Taon.dev

v21

Framework / CLI / Cloud

10 Reasons Why Taon is THE FUTURE of Angular/NodeJS development

About me

First name: Dariusz

Last name: Filipiak

Location: Warsaw

/ darekf77

Experience in coding: 22 years

Experience in frontend: 13 years

Experience in TypeScript/Angular: 10 years

Origins of taon

9 years ago, I wanted to build

a backend + frontend website (a simple one)

using **Angular + NodeJS + TypeScript + TypeORM**.

I wanted to build something

at the highest possible abstraction level

with the best tools available.

New idea

After exploring various frameworks and solutions,

I realized that a new kind of

framework / CLI / cloud was needed.

I also had a strong feeling that building everything

around **Visual Studio Code** and **TypeScript**

was the right direction.

What is isomorphic code?

Isomorphic code can be deployed to multiple environments.

While the code remains the same, its behavior varies depending on the environment.

```
// my isomorphic JS code
function getOs() {
  return process.platform;
}
```

Chrome Browser

```
console.log(getOs())
// -> “Browser”
```

Ubuntu server

```
console.log(getOs())
// -> “Unix”
```

Initial purpose of Taon

– Explore the maximum isomorphic way of creating backend / frontend TypeScript apps and libraries

– Build a community around high-level coding

– Help me in my everyday Angular work

Goals of Taon today

– A natural backend for Angular

– Making web development fun again
(like in the PHP 2005 era, but with AI)

– High-quality code with CI/CD without huge DevOps costs

– Java-like security with JavaScript suitable even for critical systems

– One type of TypeScript developer instead of separate frontend and backend roles

What is Taon used for today

Creating, building, testing, and releasing:

– npm CLI tools

– Isomorphic npm libraries (backend + frontend)

– Angular PWA and SSR applications

– NodeJS backends with TypeORM

– Electron desktop apps

– Visual Studio Code extensions

– Capacitor mobile apps

In the most unified and elegant way possible.

Who is Taon built for?

Taon is built for **scale** —

supporting everything from small hobby projects

to large organizations with hundreds of developers.

Taon is **flexible**

and perfect for incremental development.

Just like Angular v21.

Taon uses Angular v21

Angular is now the best TypeScript framework. Period.

- Signals – faster and more stable than React Solid SSR, hydration, and tree-shaking – no more worries about large bundles
- Object-oriented approach – feels natural if you know design patterns
- Big projects feel lighter in Angular
- Backed by one of the biggest corporations on the planet

What is taon ?

- **Framework** written in TypeScript with building blocks for isomorphic architecture:
  - **Backend**: ExpressJS controllers, TypeORM entities,repositories, and subscribers
  - **Frontend**: Angular with API services
- **CLI** (powerful commands for building, testing, and releasing apps, libraries, and plugins)
- **Cloud** (easy to install, with CI/CD done in a way that normally requires a lot of DevOps)

EVERYTHING IS WRITTEN IN TYPESCRIPT.
EXTENSIBLE AND EASY TO IMPROVE.
DESIGNED IN A DEVELOPER-FRIENDLY WAY.
FIRST-CLASS VSCODE (VSCODIUM) SUPPORT

What can Taon be compared to?

Frameworks: NestJS, Blazor, NextJS

CLIs: Firebase CLI, Google Cloud CLI, Angular CLI

Cloud: Firebase, Google Cloud, Vercel

Taon adopts only
the most developer-friendly approaches.

Should I trust a backend written in JavaScript?

- JavaScript is not slow anymore.
- Single-core performance favors predictable, event-driven systems.
- Taon Cloud uses Traefik (written in Go) and multiple NodeJS backend instances for scalable performance.

10 Reasons Why Taon is THE FUTURE of Angular/NodeJS development

LET’S BEGIN!

10 Reasons Why Taon is THE FUTURE of Angular/NodeJS development

1. Tree-shakable Namespaces in TS/JS production solved with :

   a. clever TypeScript API
   b. taon npm package structure
   c. ts imports/exports manipulations

TAON IS A **FIX** FOR entire ESM JAVASCRIPT ECOSYSTEM

HOW TAON PROCESSES NAMESPACES

```
export namespace Utils {
 export const PI = 3.14;
 export function uniqueArray() { /* */ }
  export namespace css {
   export function calculateGoldenRatio(px:number) { /* */ }
 }
}
```

BECOMES THIS IN PRODUCTION

```
export function Utils__NS__PI = 3.14;
export function Utils__NS__uniqueArray() { /* */ }
export function Utils__NS__css__NS__calculateGoldenRatio(px:number) {
/* */ }

// DEEP TREE-SHAKABLE NAMESPACES
// each Taon artifact does this automatically
// minification will obviously make
// function names much smaller
```

