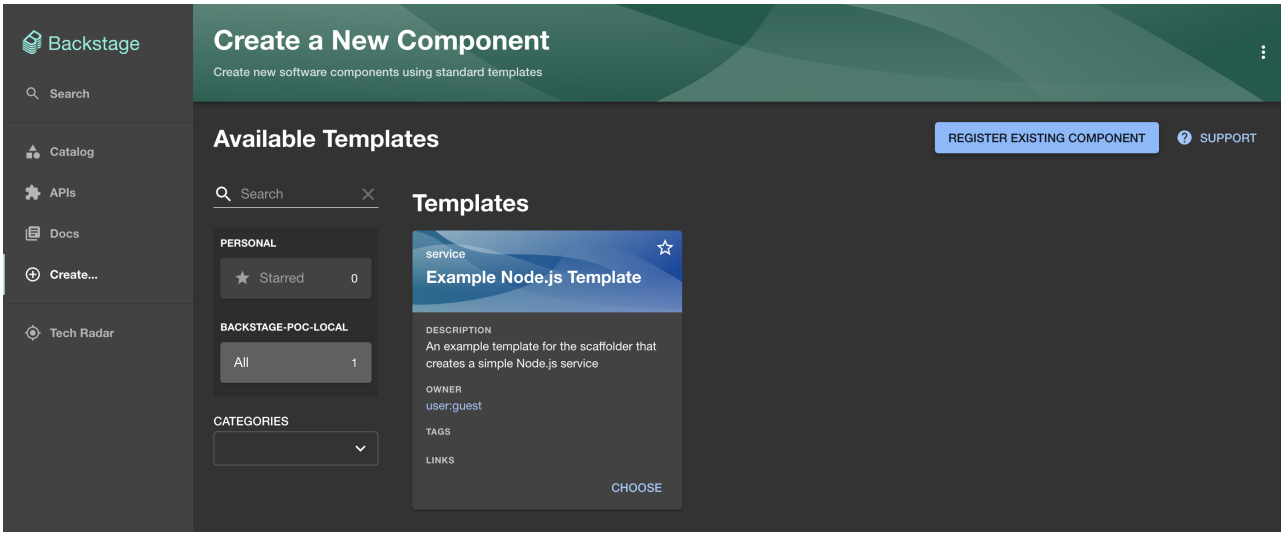
Backstage

- Backstage: 개발자 포탈을 만들기 위한 open platform

Backstage Software Catalog

- 마이크로서비스, 라이브러리, 데이터 pipeline, 웹사이트 등을 관리하기 위한 도구
- Backstage는 코드와 함께 위치해 있는 YAML 파일을 가시화해준다.
- 두 가지 주요 목적
  - 방대한 소프트웨어들의 관리 및 운영
  - 특정 조직이 운영하는 모든 소프트웨어가 각각 소유자가 누구인지, 어떻게 사용되는지 정보 제공
- Catalog를 추가하는 방법
  - 방법 1: 직접 추가
    - 아래 사진처럼 `REGISTER EXISTING COMPONENT` 를 클릭해 등록하는 방법
    - 

  - 방법 2: yaml 파일에 등록
    - `app-config.yaml` 의 `catalog.locations` 에 아래와 같이 추가

```
#..
catalog:
  #..
  locations:
    - type: url
      target: https://github.com/backstage/backstage/blob/master
/packages/catalog-model/examples/components/artist-lookup-
component.yaml
```

Backstage Software Templates

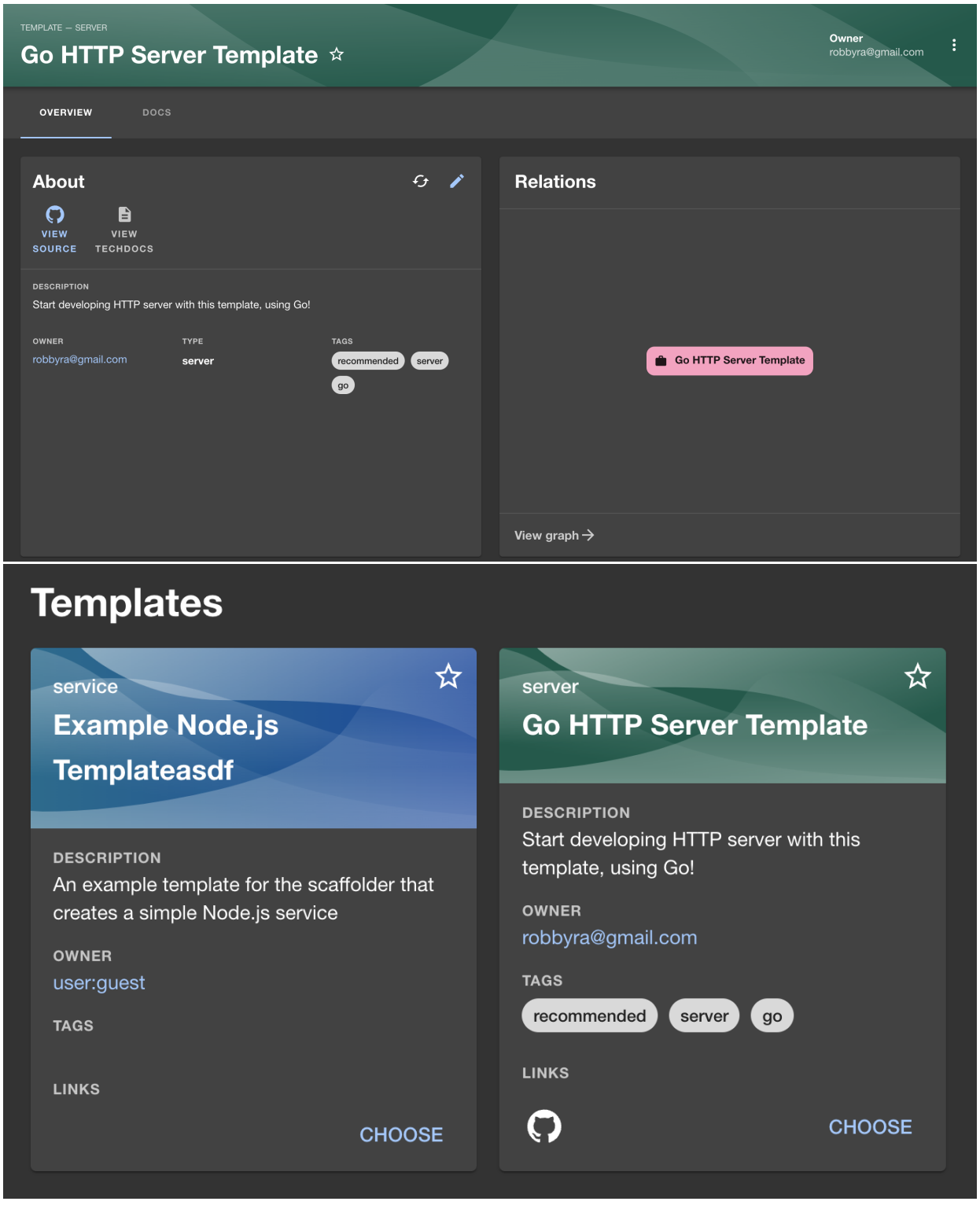
- 새로운 프로젝트를 빠르게 시작하기 위한 template code 등을 제공하는 도구
- 사용자들은 해당 template code 부터 시작해 새로운 프로젝트를 해나갈 수 있다.
  - GitHub, GitLab에 자동으로 repository 생성까지 가능하다.
- Template을 설정하는 방법
- Template을 직접 추가하는 방법
- 예시 - GitHub Public repository의 코드를 기반으로 새로운 repository 생성
  - 예시 레포지토리 코드
  - (1) Template repository에는 두 개의 파일이 있어야 합니다.
    - `template.yaml`
    - `catalog-info.yaml`

