



# Nx を完全に理解した

エンジニア達の「完全に理解した」Talk #61

# 自己紹介

akihiko.KIfigure a.k.a グレさん

恵比寿とグンマー帝国の2拠点でフロントエンド開発をしています。



# 個人的推しOSS

※アルファベット順



Angular



CHIRIMEN Open Hardware



MDN Web Docs

# 個人的推しOSS

※アルファベット順



Nx



Rust

# 今日テーマ



# 今日お話しする事

- Nx の簡単な歴史
- Nx のメリット
- Nx での構築例
- Nx の便利コマンド

# ● Nx の簡単な歴史

# ● Nx の簡単な歴史 part.1

2017 年

Nrwl (ナウラル) によって開発が開始される。

Nrwl は元 Google の Angular チームメンバーによって設立された企業。

初期は Angular プロジェクトのための拡張ツールとして設計された。

2018 年

バージョン 1.0 がリリース。モノレポの管理機能が本格的に導入される。

Bazel (Google 製のビルドツール) からインスピレーションを得たキャッシュ機能が注目される。

2019 年

Angular 以外のフレームワーク (React、Node.js など) にも対応を開始。

モノレポ管理ツールとしての汎用性が強化される。



# ● Nx の簡単な歴史 part.2

2020 年

Nx Cloud が発表され、分散キャッシュや CI/CD の高速化が可能に。  
開発者体験（DX）を重視した CLI や GUI（Nx Console）が進化。

2021年

JavaScript/TypeScript 以外の言語（Rust、Go、Python など）のサポートが追加される。  
コミュニティが拡大し、Nx がモノレポツールのデファクトスタンダードの一つとなる。

# ● Nx の簡単な歴史 part.3

2022 年

React、Vue、Angular、Svelte など、主要なフロントエンドフレームワーク向けのプラグインが充実。


API ドキュメントやコード生成ツールがさらに強化される。

2023 年以降

バージョン 20.x がリリースされ、大規模プロジェクトのスケラビリティとパフォーマンスが向上。

Tauri や Rust のようなクロスプラットフォーム開発ツールとの統合事例が増加。コミュニティとエコシステムの成長が継続。

Nx は、効率的なビルド、テスト、デプロイを可能にしつつ、拡張性と開発体験の向上を追求し続けています。



● Nx のメリット

# ● Nx メリット part.1

## 1. プロジェクトの管理が楽になる

1つのリポジトリで複数のアプリやライブラリを効率よく管理できます。例えば、どの部分がどこに影響するかを Nx が自動で調べてくれるので、「どれを修正すればいいの？」と迷うことが少なくなります。