TAON NPM PACKAGE STRUCTURE

```
/browser      // development frontend (ng serve)

/browser-prod // tree-shakable frontend build

/lib          // development backend code

/lib-prod     // tree-shakable backend build

/websql       // backend-in-browser (development)

/websql-prod  // tree-shakable backend-in-browser build


/ < additional .json(s) with namespaces metadata >
```

TREE-SHAKABLE LODASH AGAIN!

```
import { _ } from '@taon/core/src';
// tree-shakable lodash


_.first(['a','b']) // -> works again!!!
// MUCH BETTER DX THAN first()
```

10 Reasons Why Taon is THE FUTURE of Angular/NodeJS development

2. Zero-string, automatically generated configuration for development and production:

- no more defining ports for your backend or frontend
- no more defining Docker configurations
- no more string-based host or API endpoints
- no hard-to-read .env files — Taon uses .ts configs

TAON HAS IT OWN PORTS DATABASE

```
$ taon app // Taon automatically assigns
           // a development port
$ taon release // Taon Cloud uses the ports
               // database for production and
               // compose files
```

IT DOESN'T MATTER WHETHER IT'S DEVELOPMENT OR PRODUCTION — YOU NEVER ASSIGN PORTS MANUALLY AGAIN

ALL HOSTS COMBINED IN app.hosts.ts

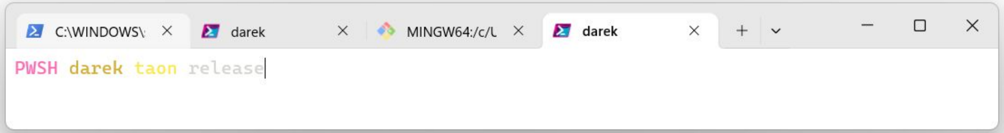
`src/app.hosts.ts` - contains all required hosts for your applications entry points:

```
src/app.ts              // webapp
src/app.vscode.ts       // vscode plugin
src/app.electron.ts     // electron
src/app.mobile.ts       // mobile
```

ONE MAIN env.ts + MANY .ts environments

```
import type { EnvOptions } from 'taon/src';

const env: Partial<EnvOptions> = {
 website: {
   domain: 'taon.dev', // if you create project www-taon-dev
   useDomain: true,    // -> proper domain is generated
   title: 'Taon.Dev',
 },
};
export default env;
```

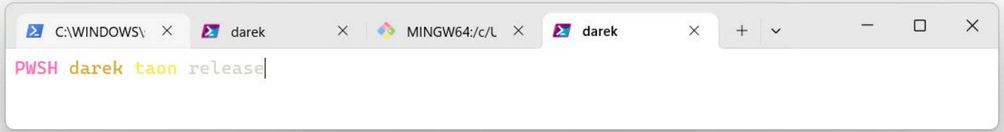


```
PWSH darek taon release
```

INHERITANCE OF ENVIRONMENTS CONFIGS

```
// angular-node-app.env.dev.ts
import type { EnvOptions } from 'taon/src';
import MainEnv from '../env';
const devDomain = `${MainEnv.website.domain}.dev`;

const env: Partial<EnvOptions> = {
  ...MainEnv,
  build: { ...MainEnv.build, prod: true },
  website: { ...MainEnv.website, domain: devDomain  }
};
export default env;
```

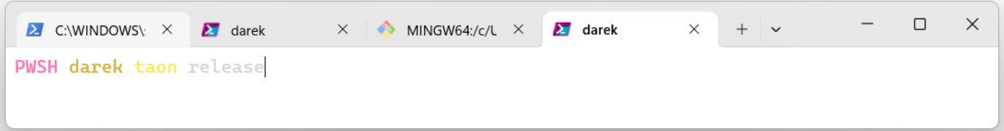


```
PWSH darek taon release
```

INHERITANCE OF ENVIRONMENTS CONFIGS

```
// electron-app.env.dev.ts  EACH TAON ARTIFACT GETS SEPARATE ENV CONFIGS
import type { EnvOptions } from 'taon/src';
import MainEnv from '../env';
const devDomain = `${MainEnv.website.domain}.local`;

const env: Partial<EnvOptions> = {
  ...MainEnv,
  build: { ...MainEnv.build, prod: true, angular: { ssr: false } },
  website: { ...MainEnv.website, domain: devDomain  }
};
export default env;
```



```
PWSH darek taon release
```

10 Reasons Why Taon is THE FUTURE of Angular/NodeJS development

3. (DX) Developer experience FIRST framework/cli/cloud

- Cross-project TypeScript debugging
- Automatically generated VS Code debugger configs (launch.json) for all Taon artifacts
- Angular v21 ng serve with watch-build of third-party node_modules packages
- Automatic recreation of core project structure from templates on every build

