**Backstage**

# Overview

Backstage is an open platform for building developer portals. Powered by a centralized software catalog, Backstage restores order to your microservices and infrastructure and enables your product teams to ship high-quality code quickly — without compromising autonomy.

Backstage unifies all your infrastructure tooling, services, and documentation to create a streamlined development environment from end to end.

# Get started

1. **Prerequisites**

* **Node.js**

First make sure you are using Node.js with an Active LTS Release, currently v14. This is made easy with a version manager such as nvm which allows for version switching.

# Installing a new version

***nvm install 14***

> Downloading and installing node v14.15.1...

> Now using node v14.15.1 (npm v6.14.8)

# Checking your version

***node –version***

> v14.15.1

* **Yarn**
* **Install via npm**

***npm install --global yarn***

#Check installation

***yarn –version***

* **Docker**

We use Docker for a few of our core features. So, you will need Docker installed locally to use features like Software Templates and TechDocs.

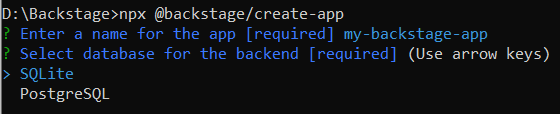
1. **Create an App**

To install the Backstage Standalone app, we make use of npx, a tool to run Node executables straight from the registry. Running the command below will install Backstage. The wizard will create a subdirectory inside your current working directory.

*npx @backstage/create-app*

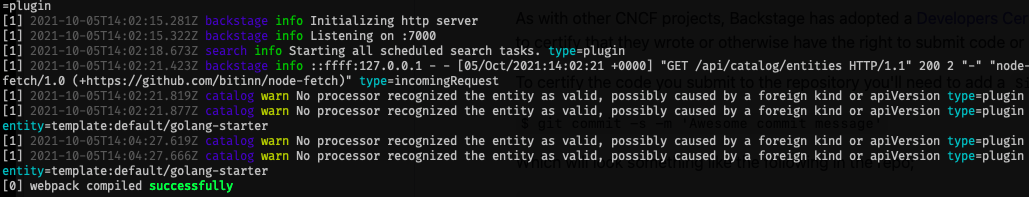
The wizard will ask you

* The name of the app, which will also be the name of the directory
* The database type to use for the backend. For this guide, you'll be using the SQLite option.

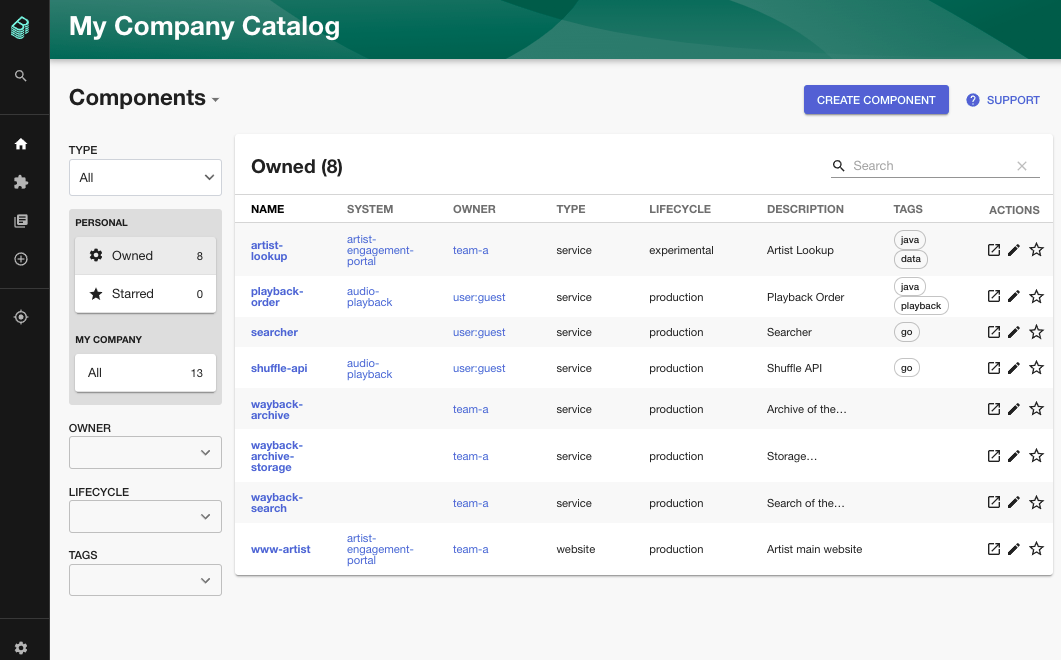


1. **Run the Backstage App**

When the installation is complete you can go to the application directory and start the app. The yarn dev command will run both the frontend and backend as separate processes (named [0] and [1]) in the same window.



It might take a little while, but as soon as the message [0] webpack compiled successfully appears, you can open a browser and directly navigate to your freshly installed Backstage portal at http://localhost:3000. You can start exploring the demo immediately.



# Running Backstage Locally

**1. Authentication**

When Backstage starts, you can choose to enter as a Guest user and start exploring.

But you can also set up any of the available authentication methods. The easiest option will be GitHub. To setup GitHub authentication in Backstage, see [these instructions](https://github.com/backstage/backstage/tree/master/plugins/auth-backend#github).

* **Adding Authentication**

Authentication in Backstage identifies the user, and provides a way for plugins to make requests on behalf of a user to third-party services. Backstage can have zero (guest access), one, or many authentication providers. The default @backstage/create-app template uses guest access for easy startup.

* **Adding provider configuration**

Each built-in provider has a configuration block under the auth section of app-config.yaml. For example, the GitHub provider:

*auth:*

*environment: development*

*providers:*

*github:*

*development:*

*clientId: ${AUTH\_GITHUB\_CLIENT\_ID}*

*clientSecret: ${AUTH\_GITHUB\_CLIENT\_SECRET}*

* **Adding the provider to the sign-in page**

After configuring an authentication provider, the app frontend package needs a small update to show this provider as a login option. The SignInPage component handles this, and takes either a provider or providers (array) prop of SignInProviderConfig definitions.

