Backstage의 인증 및 권한 관리

- 공식 문서
- Backstage의 인증 시스템은 아래 두 가지 목적을 가집니다.
  - 로그인 및 사용자 인증
  - 3rd-party 리소스에 접근 권한 위임
- Backstage는 기본적으로 GitHub, GitLab, Okta 등 다양한 인증 제공자들을 제공합니다.

인증 제공자 설정

- 각 인증 제공자의 설정 부분은 `app-config.yaml` 의 아래 부분에 있습니다. 예를 들어 GitHub의 경우, 아래와 같습니다.

```
• auth:
    environment: development
    providers:
      github:
        development:
          clientId: ${AUTH_GITHUB_CLIENT_ID}
          clientSecret: ${AUTH_GITHUB_CLIENT_SECRET}
```

로그인 설정

- 로그인 설정을 하기 위해서는 프론트엔드 (`app`) 과 백엔드의 `auth` 플러그인 모두를 설정해야 합니다.
  - 기본적으로 `packages/app/src/apis.ts` 에 설정되어 있는 `auth` 플러그인은 GitHub, GitLab, Azure, 그리고 BitBucket을 지원합니다.

로그인 설정 - 프론트엔드 `app`

- `packages/app/src/App.tsx` 에 아래와 같이 `createApp()` 함수에 components 인자로 들어가 있는 `SignInPage` 를 수정하면 됩니다.

```
import { githubAuthApiRef } from '@backstage/core-plugin-api';
import { SignInPage } from '@backstage/core-components';

//..

const app = createApp({
  //..
  components: {
    SignInPage: props => {
      <SignInPage
        {...props}
        auto
        provider={{
          id: 'github-auth-provider',
          title: 'Github',
          message: 'Sign in using GitHub',
          apiRef: githubAuthApiRef
        }}
    }
  }
})
```

Sign-in identities and resolvers

- 공식 문서
- 기본적으로 Backstage 인증 제공자(auth provider)는 오직 접근 권한 위임을 위해서만 사용됩니다.
  그리고 이는 Backstage가 사용자를 대신해 외부 시스템에 접근하도록 해줍니다.
  (ex. CI에서 빌드 trigger)
- 사용자들이 로그인하도록 하는 인증 제공자를 사용하고 싶다면, 명시적으로 로그인을 활성화시켜야 하고 외부의 identity가 Backstage 내부의 identity와 mapping될 방법을 명시해야 합니다.

- `npx @backstage/create-app` 으로 생성된 Backstage 프로젝트는 기본적으로 GitHub guest access를 위한 sign-in resolver를 제공합니다. 이 resolver는 모든 사용자들을 단일한 “guest” 라는 신원으로 판단하고, 단지 Backstage를 빠르게 실행해보기 위한 최소한의 기능만을 수행합니다.
- 이 resolver는 모든 사용자가 Backstage내에서 동일한 “guest”라는 identity로 인식되므로 프로덕션에서는 절대 사용되어서는 안되며, 심지어 누가 로그인할 수 있는지도 제어가 되어 있지 않습니다.
- 기본적으로 제공되는 guest resolver는 `packages/backend/src/plugins/auth.ts` 에 있으며, 아래와 같이 구성되어 있습니다.

```
• export default async function createPlugin(
    env: PluginEnvironment
  ): Promise<Router> {
    return await createRouter({
      //
      providerFactories: {
        ...defaultAuthProviderFactories,
        github: providers.github.create({
          signIn: {
            resolver(_, ctx) {
              const userRef = 'user:default/guest';
              return ctx.issueToken({
                claims: {
                  sub: userRef,
                  ent: [userRef]
                }
              })
            }
          }
        })
      }
    })
  }
```

Backstage 사용자 identity

- Backstage에서 사용되는 사용자 identity는 아래 두 가지로 구성되어 있습니다.
  - User identity reference
  - Ownership entity reference들의 목록
- 특정 사용자가 로그인을 하면 Backstage는 위의 두 가지 정보를 담은 token을 생성하고, 이후 해당 사용자는 이 token을 들고 Backstage를 사용하게 됩니다.
  - `sub` : user entity reference
  - `ent` : ownership entity reference
- User identity reference는 Backstage에 로그인한 사용자를 고유하게 식별할 수 있어야 합니다. Software Catalog에 로그인한 사용자와 매칭되는 user identity가 있는 것이 권장되지만, 필수는 아닙니다. 만약 catalog에 user identity가 있다면, 해당 사용자에 대한 추가적인 정보를 저장할 수 있습니다. 일부 plugin들은 이를 필수 요구사항으로 두기도 합니다.