TAON VS ANGULAR - PROJECTS PHILOSOPHY

```
ANGULAR PROJECT
{[LIB1] [LIB2] [LIB3] }// many libs or apps
{[APP1] [APP2] }        // inside 1 project

TAON PROJECT1 // exactly one library
{[LIB] [APP]} // and one app per project
TAON PROJECT2 // TAON PROJECTS ARE
{[LIB] [APP]} // RECREATED FROM CORE
              // TEMPLATES EVERY TIME
```

TAON CONTAINERS

```
TAON CONTAINER PROJECT // Can be used as
                       // npm organization
TAON PROJECT1
{[LIB] [APP]}  // Group actions:
TAON PROJECT2  // init, build, release etc.
{[LIB] [APP]}  // can be executed on
     …         // container level
TAON PROJECTn
```

DEVELOPMENT BUILD OF MULTIPLE PROJECTS

```
TAON PROJECT my-person-lib
{[LIB: person.ts class Person {}] [APP]}
TAON PROJECT my-user-lib
{[LIB: class User extends Person {}] [APP]}

// both project started in dev watch build
// with $ taon build:watch:lib
// => proper npm packages inside core
// container node_modules after rebuild
```

HOW DEBUGGING IS DONE IN **OTHER** FRAMEWORKS

```
import { Person } from 'my-person-lib';
// by clicking Person inside import
// we are redirected to d.ts file
```

🥴 🤮

```
export class User extends Person {}
```

HOW DEBUGGING IS DONE IN **OTHER** FRAMEWORKS

```
declare class Person { /*  */ }

// person.d.ts files 🥴  🤮
// not useful at all in development
```

HOW DEVELOPMENT BUILDS WORK IN TAON

```
import {Person} from 'my-person-lib/src';
// clicking Person opens person.ts file
// if my-person-lib is running
// in development taon watch mode
export class User extends Person {}
```

HOW DEVELOPMENT BUILDS WORK IN TAON

```
export class Person {
 // we can not only see this code
 // but we can debug it ⚡⚡⚡
 // my-person-lib/src is a link
 // to the real src/lib directory
}
```

HOW IT IS POSSIBLE - TAON RULES OVER IMPORTS

```
import { Person } from 'my-person-lib/src';
// without Taon watch mode:
// 'my-person-lib/src' => points to d.ts files.


// With Taon, imports are rewritten per target:
import { Person } from 'my-person-lib/lib';
import { Person } from 'my-person-lib/browser';
```

10 Reasons Why Taon is THE FUTURE of Angular/NodeJS development

4. Taon’s “impossible” TypeScript refactoring capabilities

- Renaming a database entity property automatically updates Angular template bindings (if Angular Language Service is enabled in VS Code)

- Discovering project source code through Taon imports enables safe cross-project renaming and refactoring

IN OUR EDITOR — TWO TABS OPEN

```
// VSCode tab 1 (person.ts) from my-person-lib project
export class Person {}

// VSCode tab 2 (user.ts) from my-user-lib project
export class User extends Person {}
// cross-project TypeScript renaming / refactor ⚡
```

10 Reasons Why Taon is <u>THE FUTURE</u> of Angular/NodeJS development

5. Ultime Isomorphic Design

- Ultimate DRY solution
- Taon controllers / middleware as glue between backend and frontend
- Cutting isomorphic .ts files for:
  > frontend code
  > backend code
  > npm library only code

Reuse backend controllers as angular services

```
// BACKEND CODE
export class UserController {
  @GET()
  getAll(): Taon.Response<User[]> {
    //#region @backendFunc
    return () => this.db.find(); // get all users
    //#endregion
  }
}
```

Reuse backend controllers as angular services

```
// FRONTEND CODE
export class UserControlller {
  @GET()
  getAll(): Taon.Response<User[]> {
    /* */
    /* */
    /* */
  } // TAON INTERNALLY BUILDS PROPER GET PATH FOR GET REQUEST
}
```

Reuse backend controllers as angular services

```
@Injectable()
export class UserApiService{
  userController = this.injectController(UserController);
  async allUsers():User[] {
                                 // .request() => taon internal generaed method
    const requestData = await this.userController.getAll().request();
    // by calling .request()
    // we do GET /UserController/getAll rest api call
    return requestData.body.json;
  }
}   // UserController in theory can be used as api service, but in practise
    // it is better to use special taon api services with controller injected
```