These reference the ApiRef exported by the provider. Again, an example using GitHub that can be adapted to any of the built-in providers:

*# packages/app/src/App.tsx*

*import { githubAuthApiRef } from '@backstage/core-plugin-api';*

*import { SignInProviderConfig, SignInPage } from '@backstage/core-components';*

*const githubProvider: SignInProviderConfig = {*

*id: 'github-auth-provider',*

*title: 'GitHub',*

*message: 'Sign in using GitHub',*

*apiRef: githubAuthApiRef,*

*};*

*const app = createApp({*

*apis,*

*components: {*

*SignInPage: props => (*

*<SignInPage*

*{...props}*

*auto*

*provider={githubProvider}*

*/>*

*),*

*},*

*bindRoutes({ bind }) {*

**2. Included providers**

* **GitHub**
  + **Create an OAuth App on GitHub**

Settings for local development:

Application name: Backstage (or your custom app name)

Homepage URL: http://localhost:3000

Authorization callback URL: http://localhost:7000/api/auth/github

* + **Configuration**

The provider configuration can then be added to your app-config.yaml under the root auth configuration:

*auth:*

*environment: development*

*providers:*

*github:*

*development:*

*clientId: ${AUTH\_GITHUB\_CLIENT\_ID}*

*clientSecret: ${AUTH\_GITHUB\_CLIENT\_SECRET}*

The GitHub provider is a structure with three configuration keys:

clientId: The client ID that you generated on GitHub, e.g. b59241722e3c3b4816e2

clientSecret: The client secret tied to the generated client ID.

enterpriseInstanceUrl (optional): The base URL for a GitHub Enterprise instance, e.g. https://ghe.<company>.com. Only needed for GitHub Enterprise.

callbackUrl (optional): The callback url that GitHub will use when initiating an OAuth flow, e.g.

https://your-intermediate-service.com/handler. Only needed if Backstage is not the immediate receiver (e.g. one OAuth app for many backstage instances).

* **Google**
  + **Create OAuth Credentials**

To support Google authentication, you must create OAuth credentials:

1. Log in to the [Google Console](https://console.cloud.google.com/)
2. Select or create a new project from the dropdown menu on the top bar
3. Navigate to [APIs & Services > Credentials](https://console.cloud.google.com/apis/credentials)
4. Click Create Credentials and choose OAuth client ID
5. Configure an OAuth consent screen, if required

* For local development, you do not need to enter any Authorized domain
* For scopes, select openid, auth/userinfo.email and auth/userinfo.profile
* Add yourself as a test user, if using External user type

1. Set Application Type to Web Application with these settings:

* Name: Backstage (or your custom app name)
* Authorized JavaScript origins: http://localhost:3000
* Authorized Redirect URIs: http://localhost:7000/api/auth/google/handler/frame

1. Click Create

* **Configuration**

*auth:*

*environment: development*

*providers:*

*google:*

*development:*

*clientId: ${AUTH\_GITHUB\_CLIENT\_ID}*

*clientSecret: ${AUTH\_GITHUB\_CLIENT\_SECRET}*

* **Adding the provider to the Backstage frontend**

*# packages/app/src/App.tsx*

*import { googleAuthApiRef } from '@backstage/core-plugin-api';*

*import { SignInProviderConfig, SignInPage } from '@backstage/core-components';*

*const googleProvider: SignInProviderConfig = {*

*id: 'google-auth-provider',*

*title: 'Google',*

*message: 'Sign in using Google',*

*apiRef: googleAuthApiRef,*

*};*

*const app = createApp({*

*apis,*

*components: {*

*SignInPage: props => (*

*<SignInPage*

*{...props}*

*providers={['guest', googleProvider]}*

*/>*

*),*

*},*

*bindRoutes({ bind }) {*

* **Microsoft Azure**
  + **Create an App Registration on Azure**

To support Azure authentication, you must create an App Registration:

1. Log in to the [Azure Portal](https://portal.azure.com/)
2. Create an [Active Directory Tenant](https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/Overview), if one does not yet exist
3. Navigate to [Azure Active Directory > App Registrations](https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps)
4. Register an application

* Name: Backstage (or your custom app name)
* Redirect URI: Web > http://localhost:7000/api/auth/microsoft/handler/frame

