**1. Hello World Plugin (Frontend)**

**How to Develop**

1. **Generate a new plugin**

yarn backstage-cli create-plugin --name hello-world

1. **Modify the frontend plugin (packages/plugins/hello-world/src/index.ts)**

import { createPlugin, createRouteRef } from '@backstage/core-plugin-api';

import { HelloWorldPage } from './components/HelloWorldPage';

export const helloWorldRouteRef = createRouteRef({

id: 'hello-world',

});

export const helloWorldPlugin = createPlugin({

id: 'hello-world',

routes: {

root: helloWorldRouteRef,

},

});

export const HelloWorldPageComponent = helloWorldPlugin.provide(

createPage({

element: <HelloWorldPage />,

title: 'Hello World',

path: '/hello-world',

}),

);

1. **Create a React Component (components/HelloWorldPage.tsx)**

import React from 'react';

export const HelloWorldPage = () => {

return <h1>Hello, Backstage!</h1>;

};

1. **Register the plugin in app.tsx**

import { HelloWorldPageComponent } from '@backstage/plugin-hello-world';

const routes = (

<FlatRoutes>

<Route path="/hello-world" element={<HelloWorldPageComponent />} />

</FlatRoutes>

);

1. **Run Backstage**

yarn dev

Visit **http://localhost:3000/hello-world** to see your plugin.

# **2. Joke Generator Plugin**

### ****How to Develop****

### ****1. Generate a new plugin****

yarn backstage-cli create-plugin --name joke-generator

### ****2. Modify the frontend plugin (****packages/plugins/joke-generator/src/index.ts****)****

import { createPlugin, createRouteRef } from '@backstage/core-plugin-api';

import { JokePage } from './components/JokePage';

export const jokeRouteRef = createRouteRef({

id: 'joke-generator',

});

export const jokePlugin = createPlugin({

id: 'joke-generator',

routes: {

root: jokeRouteRef,

},

});

export const JokePageComponent = jokePlugin.provide(

createPage({

element: <JokePage />,

title: 'Joke Generator',

path: '/joke-generator',

}),

);

### ****3. Create a React Component (****components/JokePage.tsx****)****

import React, { useEffect, useState } from 'react';

export const JokePage = () => {

const [joke, setJoke] = useState('');

const fetchJoke = () => {

fetch('https://official-joke-api.appspot.com/random\_joke')

.then(response => response.json())

.then(data => setJoke(`${data.setup} - ${data.punchline}`));

};

useEffect(() => {

fetchJoke();

}, []);

return (

<div>

<h1>Random Joke</h1>

<p>{joke}</p>

<button onClick={fetchJoke}>Get Another Joke</button>

</div>

);

};

### ****4. Register the plugin in**** app.tsx

Modify packages/app/src/App.tsx:

import { JokePageComponent } from '@backstage/plugin-joke-generator';

const routes = (

<FlatRoutes>

<Route path="/joke-generator" element={<JokePageComponent />} />

</FlatRoutes>

);

### ****5. Run Backstage****

yarn dev

**Visit:** http://localhost:3000/joke-generator

# **3. To-Do List Plugin**

### ****How to Develop****

### ****1. Generate a new plugin****

yarn backstage-cli create-plugin --name todo-list

### ****2. Modify the frontend plugin (****packages/plugins/todo-list/src/index.ts****)****

import { createPlugin, createRouteRef } from '@backstage/core-plugin-api';

import { TodoPage } from './components/TodoPage';

export const todoRouteRef = createRouteRef({

id: 'todo-list',

});

export const todoPlugin = createPlugin({

id: 'todo-list',

routes: {

root: todoRouteRef,

},

});

export const TodoPageComponent = todoPlugin.provide(

createPage({

element: <TodoPage />,

title: 'To-Do List',

path: '/todo-list',

}),

);

### ****3. Create a React Component (****components/TodoPage.tsx****)****

import React, { useState } from 'react';

export const TodoPage = () => {

const [tasks, setTasks] = useState<string[]>([]);

const [newTask, setNewTask] = useState('');

const addTask = () => {

if (newTask.trim() !== '') {

setTasks([...tasks, newTask]);

setNewTask('');

}

};

return (

<div>

<h1>To-Do List</h1>

<input

type="text"

value={newTask}

onChange={(e) => setNewTask(e.target.value)}

/>

<button onClick={addTask}>Add Task</button>

<ul>

{tasks.map((task, index) => (

<li key={index}>{task}</li>

))}

</ul>

</div>

);

};