- (2) `template.yaml` 보기
  - 이 yaml 파일은 아래와 같습니다.

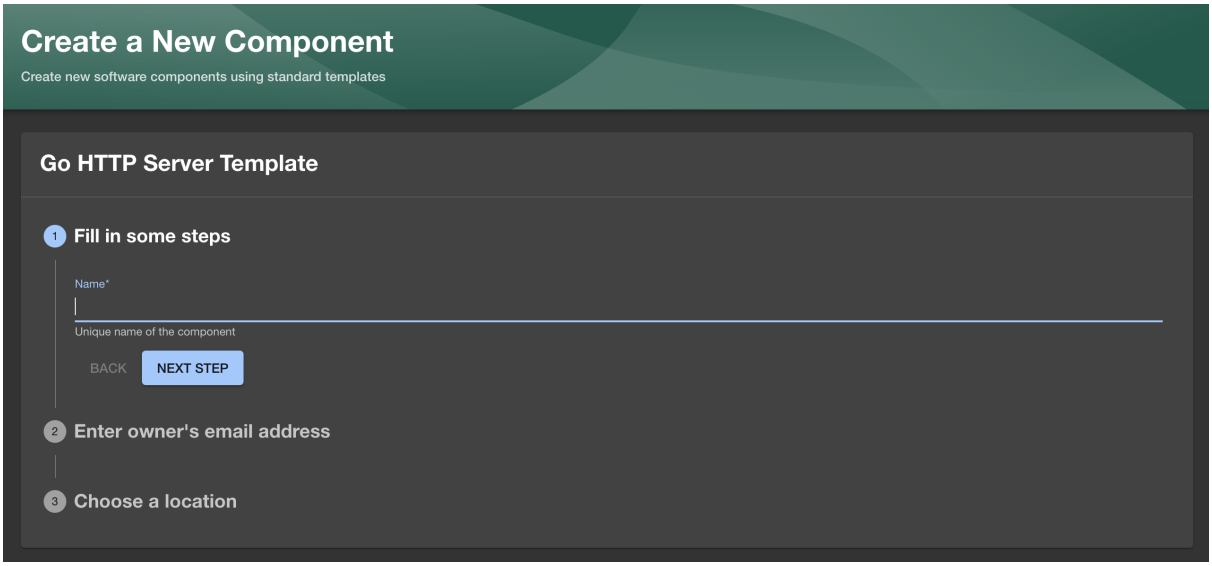
```
- apiVersion: scaffolder.backstage.io/v1beta3
  kind: Template
  metadata:
    name: go-http-server-template
    title: Go HTTP Server Template
    description: 'Start developing HTTP server with this template,
  using Go!'
    tags:
      - recommended
      - server
      - go
  spec:
    owner: robbyra@gmail.com
    type: server
    parameters:
      - title: Fill in some steps
        required:
          - name
        properties:
          name:
            title: Name
            type: string
            description: Unique name of the component
            ui:autofocus: true
            ui:options:
              rows: 5
      - title: Enter owner's email address
        required:
          - owner
        properties:
          owner:
            title: Owner
            type: string
            description: Owner of the component
            ui:autofocus: true
            ui:options:
              rows: 5
      - title: Choose a location
        required:
          - repoUrl
        properties:
          repoUrl:
            title: Repository Location
            type: string
            ui:field: RepoUrlPicker
            ui:options:
              allowedHosts:
```

```
              - github.com
  steps:
    - id: fetch-base
      name: Fetch Base
      action: fetch:template
      input:
        targetPath: .
        url: https://github.com/sang-w0o/backstage-template-
example-repo
        values:
          name: ${{ parameters.name }}
          owner: ${{ parameters.owner }}

    - id: publish
      name: Publish
      action: publish:github
      input:
        allowedHosts: ['github.com']
        description: This is ${{ parameters.name }}
        repoUrl: ${{ parameters.repoUrl }}

    - id: register
      name: Register
      action: catalog:register
      input:
        repoContentsUrl: ${{ steps['publish'].output.
repoContentsUrl }}
        catalogInfoPath: '/catalog-info.yaml'

  output:
    links:
      - title: Repository
        url: ${{ steps['publish'].output.remoteUrl }}
      - title: Open in catalog
        icon: catalog
        entityRef: ${{ steps['register'].output.entityRef }}
```

- 위 yaml 파일을 읽어와 Backstage에 적용하면, 아래와 같이 나옵니다.