1. Navigate to Certificates & secrets > New client secret to create a secret

* **Configuration**

*auth:*

*environment: development*

*providers:*

*microsoft:*

*development:*

*clientId: ${AUTH\_MICROSOFT\_CLIENT\_ID}*

*clientSecret: ${AUTH\_MICROSOFT\_CLIENT\_SECRET}*

*tenantId: ${AUTH\_MICROSOFT\_TENANT\_ID}*

* **Adding the provider to the Backstage frontend**

*# packages/app/src/App.tsx*

*import {microsoftAuthApiRef} from '@backstage/core-plugin-api';*

*import { SignInProviderConfig, SignInPage } from '@backstage/core-components';*

*const microsoftAzureProvider: SignInProviderConfig = {*

*id: azure-auth-provider',*

*title: Azure,*

*message: 'Sign in using Microsoft Azure',*

*apiRef: microsoftAuthApiRef,*

*};*

*const app = createApp({*

*apis,*

*components: {*

*SignInPage: props => (*

*<SignInPage*

*{...props}*

*providers={['guest', microsoftAzureProvider]}*

*/>*

*),*

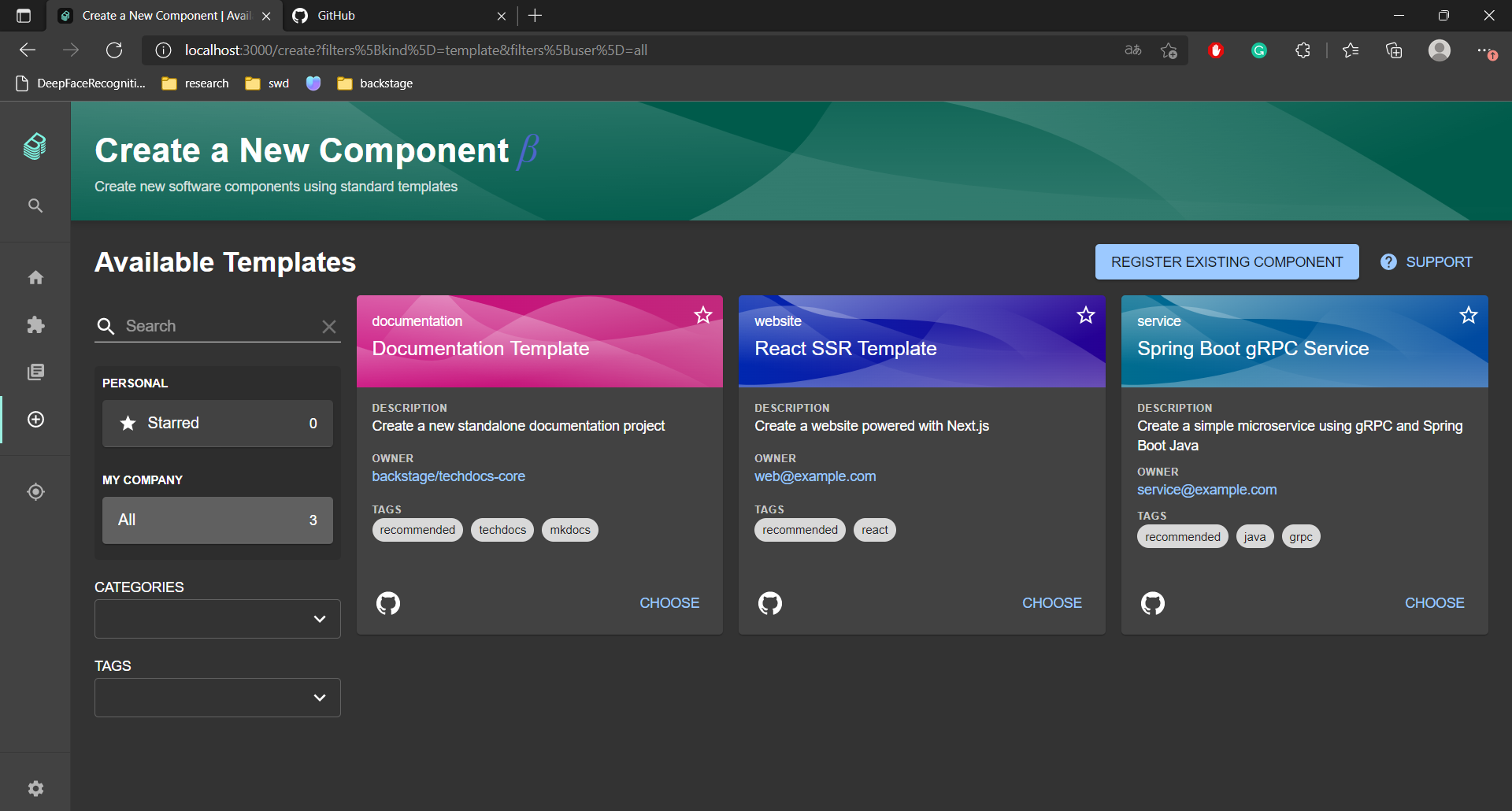
*},*

*bindRoutes({ bind }) {*

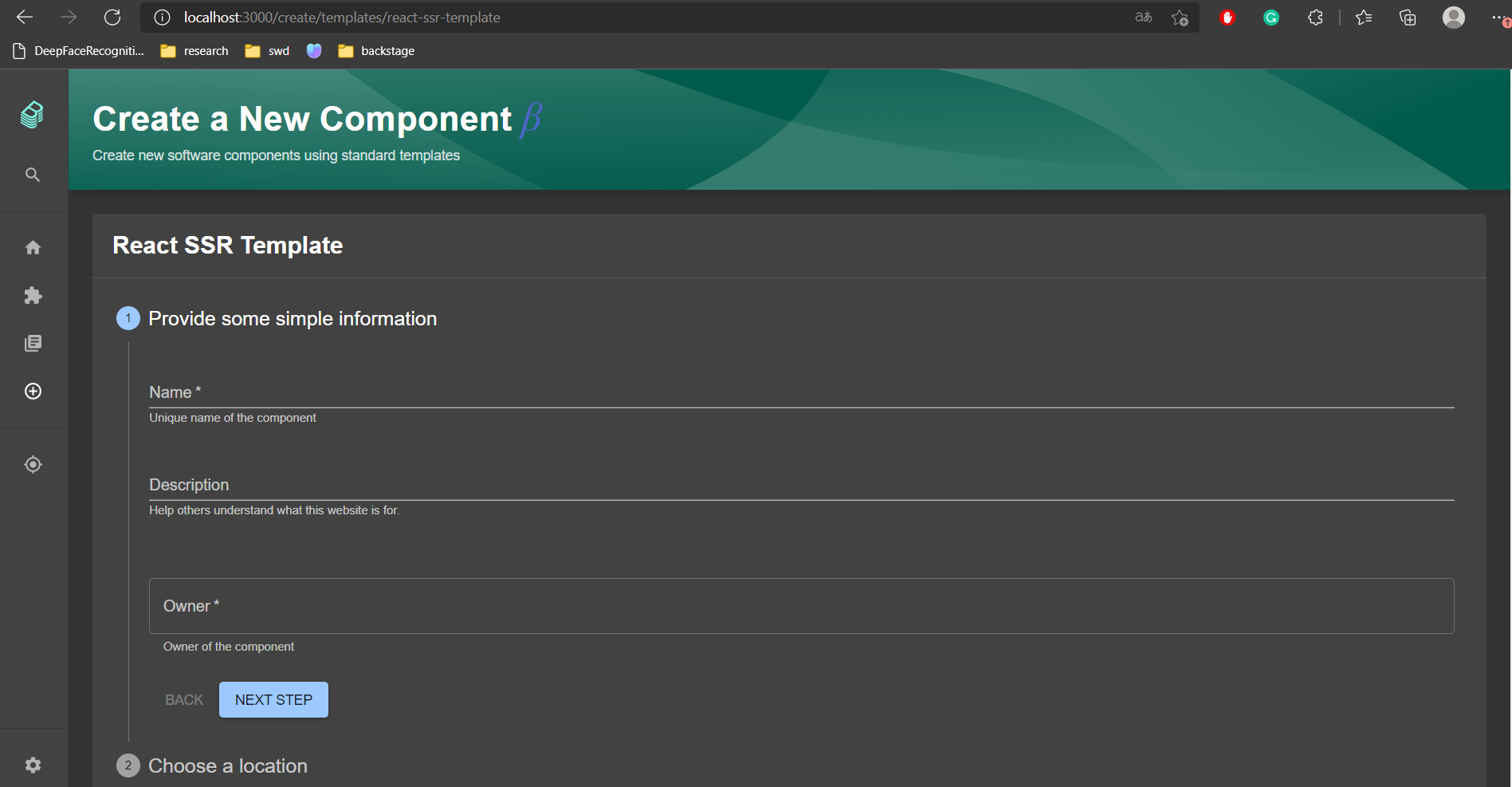
# Component

1. Website
   1. Create new component

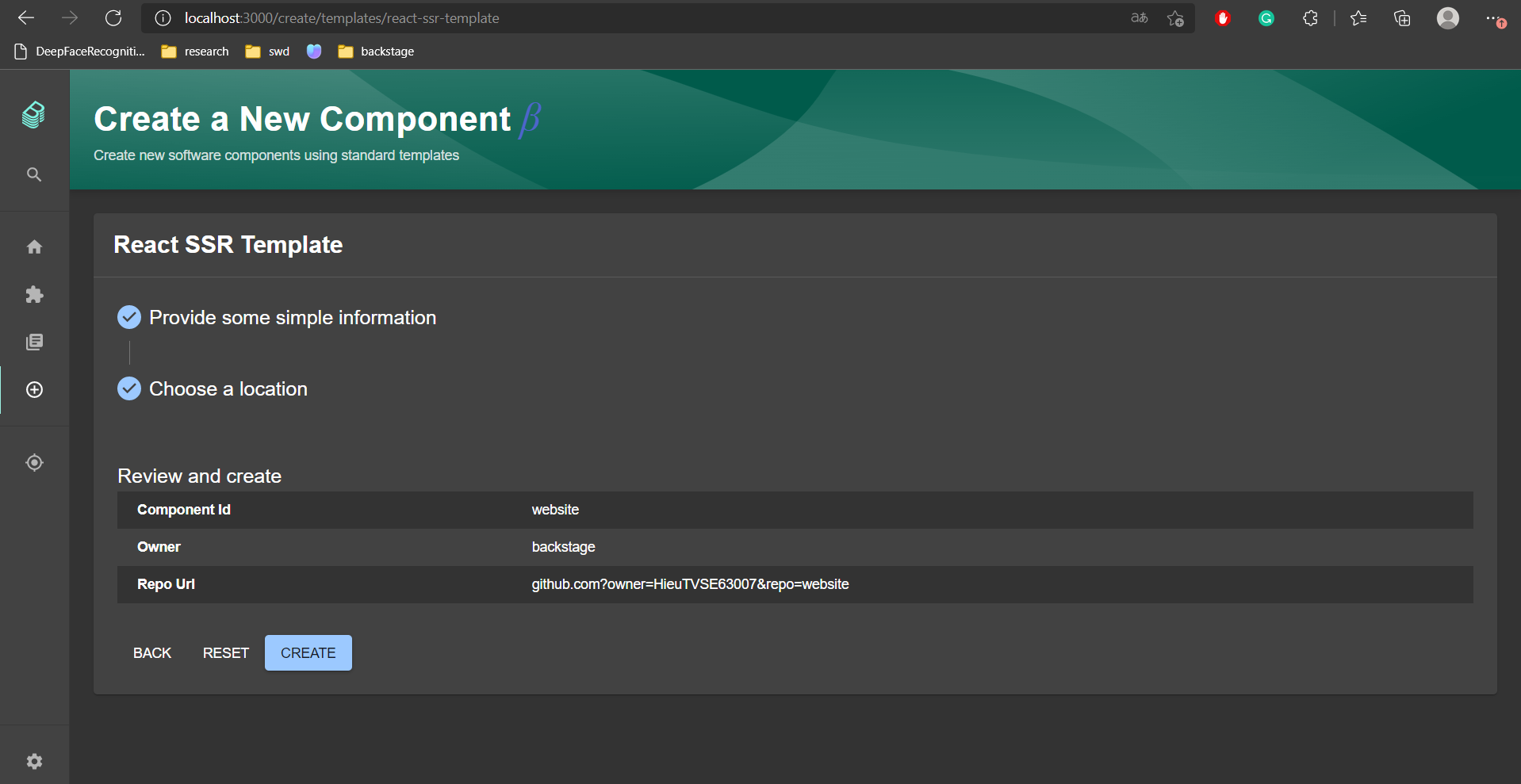
* Select website template



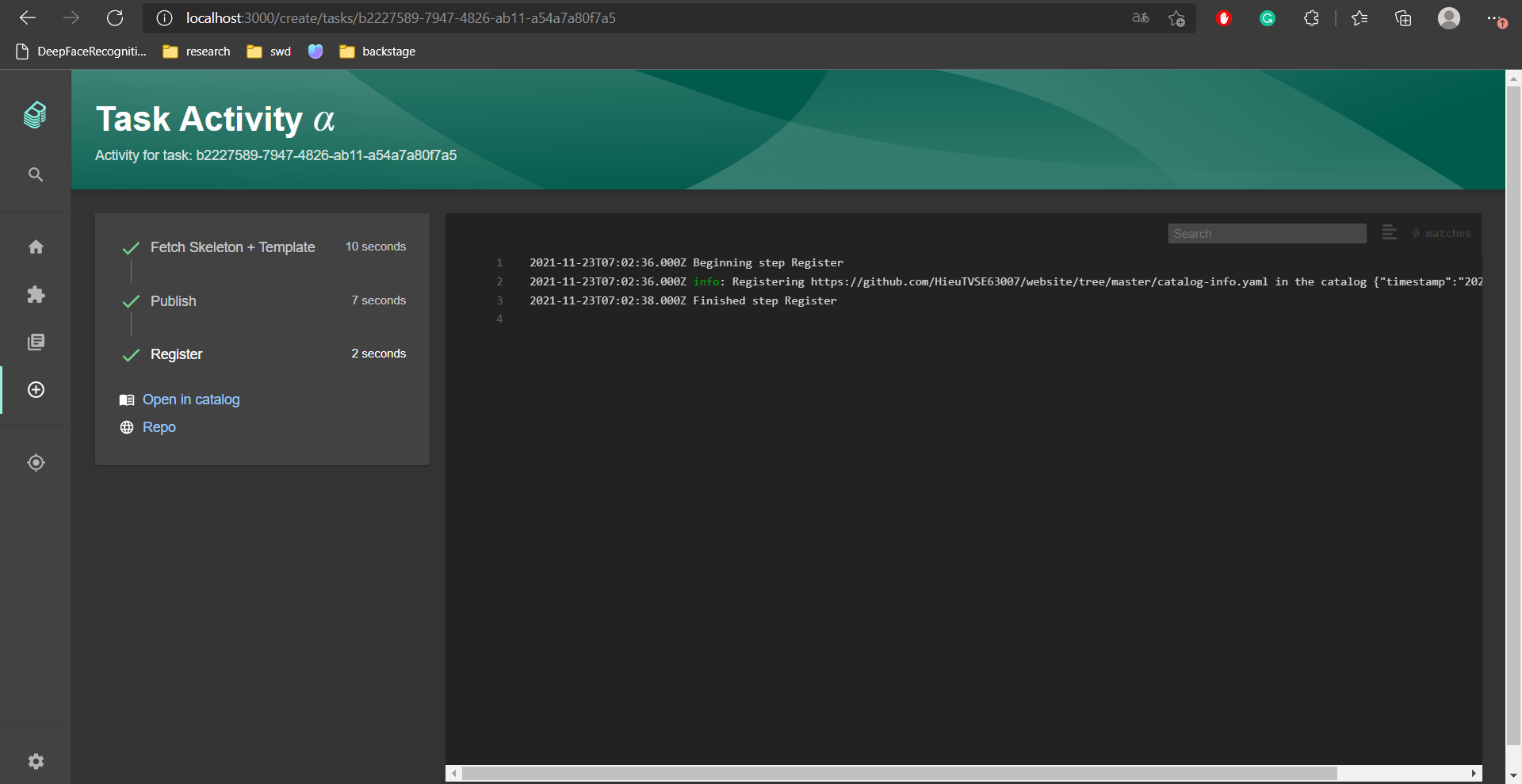
* Provide some simple information and choose a location