### ****4. Register the plugin in**** app.tsx

Modify packages/app/src/App.tsx:

import { TodoPageComponent } from '@backstage/plugin-todo-list';

const routes = (

<FlatRoutes>

<Route path="/todo-list" element={<TodoPageComponent />} />

</FlatRoutes>

);

### ****5. Run Backstage****

yarn dev

**Visit:** http://localhost:3000/todo-list

# **4. Counter Plugin**

### ****How to Develop****

### ****1. Generate a new plugin****

yarn backstage-cli create-plugin --name counter

### ****2. Modify the frontend plugin (****packages/plugins/counter/src/index.ts****)****

import { createPlugin, createRouteRef } from '@backstage/core-plugin-api';

import { CounterPage } from './components/CounterPage';

export const counterRouteRef = createRouteRef({

id: 'counter',

});

export const counterPlugin = createPlugin({

id: 'counter',

routes: {

root: counterRouteRef,

},

});

export const CounterPageComponent = counterPlugin.provide(

createPage({

element: <CounterPage />,

title: 'Counter',

path: '/counter',

}),

);

### ****3. Create a React Component (****components/CounterPage.tsx****)****

import React, { useState } from 'react';

export const CounterPage = () => {

const [count, setCount] = useState(0);

return (

<div>

<h1>Counter</h1>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

<button onClick={() => setCount(count - 1)}>Decrement</button>

<button onClick={() => setCount(0)}>Reset</button>

</div>

);

};

### ****4. Register the plugin in**** app.tsx

Modify packages/app/src/App.tsx:

import { CounterPageComponent } from '@backstage/plugin-counter';

const routes = (

<FlatRoutes>

<Route path="/counter" element={<CounterPageComponent />} />

</FlatRoutes>

);

### ****5. Run Backstage****

yarn dev

**Visit:** http://localhost:3000/counter

# **5. API Data Fetcher Plugin**

### ****How to Develop****

### ****1. Generate a new plugin****

yarn backstage-cli create-plugin --name api-fetcher

### ****2. Modify the frontend plugin (****packages/plugins/api-fetcher/src/index.ts****)****

import { createPlugin, createRouteRef } from '@backstage/core-plugin-api';

import { ApiFetcherPage } from './components/ApiFetcherPage';

export const apiFetcherRouteRef = createRouteRef({

id: 'api-fetcher',

});

export const apiFetcherPlugin = createPlugin({

id: 'api-fetcher',

routes: {

root: apiFetcherRouteRef,

},

});

export const ApiFetcherPageComponent = apiFetcherPlugin.provide(

createPage({

element: <ApiFetcherPage />,

title: 'API Fetcher',

path: '/api-fetcher',

}),

);

### ****3. Create a React Component (****components/ApiFetcherPage.tsx****)****

import React, { useEffect, useState } from 'react';

export const ApiFetcherPage = () => {

const [data, setData] = useState('');

useEffect(() => {

fetch('https://api.publicapis.org/entries')

.then(response => response.json())

.then(data => setData(JSON.stringify(data.entries.slice(0, 5), null, 2)));

}, []);

return (

<div>

<h1>API Data Fetcher</h1>

<pre>{data}</pre>

</div>

);

};

### ****4. Register the plugin in**** app.tsx

Modify packages/app/src/App.tsx:

import { ApiFetcherPageComponent } from '@backstage/plugin-api-fetcher';

const routes = (

<FlatRoutes>

<Route path="/api-fetcher" element={<ApiFetcherPageComponent />} />

</FlatRoutes>

);

### ****5. Run Backstage****

yarn dev

**Visit:** http://localhost:3000/api-fetcher

# **1. Full Backstage Plugin (Frontend + Backend)**

## ****GitHub Repository Stats Plugin****

### ✅ ****Features****

* Fetches repository statistics from GitHub API
* Displays repo stars, forks, and open issues
* Has **both frontend & backend**

### ****🔹 Generate the Plugin (Full Plugin)****

Run this in your Backstage root directory:

yarn backstage-cli create-plugin --name github-repo-stats

This will create a new plugin under plugins/github-repo-stats/.