- 그리고 이 yaml 파일에는 사용자가 해당 template으로부터 코드를 생성할 때, 어떤 단계로 만들지, 어떤 정보를 받을지 등을 지정합니다. 예를 들어, 위 `template.yaml` 에는 `spec.parameters` 에 `title` 이 3개 있습니다. 따라서 사용자는 이 아래와 같이 세 단계를 거치게 됩니다.
  -



- (3) `catalog-info.yaml` 보기
  - 이 파일은 Backstage가 Software Template 으로부터 만들어진 프로젝트를 추적하기 위해 사용됩니다.
    기존 `catalog-info.yaml` 은 아래와 같습니다.

```
• apiVersion: backstage.io/v1alpha1
  kind: Component
  metadata:
    name: ${{ values.name | dump }}
  spec:
    type: server
    lifecycle: experimental
    owner: ${{ values.owner | dump }}
```

  - 하지만 사용자가 Backstage에서 프로젝트를 생성해 새롭게 만들어진 repository에 생기는 `catalog-info.yaml` 은 아래와 같이 생성 단계에서 받은 값들로 실제 값이 채워집니다.

```
• apiVersion: backstage.io/v1alpha1
  kind: Component
  metadata:
    name: "TestGoComponent"
  spec:
    type: server
    lifecycle: experimental
    owner: "robbyra@mz.co.kr"Backstage TechDocs
```

- (4) Backstage에 Template 등록해주기
  - Backstage가 예제 레포지토리를 Software Template으로 인식해 UI에 표시해주기 위해서는, `app-config.yaml` 을 수정해야 합니다.
    아래와 같이 Github URL을 등록하면 됩니다.

```
# ..
catalog:
  # ..
  locations:
    - type: url
      target: https://github.com/sang-w0o/backstage-template-example-repo/blob/main/template.yaml
      rules:
        - allow: [Template]
```

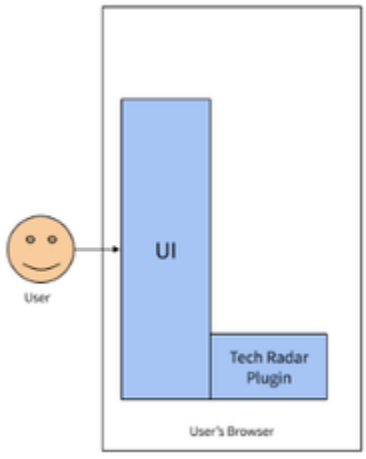
- 기술 문서를 코드처럼 작성해 생성, 유지, 검색, 그리고 사용을 쉽게 해주는 도구

Plugin 종류

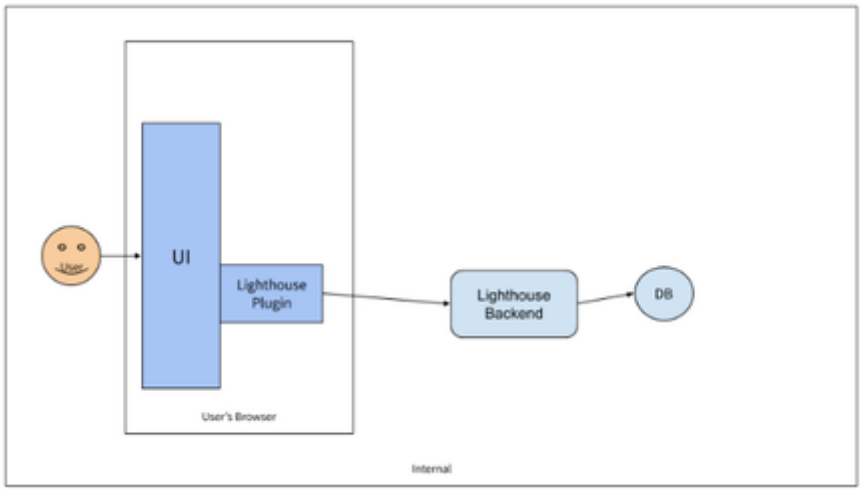
- 플러그인은 주로 React로 제공되며, 아래처럼 사용 가능하다.

```
import { CatalogIndexPage } from "@backstage/plugin-catalog";

//..

const routes = (
  <FlatRoute>
    <Route path="/catalog" element={<CatalogIndexPage />} />
  </FlatRoute>
);
```

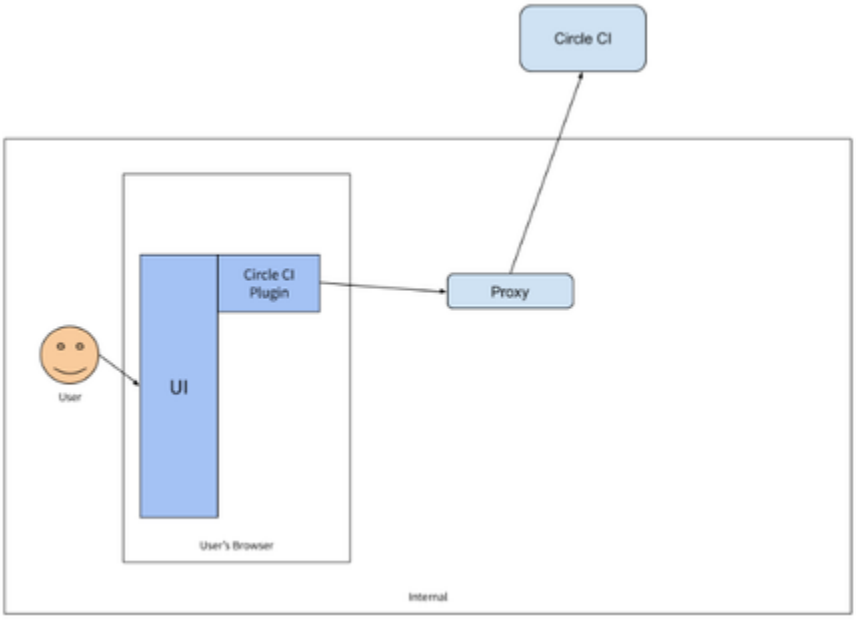
- Standalone plugin: 다른 서비스로 API 요청을 보내지 않고, UI에서만 작동하는 플러그인