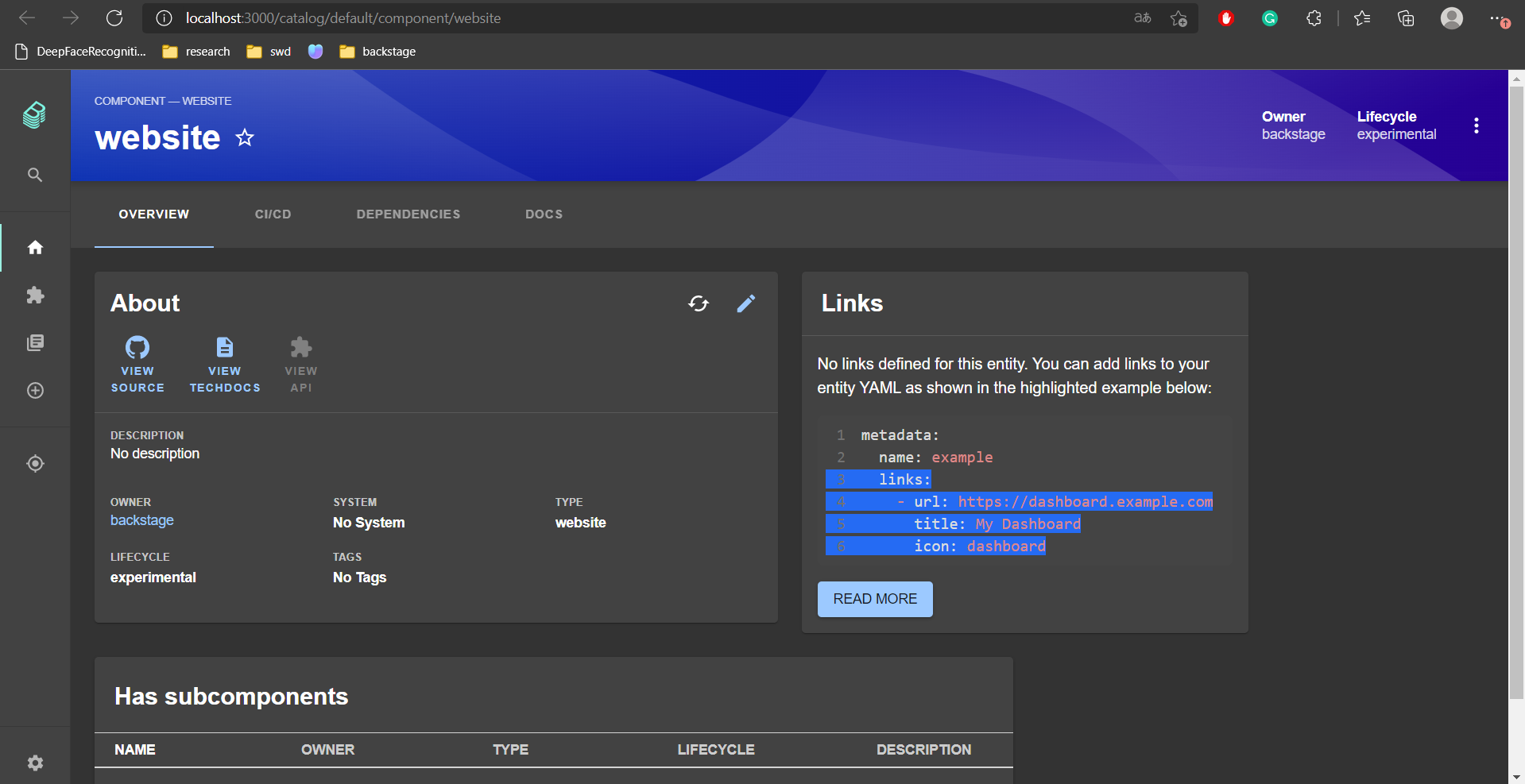
* After filling in all the information, click Create



* Wait for backstage to create your repository



* 1. Description
* The component created as below:



* **System:** An entity reference to the system that the component belongs to, *e.g. artist-engagement-portal*. This field is optional.
* **Lifecycle:** The lifecycle state of the component, e.g. production. This field is required.