## ****Frontend Code (React Component)****

Edit plugins/github-repo-stats/src/components/GithubRepoStatsPage.tsx:

import React, { useEffect, useState } from 'react';

export const GithubRepoStatsPage = () => {

const [repoStats, setRepoStats] = useState<any>(null);

useEffect(() => {

fetch('/api/github-repo-stats')

.then(response => response.json())

.then(data => setRepoStats(data));

}, []);

return (

<div>

<h1>GitHub Repository Stats</h1>

{repoStats ? (

<ul>

<li>🌟 Stars: {repoStats.stars}</li>

<li>🍴 Forks: {repoStats.forks}</li>

<li>🐛 Open Issues: {repoStats.open\_issues}</li>

</ul>

) : (

<p>Loading...</p>

)}

</div>

);

};

## ****Backend Code (Express API)****

Edit plugins/github-repo-stats-backend/src/service/router.ts:

import express from 'express';

import fetch from 'node-fetch';

export const createRouter = (): express.Router => {

const router = express.Router();

router.get('/', async (\_, res) => {

const response = await fetch(

'https://api.github.com/repos/backstage/backstage'

);

const data = await response.json();

res.json({

stars: data.stargazers\_count,

forks: data.forks\_count,

open\_issues: data.open\_issues\_count,

});

});

return router;

};

## ****Register the Plugin in**** packages/app/src/App.tsx

import { GithubRepoStatsPage } from '@backstage/plugin-github-repo-stats';

const routes = (

<FlatRoutes>

<Route path="/github-repo-stats" element={<GithubRepoStatsPage />} />

</FlatRoutes>

);

### ****🔹 Run the Plugin****

1️⃣ Start Backstage:

yarn dev

2️⃣ Open:

http://localhost:3000/github-repo-stats

# **2. Frontend-Only Plugins**

## ✅ ****These do NOT require a backend****

Each one below is a **frontend-only plugin** (React-based UI).

### ****🔹 2.1 Counter Plugin****

#### ****Features****

* Simple Counter
* Tracks count in UI

#### ****Generate the Plugin****

yarn backstage-cli create-plugin --name counter

#### ****Modify Frontend (****components/CounterPage.tsx****)****

import React, { useState } from 'react';

export const CounterPage = () => {

const [count, setCount] = useState(0);

return (

<div>

<h1>Counter</h1>

<p>Current Count: {count}</p>

<button onClick={() => setCount(count + 1)}>➕ Increment</button>

<button onClick={() => setCount(count - 1)}>➖ Decrement</button>

<button onClick={() => setCount(0)}>🔄 Reset</button>

</div>

);

};

#### ****Register in**** App.tsx

import { CounterPageComponent } from '@backstage/plugin-counter';

const routes = (

<FlatRoutes>

<Route path="/counter" element={<CounterPageComponent />} />

</FlatRoutes>

);

#### ****Run & Open****

http://localhost:3000/counter

🎉 **Counter plugin is ready!**

### ****🔹 2.2 To-Do List Plugin****

#### ****Features****

* Add & track tasks

#### ****Generate the Plugin****

yarn backstage-cli create-plugin --name todo-list

#### ****Modify Frontend (****components/TodoPage.tsx****)****

import React, { useState } from 'react';

export const TodoPage = () => {

const [tasks, setTasks] = useState<string[]>([]);

const [newTask, setNewTask] = useState('');

const addTask = () => {

if (newTask.trim() !== '') {

setTasks([...tasks, newTask]);

setNewTask('');

}

};

return (

<div>

<h1>To-Do List</h1>

<input

type="text"

value={newTask}

onChange={(e) => setNewTask(e.target.value)}

/>

<button onClick={addTask}>Add Task</button>

<ul>

{tasks.map((task, index) => (

<li key={index}>{task}</li>

))}

</ul>

</div>

);

};

#### ****Register in**** App.tsx

import { TodoPageComponent } from '@backstage/plugin-todo-list';

const routes = (

<FlatRoutes>

<Route path="/todo-list" element={<TodoPageComponent />} />

</FlatRoutes>

);

#### ****Run & Open****

http://localhost:3000/todo-list

🎉 **To-Do List plugin is ready!**

### ****🔹 2.3 API Data Fetcher Plugin****

#### ****Features****

* Fetches data from a public API
* Displays JSON data

#### ****Generate the Plugin****

yarn backstage-cli create-plugin --name api-fetcher

#### ****Modify Frontend (****components/ApiFetcherPage.tsx****)****

import React, { useEffect, useState } from 'react';

export const ApiFetcherPage = () => {

const [data, setData] = useState('');

useEffect(() => {

fetch('https://api.publicapis.org/entries')

.then(response => response.json())

.then(data => setData(JSON.stringify(data.entries.slice(0, 5), null, 2)));

}, []);

return (

<div>

<h1>API Data Fetcher</h1>

<pre>{data}</pre>

</div>

);

};