- Service backed plugin: Backstage를 실행 중인 서비스에 요청을 보내 활용하는 플러그인



- Third-party backed plugin: CircleCI 처럼 외부에서 운영되는 서비스에 요청을 보내 활용하는 플러그인



Backstage TechDocs

- TechDocs: 코드와 함께 markdown 형식의 코드를 작성해 문서화하는 도구

문서 생성 방식 지정(builder)

- Backstage는 문서 생성을 CI/CD 단계에서 처리할 것을 권장한다. 이는 `app-config.yaml` 에서 지정 가능하다.

```
techdocs:
  bulder: 'local' # 'external'
```

문서 저장 방식 지정(publisher)

- TechDocs가 생성된 문서가 어디에 저장되어 있는지 파악하기 위해 설정해야 하는 부분이다.
  이를 publisher가 담당하며, local, GCS, S3가 사용 가능하다.
- 마찬가지로 `app-config.yaml` 에서 설정 가능하다.

```
techdocs:
  builder: 'local'
  publisher:
    type: 'local' # 'googleGcs', 'awsS3'
```

TechDocs를 생성하기 위한 과정

- (1) TechDocs Preparer
  - Preparer는 GitHub, GitLab 등의 소스 코드 호스팅 제공자에게서 markdown 파일들을 불러오고, 이 파일들을 generator에게 넘겨준다.
  - 현재는 2 가지의 preparer들이 있다.
    - Common Git Preparer - `git clone` 을 사용해 repository URL을 주면 가져오는 방식
    - URL Reader - GitHub, GitLab 등의 API를 제공해 파일을 불러오는 방식 (Backstage 권장)
- (2) TechDocs Generator
  - Markdown 파일을 토대로 정적 HTML 파일 및 asset들을 생성한다.
    - TechDocs container 또는 `mkdocs` CLI 를 활용한다.
- (3) TechDocs Publisher
  - Publisher는 Generator가 만들어낸 문서를 스토리지에 업로드한다.

Github Public Repository template으로부터 TechDocs 생성하기

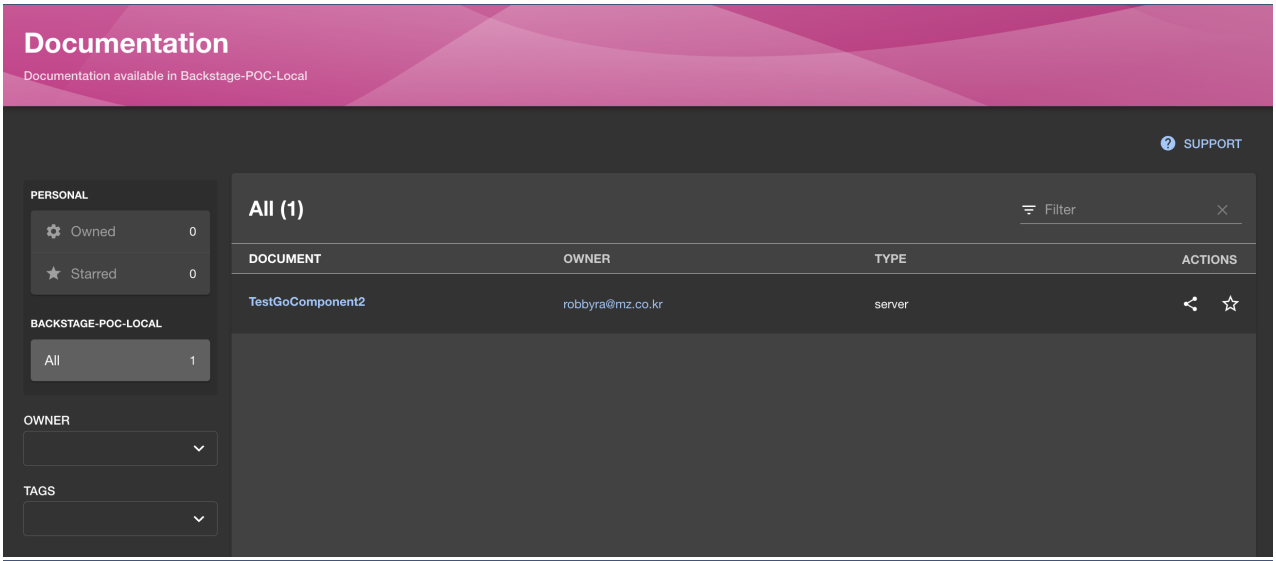
- TechDocs를 지원하는 레포지토리의 구조는 아래와 같다.