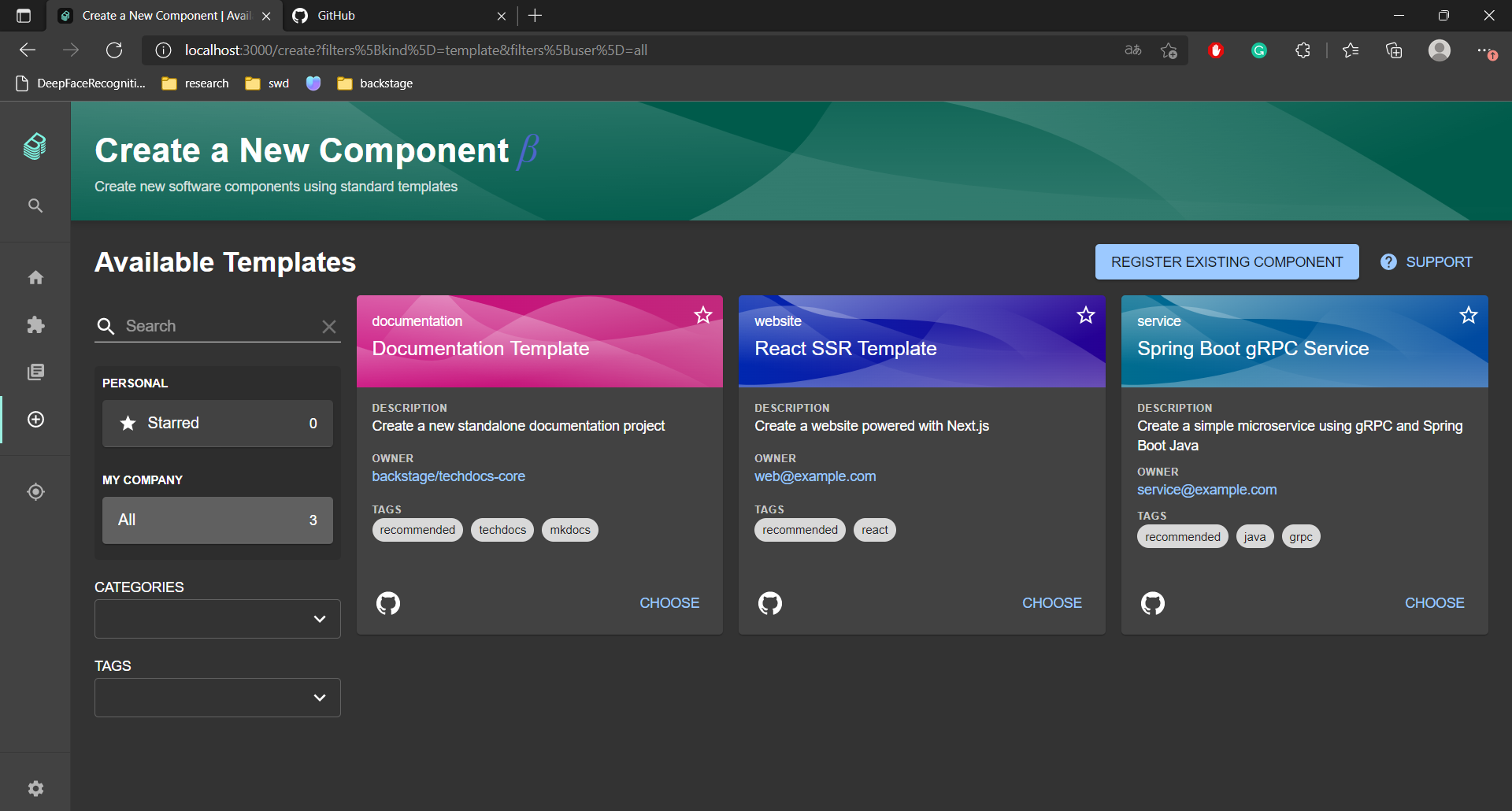
The software catalog accepts any lifecycle value, but an organization should take great care to establish a proper taxonomy for these.

The current set of well-known and common values for this field is:

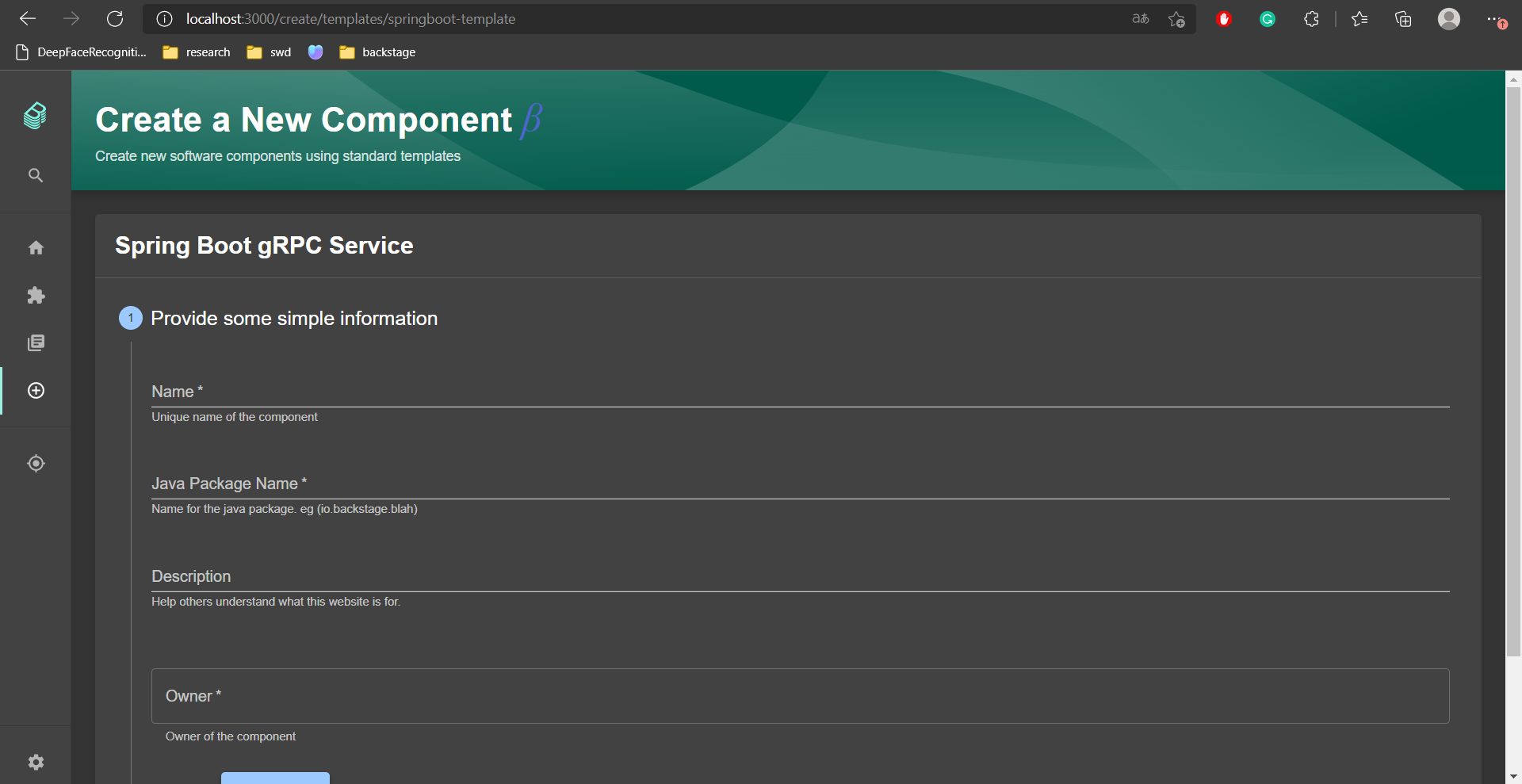
* + **experimental** - an experiment or early, non-production component, signaling that users may not prefer to consume it over other more established components, or that there are low or no reliability guarantees
  + **production** - an established, owned, maintained component
  + **deprecated** - a component that is at the end of its lifecycle, and may disappear at a later point in time
* **Subcomponent:** An entity reference to another component of which the component is a part, *e.g. spotify-ios-app*. This field is optional.
* **DependsOn:** An array of entity references to the components and resources that the component depends on, *e.g. artists-db*. This field is optional.