#### ****Register in**** App.tsx

import { ApiFetcherPageComponent } from '@backstage/plugin-api-fetcher';

const routes = (

<FlatRoutes>

<Route path="/api-fetcher" element={<ApiFetcherPageComponent />} />

</FlatRoutes>

);

#### ****Run & Open****

http://localhost:3000/api-fetcher

🎉 **API Fetcher plugin is ready!**

## ****Takeaway****

| **Plugin Type** | | **Plugin Name** | | **Backend?** | **Features** | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ✅ Full Plugin | | **GitHub Repo Stats** | | ✅ Yes | Fetches GitHub stats (stars, forks, issues) | | |
| 🚀 Frontend-Only | | **Counter** | | ❌ No | Increment, Decrement, Reset | | |
| 🚀 Frontend-Only | | **To-Do List** | | ❌ No | Add, View tasks | | |
| 🚀 Frontend-Only | | **API Fetcher** | | ❌ No | Fetch API data | | |
| 🚀 Frontend-Only | **Random Jokes** | | ❌ No | | | Fetches random jokes from an API |

### ****How to Generate Each Plugin in Backstage (Frontend vs Backend vs Both?)****

When running yarn backstage-cli create-plugin --name <plugin-name>, Backstage will ask:

🔹 **"What kind of plugin would you like to create?"**  
1️⃣ **Just the plugin** (default, includes both frontend and backend)  
2️⃣ **Frontend plugin** (only frontend UI, no backend logic)  
3️⃣ **Backend plugin** (only backend API logic)

For the **above examples**, you should always select:  
✅ **Frontend plugin** (option 2)

This ensures that these plugins appear in the UI without needing backend logic.

Each plugin needs to be:

1. Created using the CLI
2. Registered in App.tsx
3. Accessed via http://localhost:3000/<plugin-route>

| **Plugin Name** | **CLI Command** | **Select Option** | **UI Route** |
| --- | --- | --- | --- |
| **Hello World** | yarn backstage-cli create-plugin --name hello-world | ✅ **Frontend plugin** | /hello-world |
| **To-Do List** | yarn backstage-cli create-plugin --name todo-list | ✅ **Frontend plugin** | /todo-list |
| **Counter** | yarn backstage-cli create-plugin --name counter | ✅ **Frontend plugin** | /counter |
| **API Data Fetcher** | yarn backstage-cli create-plugin --name api-data-fetcher | ✅ **Frontend plugin** | /api-data |
| **Random Jokes** | yarn backstage-cli create-plugin --name random-jokes | ✅ **Frontend plugin** | /random-jokes |

Example:

* **Hello World** → http://localhost:3000/hello-world
* **To-Do List** → http://localhost:3000/todo-list
* **Counter** → http://localhost:3000/counter
* **API Data Fetcher** → http://localhost:3000/api-data
* **Random Jokes** → http://localhost:3000/random-jokes

### ****Backend Plugins in Backstage: Setup, Run, and View in UI****

A **backend plugin** in Backstage provides API services, processing, and custom logic that power the frontend. Unlike frontend plugins, **backend plugins don’t have a direct UI**, but their functionality can be exposed through Backstage’s software catalog or custom frontend pages.

**How to Generate Backend Plugins in Backstage**

Run the command:

yarn backstage-cli create-plugin --name <plugin-name>

📌 **Select:** Backend plugin

After generating, the plugin will be available inside:

plugins/<plugin-name>/backend/

Then, you need to -

1️⃣ **Register the backend plugin** in packages/backend/src/index.ts  
2️⃣ **Start the backend** using yarn start-backend  
3️⃣ **Verify it in the UI using API calls or software catalog**

**Backend Plugin 1: System Health Check API**

✅ **Purpose:** A simple backend plugin that exposes a /health endpoint to check system status.

**1️⃣ Generate the Backend Plugin**

yarn backstage-cli create-plugin --name system-health

📌 **Select:** Backend plugin

**2️⃣ Modify the Backend Plugin**

Edit plugins/system-health/src/service/router.ts:

import { createRouter } from '@backstage/plugin-system-health-backend';

import { Router } from 'express';

import { PluginEnvironment } from '../types';

export default async function createPlugin(env: PluginEnvironment): Promise<Router> {

return createRouter(env);

}

Edit plugins/system-health/src/service/index.ts:

import express from 'express';

export const createRouter = async () => {

const router = express.Router();

router.get('/health', (\_, res) => {

res.json({ status: 'ok', message: 'Backend is healthy!' });

});

return router;

};

**3️⃣ Register in packages/backend/src/index.ts**

import { createRouter as createSystemHealthRouter } from '@backstage/plugin-system-health-backend';

async function main() {

const healthRouter = await createSystemHealthRouter(env);

apiRouter.use('/system-health', healthRouter);

}

This exposes the API at:

http://localhost:7007/api/system-health/health

**4️⃣ Start the Backend**

yarn start-backend

✅ Now, visiting http://localhost:7007/api/system-health/health should return:

{

"status": "ok",

"message": "Backend is healthy!"

