                  <td>{{ user.userid }}</td>
                  <td>{{ user.admin }}</td>
                </tr>
              </tbody>
            </table>
          </div>
        </div>
      </div>
    `
  })
  export class DashboardComponent implements OnInit {
    header: Header;
    userid: string;
    users: User[];

    constructor(
      private route: ActivatedRoute,
      private router: Router,
      private authService: AuthService,
      private userService: UserService) { }

    ngOnInit() {
      // static data 취득
      this.header = this.route.snapshot.data as Header;
      // 토큰에서 사용자 아이디 취득
      this.userid = this.authService.getUserid();

      // 사용자 정보 취득
```

```
    this.userService.getUsers()
      .subscribe(users => this.users = users);
  }

  signout() {
    this.authService.signout();
    this.router.navigate(['signin']);
  }
}
```

2.2.3 인증 서비스 (auth.service.ts)

인증 서비스는 사용자 로그인과 로그아웃을 구현한 서비스이다. 로그인이 성공하면 서버로 부터 토큰을 전달받아서 로컬스토리지에 저장한다.

토큰의 유효성 검증 메소드 isTokenExpired는 angular2-jwt 패키지의 JwtHelper.isTokenExpired 메소드를 사용하여 유효 기간을 검증한다.

TYPESCRIPT

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import { JwtHelper } from 'angular2-jwt';

import 'rxjs/add/observable/of';
import 'rxjs/add/operator/do';
import 'rxjs/add/operator/shareReplay';

import { environment } from '../../environments/environment';

import { User } from '../models/user';
import { Token } from '../models/token';

@Injectable()
export class AuthService {
  appUrl = environment.apiUrl;
  TOKEN_NAME = 'jwt_token';
```

```
  constructor(private http: HttpClient, private jwtHelper: JwtHelper) {
    console.log('[appUrl] ', this.appUrl);
  }

  signin(credential: User): Observable<Token> {
    return this.http.post<Token>(`${this.appUrl}/auth/signin`, credential)
      .do(res => this.setToken(res.token))
      .shareReplay();
  }

  signout(): void {
    this.removeToken();
  }

  // 토큰 유효성 검증
  isAuthenticated(): boolean {
    const token = this.getToken();
    return token ? !this.isTokenExpired(token) : false;
  }

  getToken(): string {
    return localStorage.getItem(this.TOKEN_NAME);
  }

  setToken(token: string): void {
    localStorage.setItem(this.TOKEN_NAME, token);
  }

  removeToken(): void {
    localStorage.removeItem(this.TOKEN_NAME);
  }

  /*
    token 유효 기간 체크
    The JwtHelper class has several useful methods that can be utilized in y
our components:

    decodeToken
    getTokenExpirationDate
    isTokenExpired
```

```
    npm install angular2-jwt
    https://github.com/auth0/angular2-jwt
  */
  isTokenExpired(token: string) {
    return this.jwtHelper.isTokenExpired(token);
  }

  getUserid(): string {
    return this.jwtHelper.decodeToken(this.getToken()).userid;
  }
}
```

2.2.4 인증 가드 (auth.guard.ts)

인증 가드는 인증 서비스의 isAuthenticated 메소드를 사용하여 토큰의 유효성을 검증하여 접근 권한을 체크하고 접근을 제어한다.

TYPESCRIPT

```
import { Injectable } from '@angular/core';
import { Router, CanActivate } from '@angular/router';
import { AuthService } from '../services/auth.service';

@Injectable()
export class AuthGuard implements CanActivate {

  constructor(private router: Router, private auth: AuthService) { }

  canActivate() {
    // 토큰 유효 기간 확인
    if (!this.auth.isAuthenticated()) {
      console.log('invalid token!');
      this.router.navigate(['signin']);
      return false;
    }
    return true;
  }
}
```

2.2.5 라우팅 모듈 (app-routing.module.ts)

라우팅 모듈은 라우트 구성을 관리한다. 대시보드 컴포넌트의 경우, 인증 가드에 의해 접근을 제어한다.

TYPESCRIPT

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { SigninComponent } from './signin/signin.component';
import { DashboardComponent } from './dashboard/dashboard.component';

import { AuthGuard } from './guards/auth.guard';

const routes: Routes = [
  { path: '', redirectTo: 'dashboard', pathMatch: 'full' },
  { path: 'signin', component: SigninComponent },
  {
    path: 'dashboard',
    component: DashboardComponent,
    canActivate: [AuthGuard],
    data: { title: 'Dashboard', subtitle: 'JWT Authentication' }
  },
  { path: '**', redirectTo: 'dashboard' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

2.2.5 사용자 서비스 (users.service.ts)

사용자 서비스는 사용자 정보를 서버에 요청할 때 사용한다. 사용자 정보를 요청할 때 요청 헤더에 토큰을 담아 전송한다.

TYPESCRIPT

```typescript
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';

import { User } from '../models/user';
import { AuthService } from './auth.service';

import { environment } from '../../environments/environment';

@Injectable()
export class UserService {
  appUrl = environment.apiUrl;

  constructor(private http: HttpClient, private auth: AuthService) {}

  getUsers(): Observable<User[]> {
    const headers = new HttpHeaders()
      .set('Authorization', this.auth.getToken());

    return this.http.get<User[]>(`${this.appUrl}/users`, { headers })
      .shareReplay();
  }
}
```

Reference

- JWT
- Stateless vs Stateful Servers
- Where to Store your JWTs – Cookies vs HTML5 Web Storage