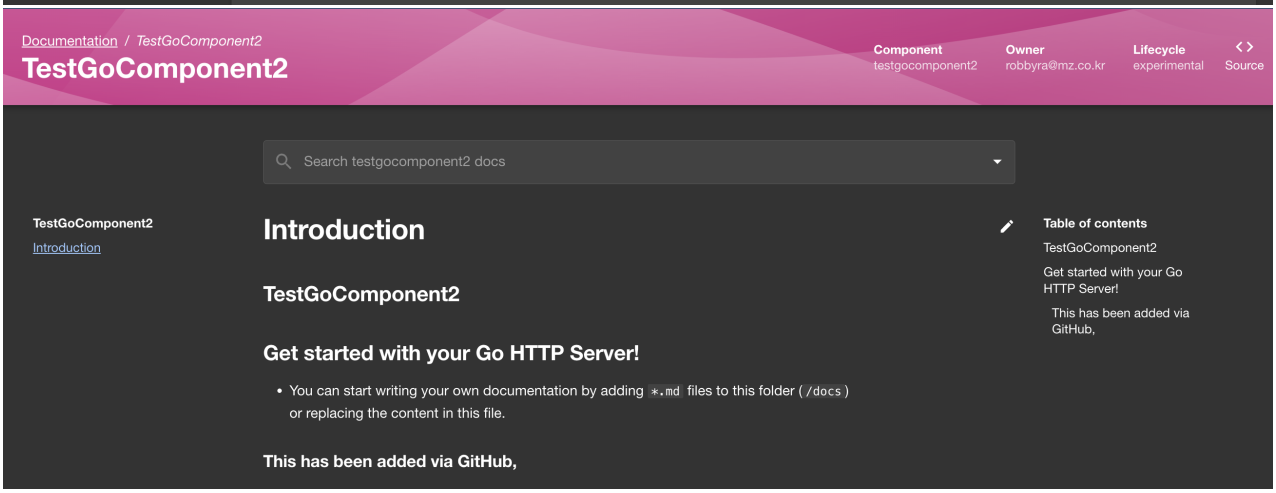
```
|-- template.yaml
|-- skeleton
      |--
      |-- catalog-info.yaml
      |-- mkdocs.yml -> mkdocs TechDocs Generator
      |-- docs
            |-- index.md
            |-- .md
```

- 예시 레포지토리를 Software Template으로 만든 프로젝트의 구조는 아래와 같다.

```
|-- docs -> markdown
|--
|-- mkdocs.yml
|-- catalog-info.yml
```

- 그리고 Backstage에서 문서를 보면, 아래와 같이 나온다.





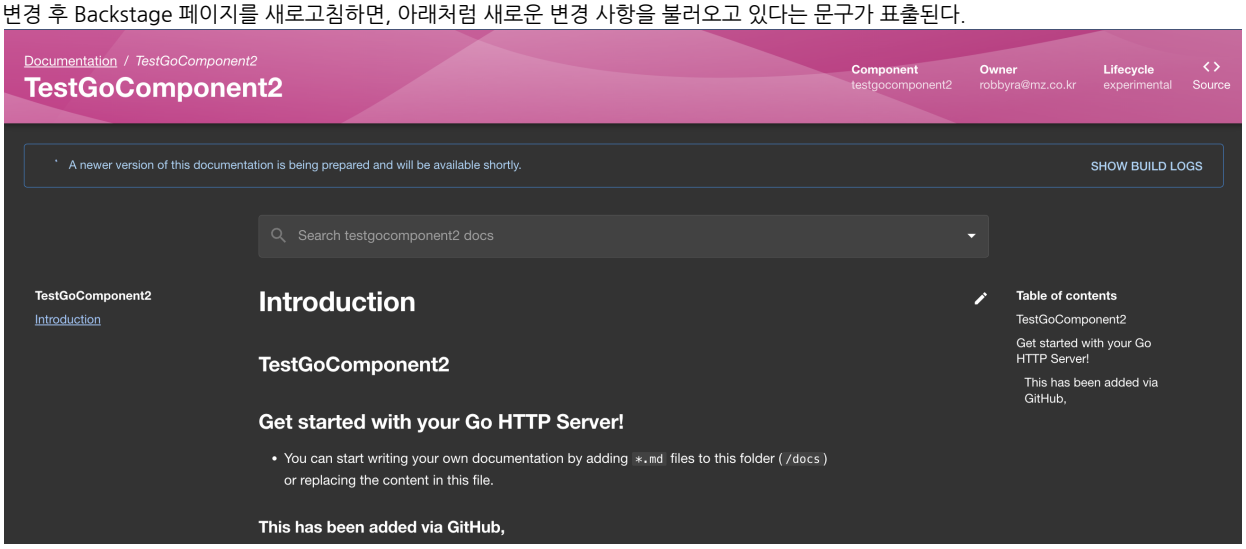
- 기본적으로 아래의 `mkdocs.yml` 에 따라, `docs/index.md` 만 Backstage에 나타난다.

```
- site_name: TestGoComponent2
  site_description:
  nav:
    - Introduction: index.md
  plugins:
    - techdocs-core
```

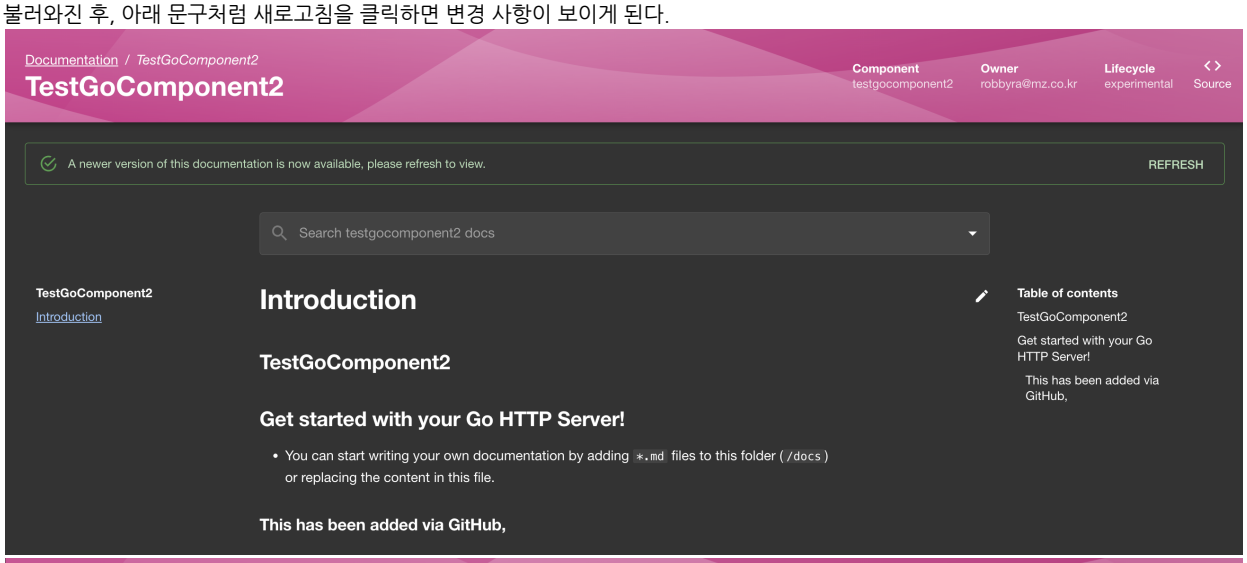
- 그리고 새로운 변경 사항이 발생하면, Backstage가 polling 형태로 변경 사항을 감지해 자동으로 문서를 재생성한다.
- 만약 `docs/index.md` 말고 추가적인 markdown 파일이 필요하다면, `mkdocs.yml` 도 함께 수정해야 한다.
  - 예를 들어 `docs/2.md` 를 만들었다면, 이 내용이 Backstage에서 보이게 하려면 `mkdocs.yml` 을 아래처럼 변경해야 한다.