## 2. 作業が早くなる

Nx は変更したところだけを素早くビルドしたりテストしたりしてくれます。全部をやり直す必要がないので、作業時間が短縮できます。

## 3. コードを書くときの便利ツールがある

**Nx Console:** 簡単なボタン操作で新しいプロジェクトを作ったり設定を変更できます。コマンドを覚えなくても使えます！

**CLI:** コマンド1つでプロジェクトやファイルを作れるので、手作業でフォルダを作る必要がなくなります。

# ● Nx メリット part.2

## 4. コードを分かりやすく整理できる

プロジェクトを小さな部品（ライブラリ）に分けることで、コードが見やすくなり、再利用しやすくなります。

## 5. 色々な言語やフレームワークに対応

Nx は **A**ngular だけじゃなく、React や Vue、Node.js など他のフレームワークや言語でも使えます。一緒に使っても問題ありません！

## 6. チームでの開発がスムーズに

コードの書き方やプロジェクトの構造を統一できるので、誰が作業しても同じように進められます。

自動化や最適化が簡単にできるので、大人数のチームでも作業がスムーズです。

要するに、「**Nx を使うと、大きなプロジェクトでも管理が簡単になり、効率よく開発できる**」ということです。

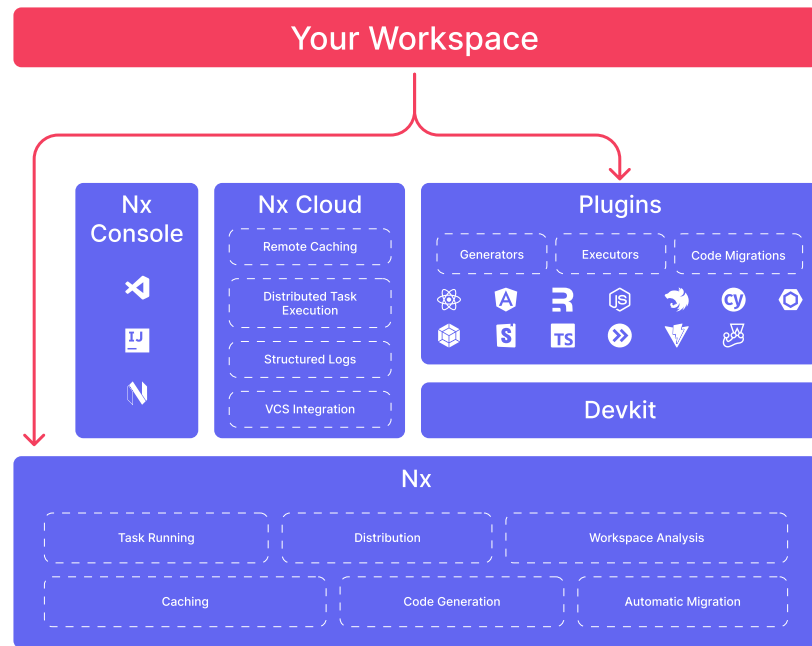
特に複数人や複数プロジェクトで開発している人にとって、とても助かるツールです！

# ● Nx メリット part.3



公式ページ









































動画リンク



<https://nx.dev/getting-started/why-nx> より引用

# ● Nx メリット part.4

## Official Packages Reference

|  |   |   |   |  |
|--|---|---|---|--|
|  Nx                  |  Workspace         |  Devkit                |  Angular               |  Cypress                    |
|  Detox               |  Esbuild           |  Eslint                |  Eslint Plugin         |  Expo                       |
|  Express             |  Gradle            |  Jest                  |  Js                    |  Module Federation          |
|  Nest                |  Next              |  Node                  |  Nuxt                  |  Playwright                 |
|  Plugin              |  React             |  React Native          |  Remix                 |  Rollup                     |
|  Rsbuild             |  Rspack            |  Storybook             |  Vite                  |  Vue                        |
|  Web                 |  Webpack           |  Powerpack Azure Cache |  Powerpack Conformance |  Powerpack Enterprise Cloud |
|  Powerpack Gcs Cache |  Powerpack License |  Powerpack Owners      |  Powerpack S3 Cache    |  Powerpack Shared Fs Cache  |



# ● Nx メリット part.5

## Nx Plugin Registry (一部抜粋)

@nx/angular

The Nx Plugin for Angular contains executors, generators, and utilities for managing Angular applications and libraries within an Nx workspace. It provides: - Integration with libraries such as...

🕒 2025-01-07 ↓ 1M ☆ 24k

Nx Open Source

@nx/cypress

The Nx Plugin for Cypress contains executors and generators allowing your workspace to use the powerful Cypress integration testing capabilities.

🕒 2025-01-07 ↓ 2M ☆ 24k

Nx Open Source

@nx/detox

The Nx Plugin for Detox contains executors and generators for allowing your workspace to use the powerful Detox integration testing capabilities.

🕒 2025-01-07 ↓ 114k ☆ 24k

Nx Open Source

@nx/devkit

The Nx Devkit is used to customize Nx for different technologies and use cases. It contains many utility functions for reading and writing files, updating configuration, working with Abstract Synt...

🕒 2025-01-07 ↓ 13M ☆ 24k

Nx Open Source

@nx/esbuild

The Nx Plugin for esbuild contains executors and generators that support building applications using esbuild

🕒 2025-01-07 ↓ 891k ☆ 24k

Nx Open Source

@nx/eslint

The ESLint plugin for Nx contains executors, generators and utilities used for linting JavaScript/TypeScript projects within an Nx workspace.

🕒 2025-01-07 ↓ 4M ☆ 24k

Nx Open Source

@nx/eslint-plugin

The eslint-plugin package is an ESLint plugin that contains a collection of recommended ESLint rule configurations which you can extend from in your own ESLint configs, as well as an Nx-...