}

**5️⃣ See It in UI (Software Catalog)**

1. **Modify catalog-info.yaml** in a registered entity:

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: system-health

description: Backend service for system health check

spec:

type: service

lifecycle: production

owner: team-devops

providesApis:

- system-health-api

---

apiVersion: backstage.io/v1alpha1

kind: API

metadata:

name: system-health-api

description: Health check service

spec:

type: openapi

lifecycle: production

owner: team-devops

definition: |

openapi: 3.0.0

info:

title: System Health API

version: 1.0.0

paths:

/health:

get:

summary: Get system health status

responses:

'200':

description: OK

1. **Register it in Backstage (http://localhost:3000/catalog-import)**
2. **Go to Backstage Catalog → View the API under “system-health”**

**Backend Plugin 2: Random Quote Generator API**

✅ **Purpose:** A backend plugin that provides random quotes via an API.

**1️⃣ Generate the Plugin**

yarn backstage-cli create-plugin --name random-quotes

📌 **Select:** Backend plugin

**2️⃣ Modify plugins/random-quotes/src/service/index.ts**

import express from 'express';

const quotes = [

"The best way to predict the future is to invent it.",

"Code is like humor. When you have to explain it, it’s bad.",

"Simplicity is the soul of efficiency."

];

export const createRouter = async () => {

const router = express.Router();

router.get('/quote', (\_, res) => {

const randomQuote = quotes[Math.floor(Math.random() \* quotes.length)];

res.json({ quote: randomQuote });

});

return router;

};

**3️⃣ Register in packages/backend/src/index.ts**

import { createRouter as createRandomQuotesRouter } from '@backstage/plugin-random-quotes-backend';

async function main() {

const quotesRouter = await createRandomQuotesRouter(env);

apiRouter.use('/random-quotes', quotesRouter);

}

Now available at:

http://localhost:7007/api/random-quotes/quote

**4️⃣ Start the Backend**

yarn start-backend

✅ Visiting http://localhost:7007/api/random-quotes/quote should return:

{

"quote": "Simplicity is the soul of efficiency."

}

**5️⃣ View in UI (Software Catalog)**

Modify catalog-info.yaml:

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: random-quotes

description: Backend service for random quotes

spec:

type: service

lifecycle: production

owner: team-backend

providesApis:

- random-quotes-api

---

apiVersion: backstage.io/v1alpha1

kind: API

metadata:

name: random-quotes-api

description: Provides a random quote

spec:

type: openapi

lifecycle: production

owner: team-backend

definition: |

openapi: 3.0.0

info:

title: Random Quotes API

version: 1.0.0

paths:

/quote:

get:

summary: Get a random quote

responses:

'200':

description: A random quote

🔹 **Register it in the UI (http://localhost:3000/catalog-import)**  
🔹 **View it in the API Catalog** 🎉

**Backend Plugin 3: Simple Weather API**

✅ **Purpose:** A backend plugin that provides dummy weather data.

**1️⃣ Generate the Plugin**

yarn backstage-cli create-plugin --name weather-api

📌 **Select:** Backend plugin

**2️⃣ Modify plugins/weather-api/src/service/index.ts**

import express from 'express';

const weatherData = {

city: "San Francisco",

temperature: "18°C",

condition: "Sunny"

};

export const createRouter = async () => {

const router = express.Router();

router.get('/weather', (\_, res) => {

res.json(weatherData);

});

return router;

};

**3️⃣ Register in Backend (packages/backend/src/index.ts)**

import { createRouter as createWeatherApiRouter } from '@backstage/plugin-weather-api-backend';

async function main() {

const weatherRouter = await createWeatherApiRouter(env);

apiRouter.use('/weather-api', weatherRouter);

}

API Endpoint:

http://localhost:7007/api/weather-api/weather

**4️⃣ Start the Backend**

yarn start-backend

✅ Visiting http://localhost:7007/api/weather-api/weather should return:

{

"city": "San Francisco",

"temperature": "18°C",

"condition": "Sunny"

}

1️⃣ **System Health API** - /api/system-health/health  
2️⃣ **Random Quotes API** - /api/random-quotes/quote  
3️⃣ **Weather API** - /api/weather-api/weather

## ****TechDocs - Creating and Viewing TechDocs in Backstage****

## ****What Are TechDocs?****

TechDocs in Backstage is a documentation system that enables teams to store and render Markdown-based docs inside the Backstage UI. It is powered by **MkDocs**.