1. Service
   1. Create new component

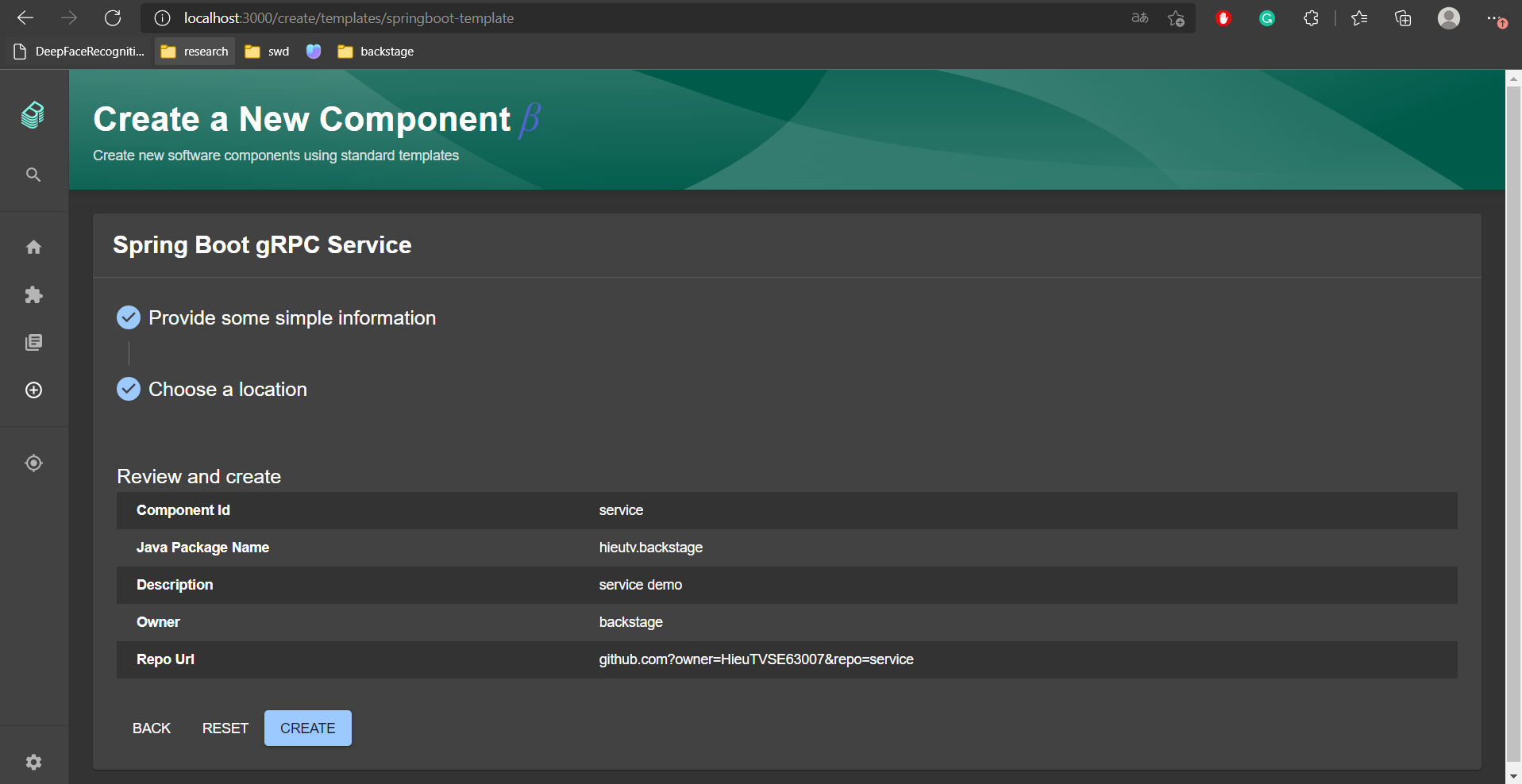
* Select Service template



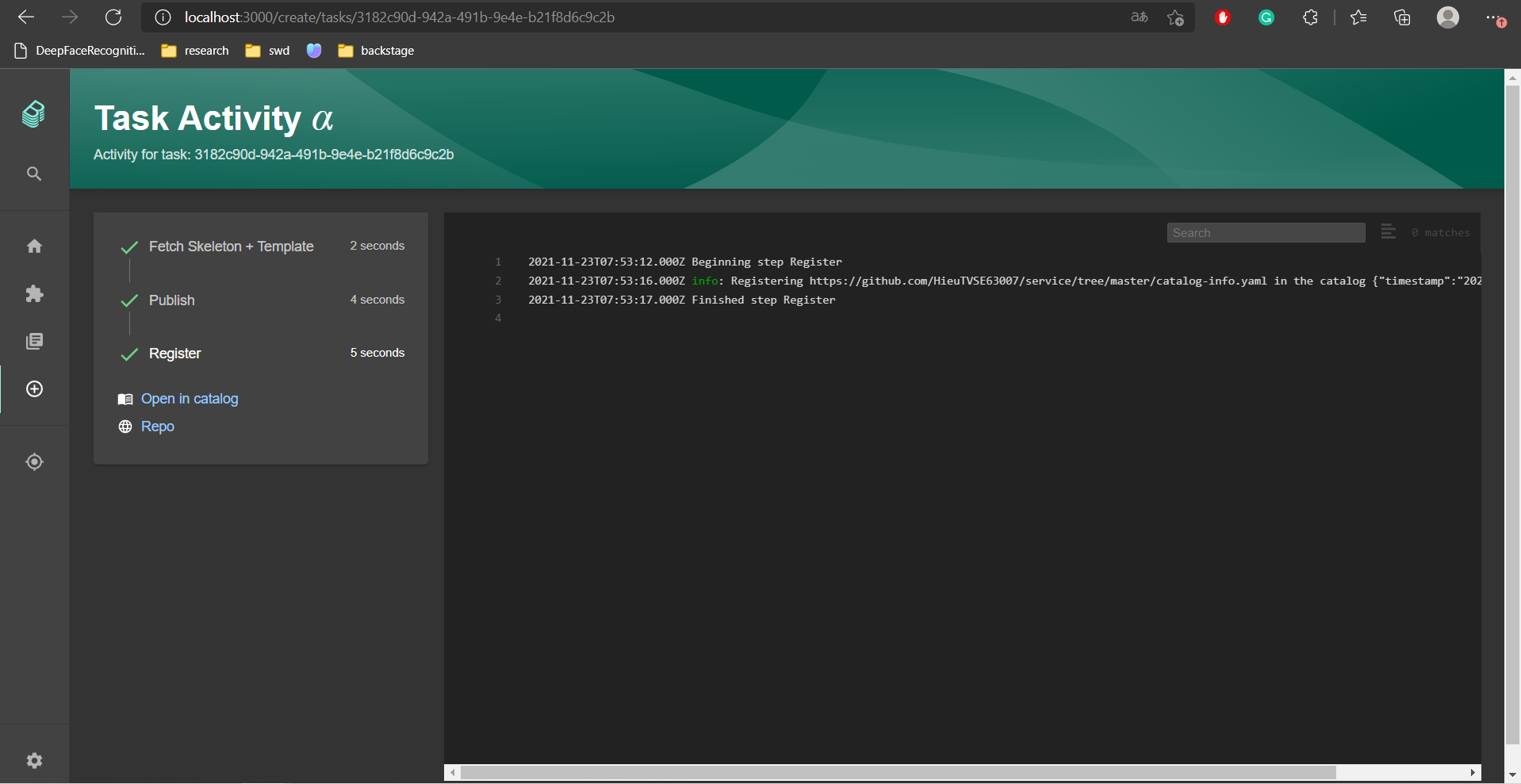
* Provide some simple information and choose a location