🕒 2025-01-07 ↓ 3M ☆ 24k

Nx Open Source

@nx/expo

The Expo Plugin for Nx contains executors and generators for managing and developing an expo application within your workspace. For example, you can directly build for different targ...

🕒 2025-01-07 ↓ 61k ☆ 24k

Nx Open Source

@nx/express

The Nx Plugin for Express contains executors and generators for allowing your workspace to create powerful Express Node applications and APIs.

🕒 2025-01-07 ↓ 322k ☆ 24k

Nx Open Source





# ● Nx メリット part.6

これだけ覚えれば、オッケー！！！！

## 80/20 アプローチ (v18.x 時点)

Nx は、アプリケーション (apps) とライブラリ (libs) という2つの要素で構成されています。アプリケーションは複数のライブラリを組み合わせで構築される仕組みです。この構造を利用して、**80% のロジックを libs に集約し、20% のロジックを apps に配置する**のが 80/20 アプローチです。

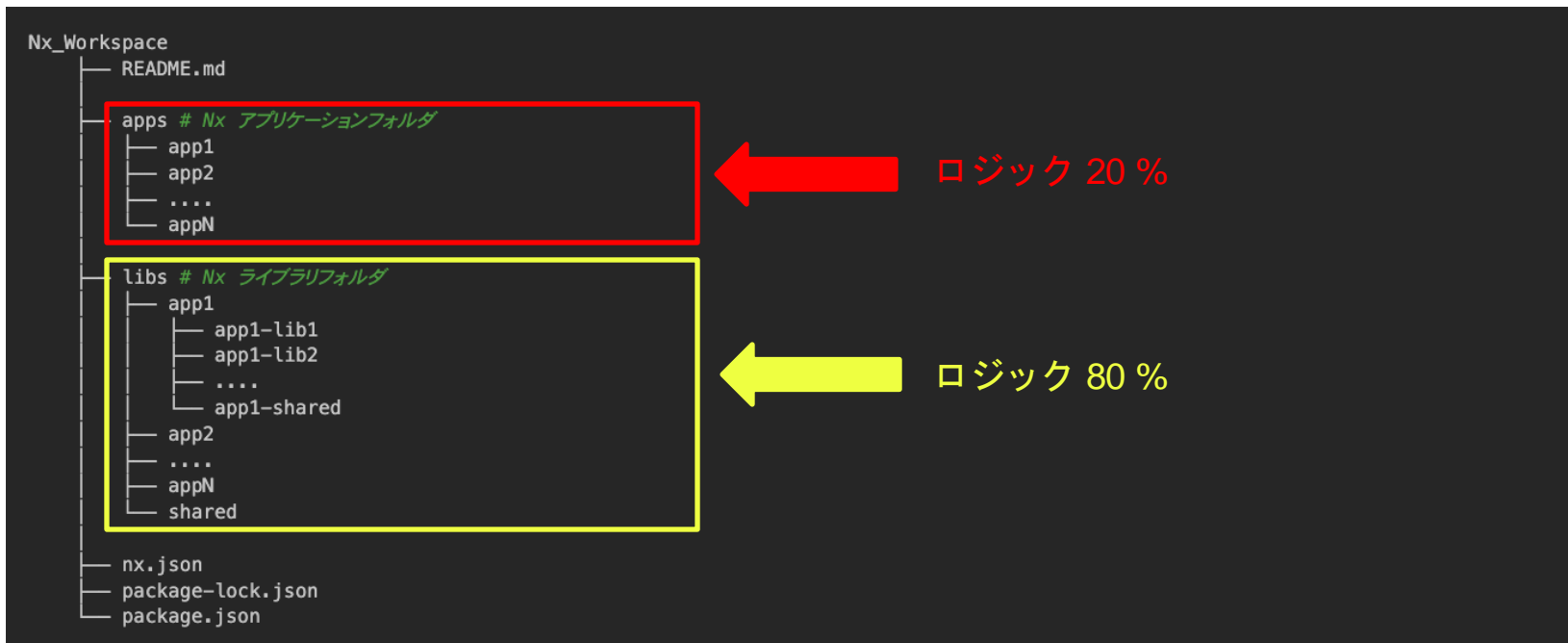
※現在のドキュメントでは削除されています。



このアプローチの魅力は、コードの再利用性を高め、CIの効率を大幅に向上させる点です。特に、Nxの「影響範囲キャッシュ」と組み合わせることで、必要な部分だけビルドやテストを行うことができます。