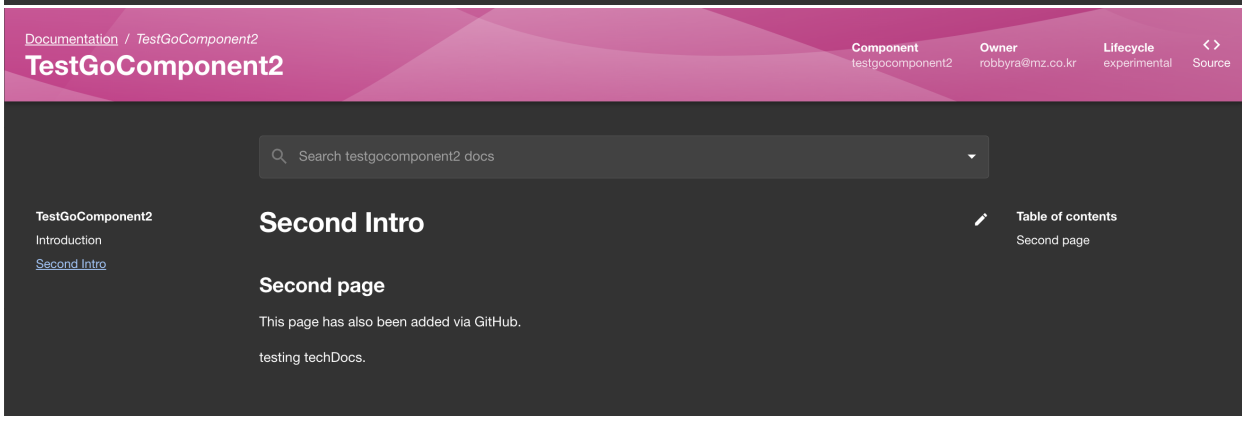
```
- site_name: TestGoComponent2
  site_description:
  nav:
    - Introduction: index.md
    - Second Intro: 2.md # key: Backstage  , value: markdown
  plugins:
    - techdocs-core
```

- 변경 후 Backstage 페이지를 새로고침하면, 아래처럼 새로운 변경 사항을 불러오고 있다는 문구가 표출된다.



- 불러와진 후, 아래 문구처럼 새로고침을 클릭하면 변경 사항이 보이게 된다.





K8S로 배포하기

- Docker 문서
- Kubernetes 문서
- 배포 자체는 위 두 문서를 따라하면 매우 쉽게 할 수 있으며, 요약하자면 Backstage를 구성하려면 두 개의 컴포넌트가 필요합니다.
  - Backstage container
  - PostgreSQL container: 이 부분은 문서에서는 pod로 띄우지만, 가능하면 별도의 데이터베이스를 구축해 사용하는 것이 프로덕션 환경에서는 좋을 것 같습니다.

- 배포 방식은 아래와 같습니다.
  - (1) `npx @backstage/create-app` 으로 생성한 앱의 디렉토리에서 `yarn build:backend` 수행
  - (2) docker build & push
    - `docker build --platform=linux/amd64 -t ${IMAGE_REPOSITORY}/${IMAGE_NAME} . -f packages/backend/Dockerfile`
    - `docker push ${IMAGE_REPOSITORY}/${IMAGE_NAME}`
  - (3) Kubernetes deployment apply

- `npx @backstage/create-app` 으로 생성한 앱에는 세 개의 설정 파일들이 있습니다.
  - `app-config.yaml` : 기본 값들로 채워 사용합니다. local을 위한 설정 파일로 기본적으로 gitignore되어 있습니다.
  - `app-config.local.yaml` : 같은 값이 `app-config.yaml`에도 지정되어 있을 때, `app-config.local.yaml` 의 값이 `app-config.yaml` 의 값을 override 합니다.
  - `app-config.production.yaml` : 같은 값이 `app-config.yaml`에도 지정되어 있을 때, `app-config.production.yaml` 의 값이 `app-config.yaml` 의 값을 override 합니다.