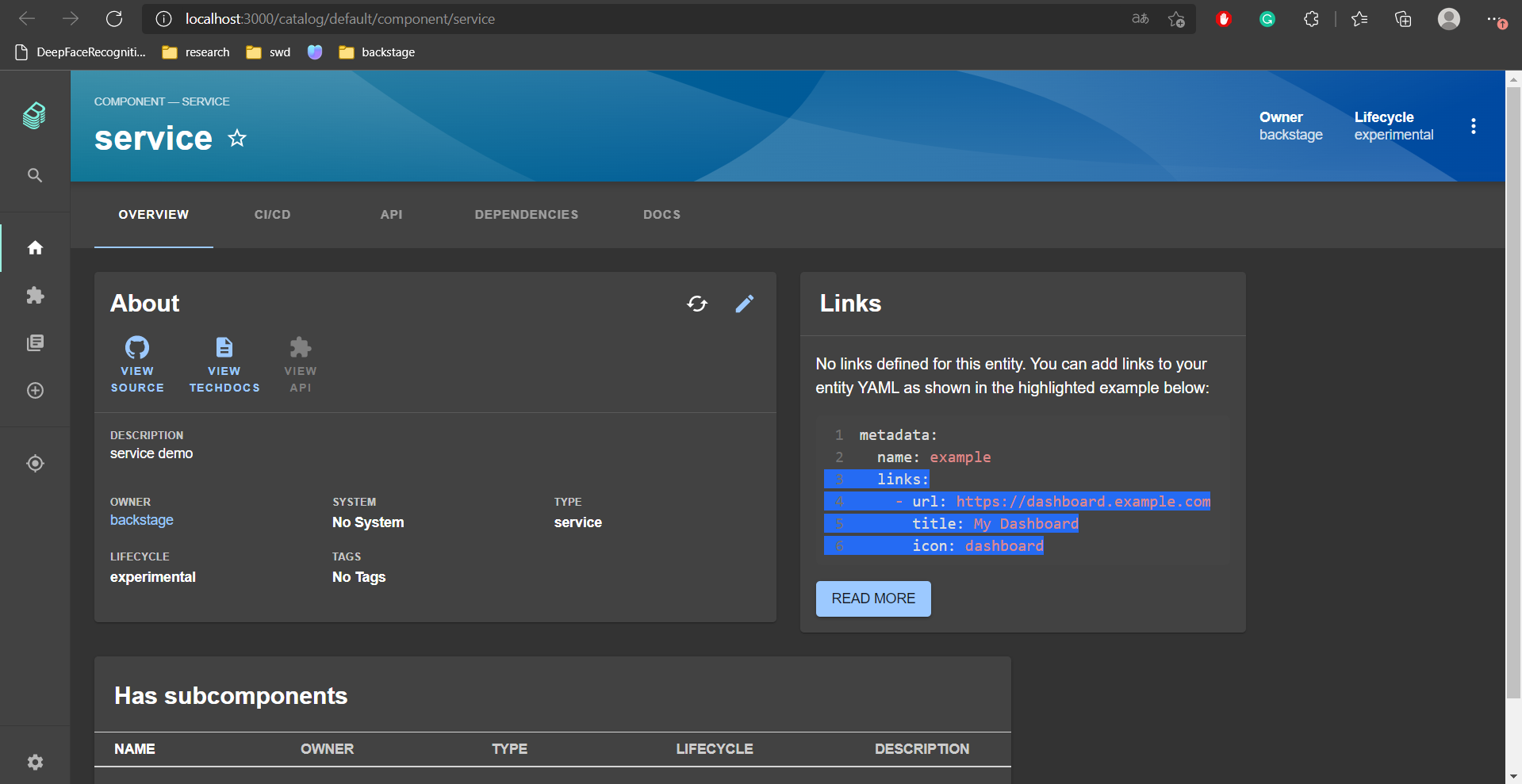
* After filling in all the information, click Create



* Wait for backstage to create your repository



* 1. Description
* The component created as below:



* **System:** An entity reference to the system that the component belongs to, *e.g. artist-engagement-portal*. This field is optional.
* **Lifecycle:** The lifecycle state of the component, e.g. production. This field is required.

The software catalog accepts any lifecycle value, but an organization should take great care to establish a proper taxonomy for these.

The current set of well-known and common values for this field is:

* + **experimental** - an experiment or early, non-production component, signaling that users may not prefer to consume it over other more established components, or that there are low or no reliability guarantees
  + **production** - an established, owned, maintained component
  + **deprecated** - a component that is at the end of its lifecycle, and may disappear at a later point in time
* **Subcomponent:** An entity reference to another component of which the component is a part, *e.g. spotify-ios-app*. This field is optional.
* **ProvidersApis:** An array of entity references to the APIs that are provided by the component, *e.g. artist-api*. This field is optional.
* **ConsumesApis:** An array of entity references to the APIs that are consumed by the component, *e.g. artist-api*. This field is optional.
* **DependsOn:** An array of entity references to the components and resources that the component depends on, *e.g. artists-db*. This field is optional.