# ● Nx メリット part.7

## 80/20 アプローチディレクトリイメージ



# ● Nx での構築例

# 前提条件

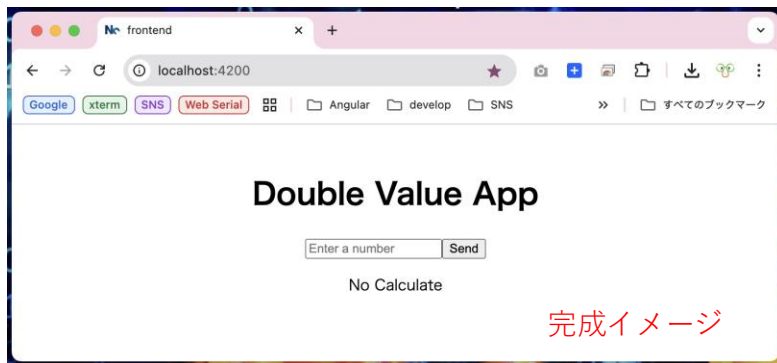
- Angular, Npm, Node の環境が構築済みであること
- Rustの環境が構築済みであること

Angular, Rust を使った

Nx ワークスペースの構築例

# 仕様概要

- 画面（Angular）で、数値を入力
- 画面（Angular）のボタンをクリックした時にAPI（Rust）呼出
- API（Rust）はセットされた値を 2 倍に計算
- API（Rust）は計算結果を画面（Angular）に返却
- 画面（Angular）は、API の計算結果を画面に表示



# ● Nx での構築例 part.1

リポジトリ

[gurezo/completely-understood-vol61](https://github.com/gurezo/completely-understood-vol61)



# ● Nx での構築例 part.2

## PR: nx ワークスペースの作成と同時に Angular プロジェクト作成

コマンド例

```
$ npx create-nx-workspace@latest completely-understood-vol61 --preset=angular --appName=frontend
```

The screenshot shows a GitHub pull request titled 'create nx and angular #1' from user 'gprico'. The PR is merged into the 'main' branch. The diff shows changes to two files: 'nx.json' and 'nx.json.backup'. The 'nx.json' file contains configuration for an Angular workspace with a single application named 'frontend'. The configuration includes details for the application's name, root, source files, and test files. The 'nx.json.backup' file is a duplicate of the original 'nx.json'.

```
1 - # Nx configuration, see https://nx.dev
2 -
3 - @nx/workspace:
4 -   root: true
5 -
6 -   defaultProject: frontend
7 -   defaultBase: main
8 -   defaultOutputFolder: dist
9 -   trackComponentChanges: true
10 -   trackTrailingSlash: true
11 -
12 - (cli)
13 -   nxJsonProject: frontend
14 -   trackTrailingSlash: false
15
16 -
17 -
18 -
19 -
20 -
21 -
22 -
23 -
24 -
25 -
26 -
27 -
28 -
29 -
30 -
31 -
32 -
33 -
34 -
35 -
36 -
37 -
38 -
39 -
40 -
41 -
42 -
43 -
44 -
45 -
46 -
47 -
48 -
49 -
50 -
51 -
52 -
53 -
54 -
55 -
56 -
57 -
58 -
59 -
60 -
61 -
62 -
63 -
64 -
65 -
66 -
67 -
68 -
69 -
70 -
71 -
72 -
73 -
74 -
75 -
76 -
77 -
78 -
79 -
80 -
81 -
82 -
83 -
84 -
85 -
86 -
87 -
88 -
89 -
90 -
91 -
92 -
93 -
94 -
95 -
96 -
97 -
98 -
99 -
100 -
```