# **🔹 Example 1: Hello World TechDocs**

### ****📌 Step 1: Create TechDocs for "Hello World"****

1️⃣ Navigate to your Backstage repository and create a new directory:

mkdir backstage-techdocs-hello-world

cd backstage-techdocs-hello-world

2️⃣ Create a mkdocs.yml file inside the directory:

site\_name: Hello World Docs

site\_description: This is a simple TechDocs example.

nav:

- Home: index.md

3️⃣ Create an index.md file with content:

# Welcome to Hello World Docs!

This is a simple TechDocs example.

### ****📌 Step 2: Register TechDocs in Backstage****

1️⃣ Commit and push your files to GitHub/GitLab.  
2️⃣ Open **Backstage UI** → **"Create..."** → **"Register Existing Component"**  
3️⃣ Provide the GitHub URL of the mkdocs.yml, e.g.:

https://github.com/my-org/backstage-techdocs-hello-world/blob/main/mkdocs.yml

4️⃣ Click **"Analyze"** → **"Import"**.

### ****📌 Step 3: View TechDocs in Backstage****

* Navigate to **"Docs"** in Backstage UI:

http://localhost:3000/docs

* Click **"Hello World Docs"** to view the documentation.

# **🔹 Example 2: API Documentation**

### ****📌 Step 1: Create API Docs****

1️⃣ Create an API TechDocs folder:

mkdir backstage-api-docs

cd backstage-api-docs

2️⃣ Add mkdocs.yml:

site\_name: API Documentation

nav:

- Overview: index.md

- Endpoints: endpoints.md

3️⃣ Create index.md:

# API Documentation

This is an example of API documentation using TechDocs.

4️⃣ Create endpoints.md:

# API Endpoints

- `/users`: Get all users

- `/orders`: Get all orders

### ****📌 Step 2: Register and View in Backstage****

Repeat the same **"Register TechDocs"** process in Backstage UI.

📍 **View in UI** → **http://localhost:3000/docs** → **API Documentation**

# **🔹 Example 3: CI/CD Pipeline Documentation**

### ****📌 Step 1: Create CI/CD Docs****

1️⃣ Create a directory:

mkdir backstage-ci-cd-docs

cd backstage-ci-cd-docs

2️⃣ Create mkdocs.yml:

site\_name: CI/CD Pipeline Docs

nav:

- Overview: index.md

- Build Process: build.md

3️⃣ Create index.md:

# CI/CD Pipeline Documentation

This document explains how our CI/CD pipeline is structured.

4️⃣ Create build.md:

# Build Process

- Step 1: Code is committed

- Step 2: Build triggers

- Step 3: Tests are run

- Step 4: Deployment happens

📍 **View in UI** → **http://localhost:3000/docs** → **CI/CD Pipeline Docs**

# **🔹 Example 4: Onboarding Guide**

### ****📌 Step 1: Create Onboarding Docs****

1️⃣ Create an onboarding directory:

mkdir backstage-onboarding-docs

cd backstage-onboarding-docs

2️⃣ Add mkdocs.yml:

site\_name: Onboarding Guide

nav:

- Welcome: index.md

- Setup: setup.md

3️⃣ Create index.md:

# Welcome to Onboarding Guide

New to our project? Follow these steps to get started.

4️⃣ Create setup.md:

# Setup Instructions

1. Install dependencies

2. Clone the repo

3. Run the project

📍 **View in UI** → **http://localhost:3000/docs** → **Onboarding Guide**

# **🔹 Example 5: Database Schema Docs**

### ****📌 Step 1: Create Database Docs****

1️⃣ Create a db-docs directory:

mkdir backstage-db-docs

cd backstage-db-docs

2️⃣ Add mkdocs.yml:

site\_name: Database Schema

nav:

- Overview: index.md

- Tables: tables.md

3️⃣ Create index.md:

# Database Schema Documentation

This document explains our database structure.