- Ownership entity reference도 마찬가지로 entity reference이며, 마찬가지로 Software Catalog에 있는 것이 권장됩니다. Ownership reference는 특정 사용자가 소유하는 것들을 판별하는 데에 사용됩니다.
  예를 들어 사용자 Jane (`user:default/jane`)이 있을 때 Jane은 `user:default/jane`, `group:default/team-a`, `group:default/admins` 의 ownership reference를 가질 수 있습니다.
  이러한 ownership reference들을 토대로 `user:jane` , `team-a` 또는 `admins` 가 소유하는 entity는 Jane이 소유하는 것과 동일하게 취급됩니다.

Sign-in Resolvers

- Backstage에 사용자를 로그인하게 하려면 3rd party 인증 제공자가 제공한 사용자의 identity와 Backstage 사용자 identity가 매핑되어야 합니다. 이 매핑 관계에는 기본(default) 방식이 없기에 조직 및 인증 정보 제공자 등에 따라 각기 방법이 다릅니다.

- Sign-in resolver 함수의 입력은 3rd party 인증 제공자에 로그인을 정상적으로 완료했을 때의 정보와 사용자 검색, 토큰 발행 등에 필요한 다양한 보조 도구들을 포함하는 context 객체입니다. 추가적으로 이미 만들어진 sign-in resolver들도 다수 존재합니다.

Built-in Resolvers

- `auth` backend plugin은 일반적인 로그인 패턴들을 위해 기본적으로 sign-in resolver를 제공합니다.
  Google, GitLab 등 다양한 resolver들이 있습니다.

Custom Resolvers

- 예를 들어 cloudforet을 위해 GitHub 이메일 주소가 `@megazone.com` 또는 `@mz.co.kr` 를 포함한다면 `group:maintainers` 에 소속하고, 그 외는 `group:guests` 에 배정하고 싶다고 한다면 아래와 같이 수정하면 됩니다. (`packages/backend/src/plugins/auth.ts`)

```
export default async function createPlugin(
  env: PluginEnvironment
): Promise<Router> {
  return await createRouter({
    //
    providerFactories: {
      ...defaultAuthProviderFactories,
      github: providers.github.create({
        signIn: {
          resolver(info, ctx) {
            const {result: { fullProfile: { displayName, emails } } } =
info;
            const userRef = `user:github/${displayName}`
            if(emails && emails.length > 0) {
              if(emails.find(email => email.value.endsWith("@megazone.
com" || "@mz.co.kr"))) {
                return ctx.issueToken({
                  claims: {
                    sub: userRef,
                    ent: [`group:maintainers`],
                  }
                })
              }
            }

            return ctx.issueToken({
              claims: {
                sub: userRef,
                ent: [`group:guests`]
              }
            })
          }
        }
      })
    }
  })
}
```

Software template으로부터 레포지토리 생성

- Software template으로 레포지토리를 생성할 때, 기본적으로는 Backstage integration 설정에 기입된 Github 사용자의 정보(`app.config.yaml` 의 `auth.providers.github`)를 토대로 레포지토리가 생성됩니다.

- 이렇게 하지 않고, Backstage에 로그인한 사용자의 GitHub에 레포지토리를 생성하게끔 하려면, Software template에 사용되는 레포지토리의 `template.yaml` 파일에 아래와 같이 추가적인 설정을 해줘야 합니다.
  - 공식 문서

```
# ..
spec:
  # ..
  parameters:
    # ..
    - title: Choose a location
      required:
        - repoUrl
      properties:
        repoUrl:
          title: Repository location
          type: string
          ui:field: RepoUrlPicker
          ui:options:
            allowedHosts:
              - github.com
            #    (requestUserCredentials)
            requestUserCredentials:
              secretsKey: USER_OAUTH_TOKEN
              additionalScopes:
                github:
                  - workflow

# ..
steps:
  # ..
  - id: publish
    name: Publish
    action: publish:github
    input:
      allowedHosts: ['github.com']
      description: This is ${{ parameters.name }}
      repoUrl: ${{ parameters.repoUrl }}
      # token
      token: ${{ secrets.USER_OAUTH_TOKEN }}
# ..
```