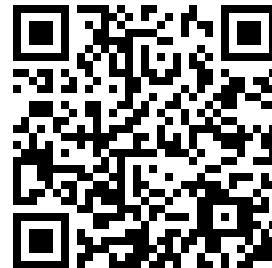
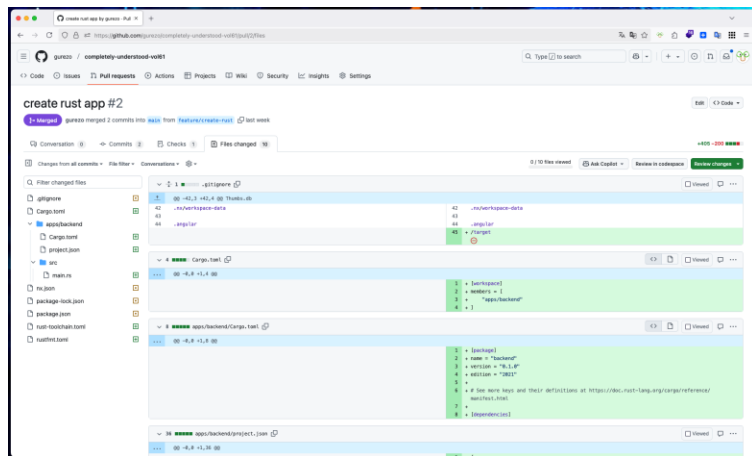
# ● Nx での構築例 part.3

## PR: nx ワークスペースに Rust プロジェクト作成

コマンド例

```
$ npm i @nxrs/cargo --save-dev # プラグイン追加
```

```
$ npx nx generate @nxrs/cargo:app backend # Rust プロジェクト作成
```



# ● Nx での構築例 part.4

## nx ワークスペース作成例

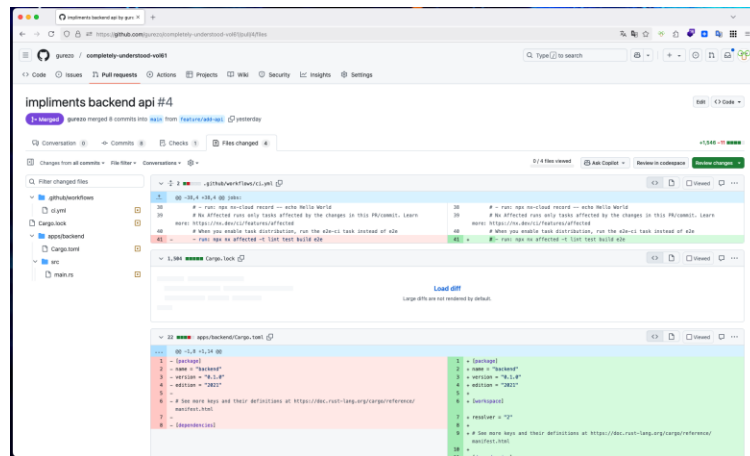
```
completely-understood-vol61
├── Cargo.lock
├── Cargo.toml
├── README.md
├── apps # Nx アプリケーションフォルダ
│   ├── backend # Rust プロジェクト
│   ├── frontend # Angular プロジェクト
│   └── frontend-e2e # Angular E2E プロジェクト
├── dist # Angular ビルド出力先
├── eslint.config.cjs
├── jest.config.ts
├── jest.preset.js
├── nx.json
├── package-lock.json
├── package.json
├── rust-toolchain.toml
├── rustfmt.toml
├── target # Rust ビルド出力先
└── tsconfig.base.json
```

# ● Nx での構築例 part.5

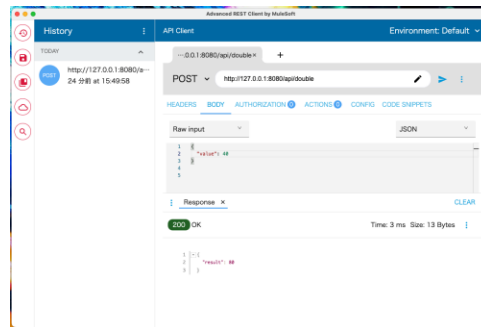
## PR:バックエンドの実装

### 動作確認 (Advanced Rest Client)

- Nx コマンド: `npm nx serve backend`
- URL: <http://127.0.0.1:8080/api/double>
- リクエストパラメータ: `{ "value": 40 }`
- レスポンス: `{ "result": 80 }`



A screenshot of a GitHub pull request titled "implements backend api #4". The interface shows a diff view of code changes. The left pane shows the file structure with folders for "githubworkflows", "src", and "main.ts". The right pane shows the code diff, with changes in "src/app.module.ts" and "src/main.ts". The diff highlights changes to the "app" module and the "main" function. The "app" module changes include adding "value" to the "data" object and updating the "value" property in the "data" object. The "main" function changes include updating the "value" property in the "data" object.