4️⃣ Create tables.md:

# Tables

- \*\*Users\*\*: Stores user data

- \*\*Orders\*\*: Stores order information

📍 **View in UI** → **http://localhost:3000/docs** → **Database Schema**

## ****Summary: How to View TechDocs in Backstage UI****

| **Step** | **Action** | **Backstage UI Section** |
| --- | --- | --- |
| **1** | Create a TechDocs folder | mkdir backstage-techdocs |
| **2** | Add mkdocs.yml & .md files | Write documentation |
| **3** | Push to GitHub/GitLab | Store the docs online |
| **4** | Register in Backstage | /catalog-import |
| **5** | View in Docs Section | /docs |

**APIs** - Backstage allows you to define and register APIs in the **Software Catalog**. APIs in Backstage can be REST, GraphQL, OpenAPI, or gRPC-based services.

## ****How to Add an API in Backstage****

1️⃣ **Create an api.yaml file (API Definition)**  
2️⃣ **Register it in the Software Catalog (http://localhost:3000/catalog-import)**  
3️⃣ **View API in the Catalog UI**  
4️⃣ **Test API using API Explorer**

# **🔹 API 1: Hello World API**

### ****1️⃣ Define API in**** api.yaml

apiVersion: backstage.io/v1alpha1

kind: API

metadata:

name: hello-world-api

description: A simple Hello World API

spec:

type: openapi

lifecycle: production

owner: team-api

definition: |

openapi: 3.0.0

info:

title: Hello World API

version: 1.0.0

paths:

/hello:

get:

summary: Returns a simple greeting

responses:

'200':

description: A simple greeting

content:

application/json:

schema:

type: object

properties:

message:

type: string

### ****2️⃣ Register API in Backstage****

Go to **Backstage UI → Catalog → Register Component**  
🔹 **Paste API URL or local file (api.yaml)**

### ****3️⃣ Start the Backend****

yarn start-backend

### ****4️⃣ Test API****

<http://localhost:7007/api/hello-world/hello>

output –

{

"message": "Hello, Backstage!"

}

### ****5️⃣ View in Backstage UI****

Go to **Backstage Catalog → APIs → hello-world-api** 🎉

# **🔹 API 2: User Profile API**

### ****1️⃣ Define API in**** api.yaml

apiVersion: backstage.io/v1alpha1

kind: API

metadata:

name: user-profile-api

description: API to fetch user profile details

spec:

type: openapi

lifecycle: production

owner: team-user

definition: |

openapi: 3.0.0

info:

title: User Profile API

version: 1.0.0

paths:

/user/{id}:

get:

summary: Get user profile by ID

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

'200':

description: User profile details

content:

application/json:

schema:

type: object

properties:

id:

type: string

name:

type: string

email:

type: string

### ****2️⃣ Register API in Backstage****

Go to **Backstage UI → Catalog → Register Component**  
🔹 **Paste API URL or local file (api.yaml)**

### ****3️⃣ Test API****

http://localhost:7007/api/user-profile/user/123

{

"id": "123",

"name": "John Doe",

"email": "john@example.com"

}

### ****4️⃣ View in Backstage UI****

Go to **Backstage Catalog → APIs → user-profile-api** 🎉

# **🔹 API 3: Weather API**

### ****1️⃣ Define API in**** api.yaml

apiVersion: backstage.io/v1alpha1

kind: API

metadata:

name: weather-api

description: Provides current weather details

spec:

type: openapi

lifecycle: production

owner: team-weather

definition: |

openapi: 3.0.0

info:

title: Weather API

version: 1.0.0

paths:

/weather/{city}:

get:

summary: Get weather by city

parameters:

- name: city

in: path

required: true

schema:

type: string

responses:

'200':

description: Weather details

content:

application/json:

schema:

type: object

properties:

city:

type: string

temperature:

type: string

condition:

type: string

### ****2️⃣ Test API****

http://localhost:7007/api/weather-api/weather/London

{

"city": "London",

"temperature": "15°C",

"condition": "Cloudy"

}

# **🔹 API 4: To-Do List API**

✅ **Purpose:** Manages tasks in a to-do list.