techdocs 삽질기

1. mkdocs 에러 관련 Dockerfile 설정

   - Backstage는 TechDocs에서 markdown 형식의 파일을 HTML로 변환하기 위해 mkdocs 라는 라이브러리를 사용합니다.
   - 기본적으로 제공되는 `packages/backend/Dockerfile` 에는 mkdocs 관련 의존성 추가 내용이 없어, 문서에 따라 추가해야 합니다. 문서에서는 Dockerfile의 가장 하단에 두 줄을 추가하라고 나와 있지만, 이렇게 하면 실패합니다.
     - 이유: 위에 `USER node` 라인이 있는데, `apt-get`은 sudo 권한을 필요로 하기에 `apt-get` 이 실패합니다.
     - PR 추가해두었어요!
   - 따라서 문서에서 추가하라는 두 줄을 `USER node` 윗 부분에 추가해야 합니다.
     완성된 Dockerfile은 아래와 같습니다.

```
FROM node:16-bullseye-slim

RUN --mount=type=cache,target=/var/cache/apt,sharing=locked \
    --mount=type=cache,target=/var/lib/apt,sharing=locked \
    apt-get update && \
    apt-get install -y python3 python3-pip --no-install-recommends
libsqlite3-dev build-essential && \
    yarn config set python /usr/bin/python3

RUN pip3 install mkdocs-techdocs-core==1.1.7

USER node

WORKDIR /app

ENV NODE_ENV production

COPY --chown=node:node yarn.lock package.json packages/backend/dist
/skeleton.tar.gz ./
RUN tar xzf skeleton.tar.gz && rm skeleton.tar.gz

RUN --mount=type=cache,target=/home/node/.cache/yarn,sharing=locked,
uid=1000,gid=1000 \
    yarn install --frozen-lockfile --production --network-timeout
300000

COPY --chown=node:node packages/backend/dist/bundle.tar.gz app-
config*.yaml ./
RUN tar xzf bundle.tar.gz && rm bundle.tar.gz

CMD ["node", "packages/backend", "--config", "app-config.yaml", "--
config", "app-config.production.yaml"]
```

- 마지막으로 techdocs generator(mkdocs를 실행하는 위치)가 컨테이너 내부, 즉 local임을 `app-config.yaml` 에 지정해줍니다.

```
techdocs:
  builder: 'local'
  generator:
    runIn: 'local'
  publisher:
    type: 'local'
```

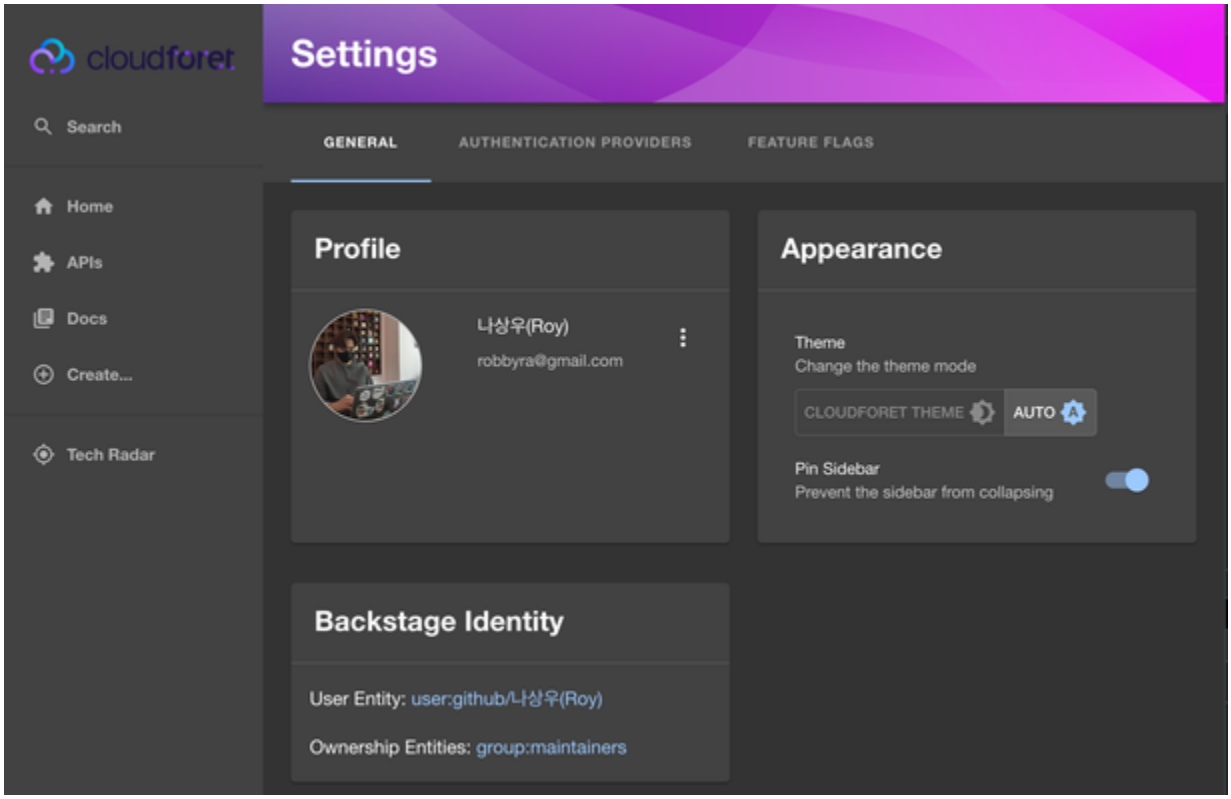
커스텀 테마 적용하기

- `npx @backstage/create-app` 으로 처음 생성한 Backstage의 초록색 header 영역을 아래 방식대로 하면 직접 수정할 수 있습니다.
- 수정할 파일은 `package/app/src/App.tsx` 입니다.