A screenshot of the Advanced REST Client interface. The "History" tab is active, showing a successful POST request to "http://127.0.0.1:8080/api/double" at 15:49:58. The request body is {"value": 40}. The response is {"result": 80}. The status is 200 OK, and the time taken is 3 ms. The size of the response is 13 Bytes.

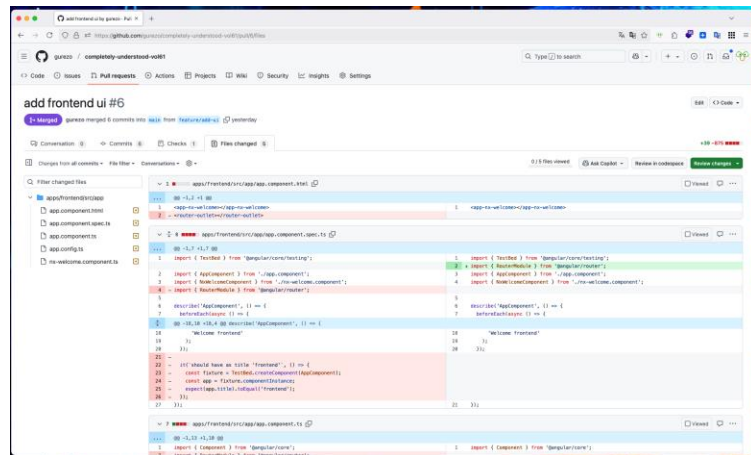
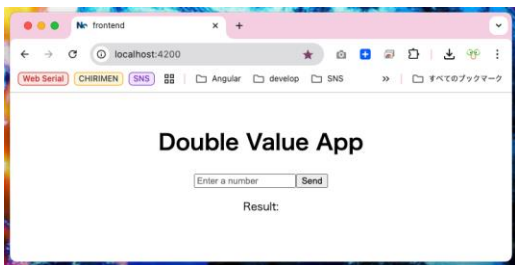


# ● Nx での構築例 part.6

## PR: フロントエンドの実装

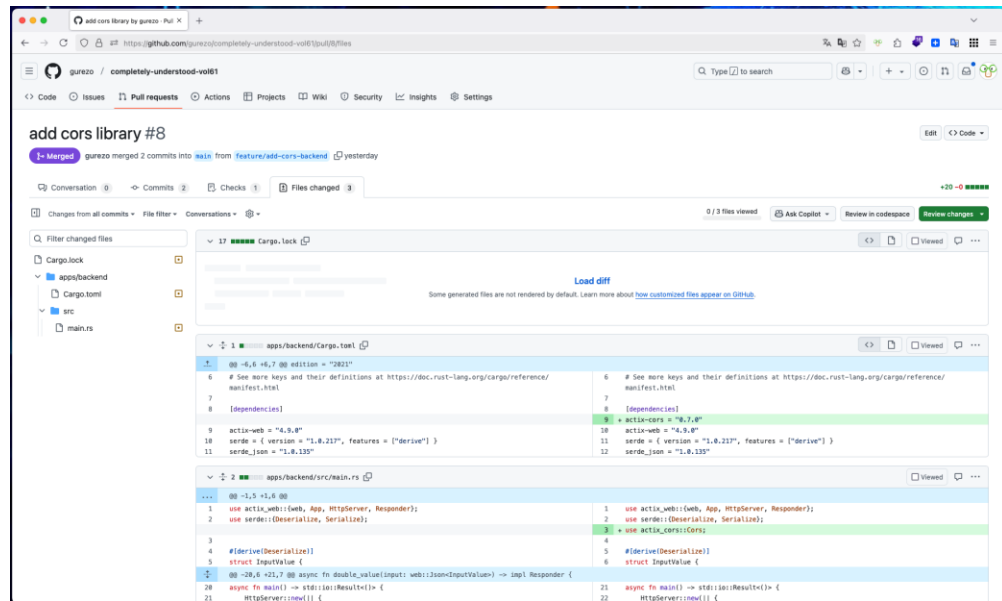
動作確認 (ローカルサーバー)