### ****1️⃣ Define API in**** api.yaml

apiVersion: backstage.io/v1alpha1

kind: API

metadata:

name: todo-api

description: API for managing a to-do list

spec:

type: openapi

lifecycle: production

owner: team-todo

definition: |

openapi: 3.0.0

info:

title: To-Do API

version: 1.0.0

paths:

/tasks:

get:

summary: Get all tasks

responses:

'200':

description: List of tasks

content:

application/json:

schema:

type: array

items:

type: object

properties:

id:

type: string

task:

type: string

status:

type: string

### ****2️⃣ Test API****

http://localhost:7007/api/todo-api/tasks

[

{ "id": "1", "task": "Complete Backstage setup", "status": "In Progress" },

{ "id": "2", "task": "Test API catalog", "status": "Pending" }

]

# **🔹 API 5: Random Joke API**

### ****1️⃣ Define API in**** api.yaml

apiVersion: backstage.io/v1alpha1

kind: API

metadata:

name: joke-api

description: API that provides random programming jokes

spec:

type: openapi

lifecycle: production

owner: team-fun

definition: |

openapi: 3.0.0

info:

title: Joke API

version: 1.0.0

paths:

/joke:

get:

summary: Get a random programming joke

responses:

'200':

description: A random joke

content:

application/json:

schema:

type: object

properties:

joke:

type: string

### ****2️⃣ Test API****

http://localhost:7007/api/joke-api/joke

✅ Expected Response:

{

"joke": "Why do programmers prefer dark mode? Because light attracts bugs!"

}

| **API Name** | **Purpose** | **Endpoint** |
| --- | --- | --- |
| Hello World API | Returns a greeting | /hello-world/hello |
| User Profile API | Fetches user details | /user-profile/user/{id} |
| Weather API | Provides weather data | /weather-api/weather/{city} |
| To-Do API | Manages a task list | /todo-api/tasks |
| Joke API | Returns a random programming joke | /joke-api/joke |

## ****Step-by-Step Guide: Registering and Testing APIs in Backstage UI****

## ****📌 Step 1: Open the Backstage UI****

1️⃣ Start Backstage by running:

yarn dev

2️⃣ Open **Backstage in your browser** at:

http://localhost:3000

## ****📌 Step 2: Register an API in Backstage Catalog****

To register an API, follow these steps:

### ****1️⃣ Open the Catalog Import Page****

* In the Backstage UI, go to **"Create..."** (or directly to /catalog-import):

http://localhost:3000/catalog-import

### ****2️⃣ Provide the API Definition URL or YAML File****

* **If the API is hosted on GitHub, GitLab, or a local server:**  
  👉 Paste the **API definition URL**, e.g.,

https://raw.githubusercontent.com/my-org/backstage-apis/main/api.yaml

* **If you have a local file (api.yaml):**  
  👉 Click **"Choose File"** and upload the YAML file.

### ****3️⃣ Click "Analyze"****

* Backstage will **validate** the API definition and extract details.
* If everything looks correct, click **"Import"**.

### ****4️⃣ Confirm API Registration****

* You will see a **"Component registered successfully"** message.
* Click **"View in Catalog"** to go to the API's details page.

## ****📌 Step 3: View the API in Backstage UI****

Once registered, you can find the API in the **Backstage Catalog**.

### ****1️⃣ Go to the API Catalog Page****

* Navigate to **"APIs"** section:

http://localhost:3000/catalog

* You will see a list of all registered APIs.

### ****2️⃣ Click on Your API****

* Click on the API (e.g., hello-world-api).
* It will open the API **Overview Page**.

### ****3️⃣ View API Details****

* The **API Overview Page** shows:
  + **API Name**
  + **Owner**
  + **Definition Type (REST, GraphQL, etc.)**
  + **API Documentation**
  + **Associated Components & Teams**

## ****📌 Step 4: Test the API Using Backstage API Explorer****

Backstage has an **API Explorer** that allows you to test APIs.

### ****1️⃣ Open the API Explorer****

* Inside the API details page, look for the **"API Definition"** section.
* Click **"View API in API Explorer"**.

### ****2️⃣ Select an Endpoint****

* You will see a list of available endpoints (e.g., /hello).
* Click on an endpoint to expand its details.

### ****3️⃣ Test API with a Request****

* Click **"Try it out"**.
* Enter any required parameters (e.g., user ID, city name).
* Click **"Execute"**.

### ****4️⃣ View API Response****

* The response will be displayed in the API Explorer.
* Example response for hello-world-api:

json

CopyEdit

{

"message": "Hello, Backstage!"

}

| **Step** | **Action** | **Backstage UI Section** |
| --- | --- | --- |
| **1** | Start Backstage | http://localhost:3000 |
| **2** | Open Catalog Import | /catalog-import |
| **3** | Provide API Definition (URL or YAML) | Upload API |
| **4** | View API in Catalog | /catalog → APIs Section |
| **5** | Click API Name | Opens API Details Page |
| **6** | Use API Explorer | Test API Endpoints |