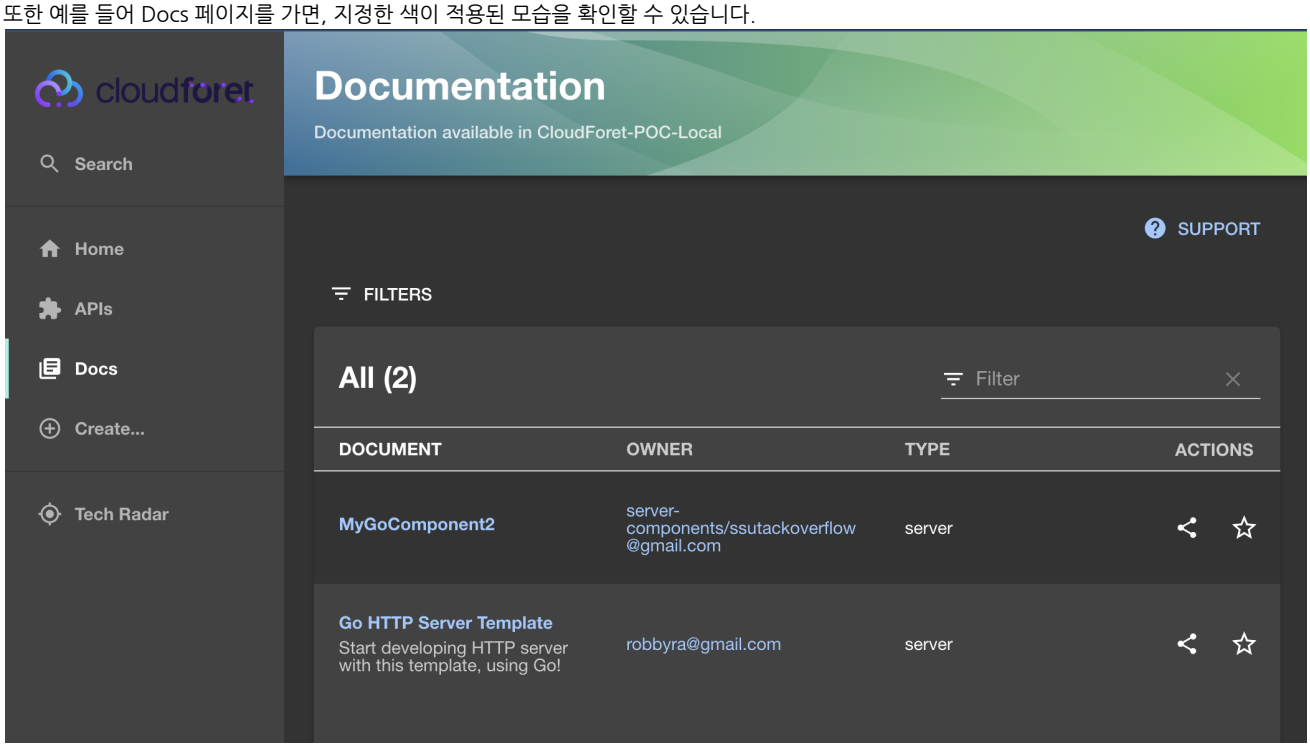
```
import {
  createTheme,
  genPageTheme,
  shapes,
} from "@backstage/theme";
import { darkTheme } from "@backstage/theme";
import { CssBaseline, ThemeProvider } from "@material-ui/core";
import DarkIcon from "@material-ui/icons/Brightness4";

const cloudForetTheme = createTheme({
  palette: darkTheme.palette,
  defaultPageTheme: "home",
  pageTheme: {
    home: genPageTheme({colors:["#572b94", "#ee1bf5"], shape: shapes.
wave}),
    website: genPageTheme({colors:["#111111", "#FFFFFF"], shape:
shapes.square}),
    apis: genPageTheme({colors:["#b81f40", "#d9771c"], shape: shapes.
wave}),
    documentation: genPageTheme({colors:["#32719c", "#83de52"],
shape: shapes.wave}),
    create: genPageTheme({colors: ['#111111', '#FFFFFF'], shape:
shapes.wave}),
  }
});

const app = createApp({
  apis,
  themes: [{
    id: "cloudforet_theme",
    title: "CloudForet Theme",
    variant: "dark",
    icon: <DarkIcon/>,
    provider: ({ children }) => (
      <ThemeProvider theme={cloudForetTheme}>
        <CssBaseline>{children}</CssBaseline>
      </ThemeProvider>
    ),
  }],
  components: {
    //..
  },
  //..
})
```

<DarkIcon /> 에 사용되는 @material-ui/icons/Brightness4 는 여기에서 확인할 수 있습니다.

- 이제 아래와 같이 Settings 페이지에 가면 `CloudForet Theme` 이 추가된 모습을 확인할 수 있습니다.



- 또한 예를 들어 Docs 페이지를 가면, 지정한 색이 적용된 모습을 확인할 수 있습니다.



2. TechDocs에서의 사진 파일 사용

- Markdown 파일에서 `<img src="" />` 처럼 이미지 파일의 경로를 지정할 때, mkdocs가 경로를 이상하게 잡습니다. 예를 들어 아래 구조처럼 되어 있다고 가정해봅니다.

```
|-- docs
      |-- en
           |-- GUIDE.md
           |-- GUIDE-img
                 |-- a.png
```

- 이 상황에서 `GUIDE.md` 가 `a.png` 를 참조하려면 아래처럼 작성합니다.

```
<img src="GUIDE-img/a.png" />
```

- 그리고 mkdocs.yml 파일은 아래처럼 되어 있을 것입니다.

```
site_name:
site_description:
nav:
  - Guide:
    - KR: ko/GUIDE.md
```

- 이렇게 하고 mkdocs를 실행하면, 사진을 제대로 불러오지 못합니다. 그 이유는 빌드된 HTML이 참조하는 사진 파일의 경로가 `<>/en/GUIDE/GUIDE-img/a.png` 이기 때문입니다. 하지만 제대로된 경로는 `<>/en/GUIDE-img/a.png` 입니다. 그리고 Backstage에서 해당 문서에 접근할 때 붙는 path 또한 `/en/GUIDE` 입니다.
- 위 path를 자세히 보면, `/en/GUIDE/` 부분에서 `GUIDE`는 markdown 파일의 이름임을 알 수 있습니다.
  따라서 이를 가장 쉽게 해결하려면 `GUIDE.md` 파일의 이름을 `index.md` 로 바꿔주면 됩니다.
- `index.md`로 바꾸고 `mkdocs.yml` 파일도 알맞게 바꿔준 후, 다시 TechDocs를 생성하면 이미지도 `<>/en/GUIDE-img/a.png` 로 잘 렌더링되며, Backstage의 path도 `/en/` 으로 변경됩니다.