- Nx コマンド: `npx nx serve frontend`



# ● Nx での構築例 part.7

## PR: CORS ライブラリを追加実装



The screenshot shows a GitHub pull request titled "add cors library #8" merged into the main branch. The diff view is open, showing changes to two files: `apps/backend/cargo.toml` and `apps/backend/src/main.rs`. The `cargo.toml` diff shows the addition of a dependency for `actix-cors` and updates to the `actix-web` and `serde` versions. The `main.rs` diff shows the addition of `actix_cors::Cors` and its use in the `main` function to create a `Cors` object and pass it to the `HttpServer::new` method.

```
diff --git a/apps/backend/cargo.toml b/apps/backend/cargo.toml
index 4670821..7282111
--- a/apps/backend/cargo.toml
+++ b/apps/backend/cargo.toml
@@ -6,6 +6,7 @@ edition = "2021"
 # See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html
 [dependencies]
-actix-web = "4.9.0"
+actix-web = "4.9.0"
-serde = { version = "1.0.217", features = ["derive"] }
+serde = { version = "1.0.217", features = ["derive"] }
-serde_json = "1.0.130"
+serde_json = "1.0.130"
+actix-cors = "0.7.0"

diff --git a/apps/backend/src/main.rs b/apps/backend/src/main.rs
index 1521600..2170000
--- a/apps/backend/src/main.rs
+++ b/apps/backend/src/main.rs
@@ -1,5 +1,5 @@
 use actix_web::{web, App, HttpServer, Response};
+use actix_cors::Cors;
 use serde::{Deserialize, Serialize};

@@ -4,5 +4,5 @@
 # [derive(Deserialize)]
 struct InputValue {
@@ -20,6 +20,6 @@
 async fn main() -> std::io::Result<()> {
     HttpServer::new(|| {
@@ -21,1 +21,1 @@
         HttpServer::new(|| {
```



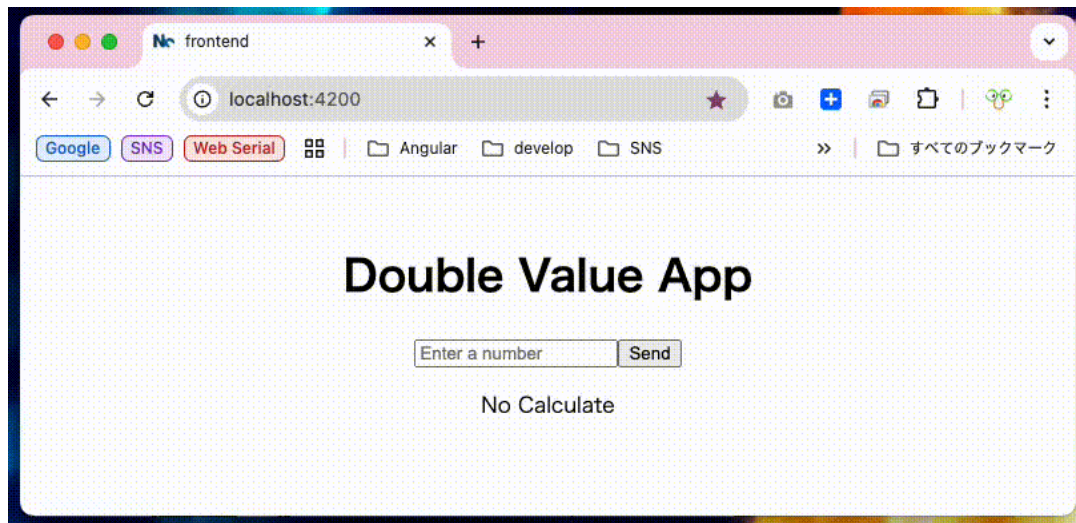
## CORS のエラー



# ● Nx での構築例 part.7

## 動作確認 (ローカル起動フロント & バック)

- Nx コマンド: `npx nx run-many --target=serve --projects=frontend,backend`



# ● Nx の便利コマンド

# ● Nx の便利コマンド part.1

nx ワークスペースの作成と同時に Angular プロジェクト作成

`create-nx-workspace@latest` ワークスペース名 `--preset=※1` `--appName=アプリ名`

※1: プリセット種類 ( React、Vue、Angular、Svelte )

nx プロジェクト複数同時起動

`nx run-many --target=実行ターゲット` `--projects=app1,app2....`



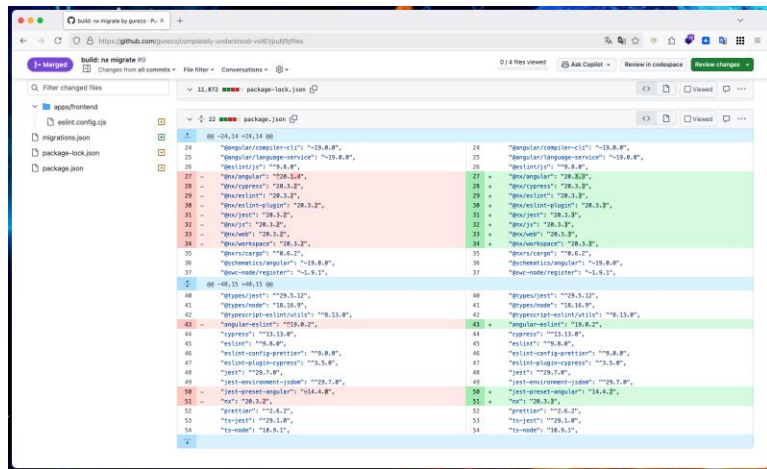
# ● Nx の便利コマンド part.2

nx ワークスペースマイグレーション (マイグレーションファイル作成)

`nx migrate latest`

nx ワークスペースマイグレーション (マイグレーションファイルを基に実行)

`nx migrate --run-migrations`



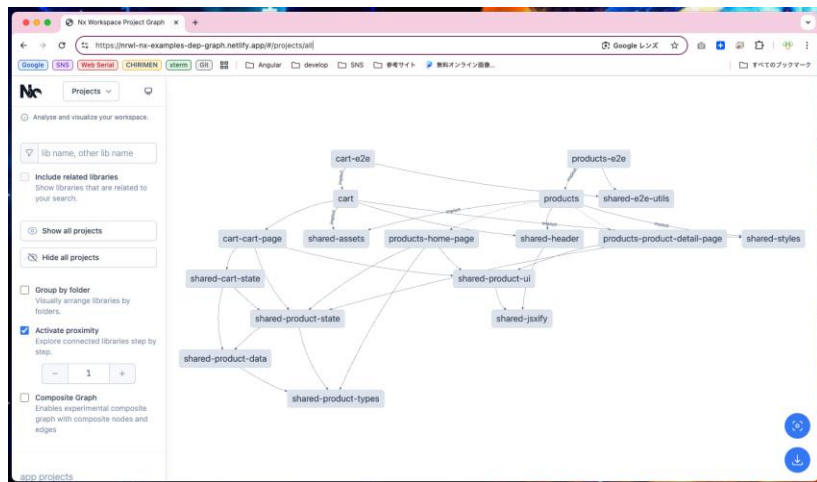
The screenshot shows a GitHub diff view for the file `package-lock.json`. The diff is comparing two versions of the file, with changes highlighted in green (additions) and red (deletions). The changes include updates to various dependencies and their versions, such as `@angular/compiler`, `ngx/cypress`, `ngx/eslint-plugin`, `ngx/jest`, `ngx/nx`, `ngx/workspaces`, `ngx/cargo`, `@storybook/angular`, `@storybook/register`, `@types/jest`, `@types/node`, `@types/react`, `@types/react-dom`, `@types/testing-library__jest-dom`, `eslint`, `eslint-config-prettier`, `eslint-plugin-cypress`, `jest`, `jest-environment-jsdom`, `jest-extended-angular`, `jest-extended`, `prettier`, `react`, and `react-dom`.



# ● Nx の便利コマンド part.3

nx ワークスペース内のプロジェクトの関係性を分かりやすく図で表示  
nx graph

<https://nrwl-nx-examples-dep-graph.netlify.app/#/projects/all>



# まとめ

Nx できるよ、にんげんだもの

ぐれを

# 質問

続編として、下記 A or B どちらが見たい／聞きたいですか？

Youtube のコメント欄に A or B どちらかポストしてください！

- A. **firebase** に今回のコードをデプロイする話
- B. **さくら VPS** に今回のコードをデプロイする話

視聴者の皆様と一緒に続編 & 登壇を決めていくスタイル

NEW



ご静聴ありがとうございました！