Isomorphic middlewares

```
// MIX OF EXPRESSJS MIDDLEWARES AND ANGULAR INTERCEPTORS
export class UserSessionMiddleware extends BaseTaonMiddleware {
  // intercepting backend or frontend request on the whole context level
  interceptServer(req, res, next) {  /* */  }
  interceptClient(req, next) {  /* */  }

  // intercepting specific controller or controller method
  interceptServerMethod(req, res, next) {  /* */  }
  interceptClientMethod(req, next) {  /* */  }
}
```

Isomorphic middlewares

```
// Building truly unified interceptors
export class UserSessionMiddleware extends BaseTaonMiddleware {
  currentSession(req):Session {
    //#region @backend
     return req.user.session;
    //#endregion
    //#region @browser
     return req.session;
    //#endregion
  }
}
```

Npm library code - clean for npm publish

```
// Publishing clean reusable npm packages
// (available outside taon)
export const ProjectConfig = {
  //#region @onlyForNpm
  secrete: 'to-hide',
  // useful in taon project app but not useful on npm
  //#endregion
}
```

10 Reasons Why Taon is THE FUTURE of Angular/NodeJS development

6. Backend with TypeORM in most most defightfull/unified way possible

- Repositories as classes again
- Websql for sql.js and coding database inside browser
- Name based framework for easy debugging and clear structure
- Cross platform data layer
  - REST based controller switches to IPC in electron
  - REST based controller switches to JS mock in websql mode
  - REST based controller switches to REST in NodeJS mode

Backend repositories as classes

```
// Repositories as classes (current typeorm uses objects)
export class UserRepository extends BaseTaonRepository<User> {

  async getLast5User(): Promise<User[]> {
    //#region @backendFunc
    return this.db.find({ limit: 5 });
    //#endregion
  }
}
```

Backend repositories as classes

```
// Repositories as classes (current typeorm uses objects)
export class UserRepository extends BaseTaonRepository<User> {

  async getLast5User(): Promise<User[]> {
    /* */
    /* */ REPOSITORIES IN GENERAL SHOULD ONLY BE USED ON BACKEND
    /* */ BUT.. THERE IS EXCEPTION..
  }
}
```

WEBSQL mode - code backend on frontend with **sql.js**

```
// Repositories as classes (current typeorm uses objects)
export class UserRepository extends BaseTaonRepository<User> {

  async getLast5User(): Promise<User[]> {
    //#region @websqlFunc         // => @websql instead @backend
    return this.db.find({ limit: 5 });
    //#endregion
  }
}
```

In websql mode backend stays in browser

```
// Repositories as classes (not available in typeorm now)
export class UserRepository extends BaseTaonRepository<User> {

  async getLast5User(): Promise<User[]> {
    //#region @websqlFunc
    return this.db.find({ limit: 5 }); // executed in browser !!!
    //#endregion
  }
}
```

10 Reasons Why Taon is <u>THE FUTURE</u> of Angular/NodeJS development

7.Taon contexts as ultimate flexible way of backends / databases / servers architecture

JavaScript already is winning

- **web** ✅
- **electron**✅
- **web mobile, pwa** ✅
- **backend** ⚠️ ???

TAON will ensure win of JS on backend.

Taon contexts

```
const UserContext = Taon.createContext(()=> (
  abstract?: true, // abstract === true    => for libraries/reusable contexts
                   // abstract === false   => context is active
  entities: { User, Session },
  controllers: { UserController },
  repositories: { UserRepository },
  middleware: { SessionMiddleware },
));

await UserContext.init() // active context can be initialized

// initialized => 1 SERVER, 1 DB per context
```

Joining Multiple Contexts

```
const UsersContext = Taon.createContext(() => (
  abstract: true, entities: { User, Session }, controllers: { UserController },
));

const BooksContext = Taon.createContext(() => (
  abstract: true, entities: { Book }, controllers: { BookController },
));

const AppContext = Taon.createContext(() => (
  abstract: false, contexts: { UsersContext, BooksContext },
));
await UserContext.init() // 1 DB (User,Book) 1 Server (UserController,BookController)
```

Active contexts comes with generated config for hosts

```
import { ...HOST_CONFIG } from './app.hosts'; // generated file

const AppContext = Taon.createContext(() => (
  ...HOST_CONFIG['AppContext'], // we are appending generated configs
  abstract: false, contexts: { UsersContext, BooksContext },
));
```

Splitting backend / frontend into multiple contexts

```
// each active context can be develop separately
// with separated frontend
const WebsiteContext = Taon.createContext();
const DashboardContext = Taon.createContext();
const SettingsContext = Taon.createContext();

// each abstract context can be shared and put in separate project
const SessionContext = Taon.createContext();
const AuthorizationContext = Taon.createContext();
```

Context cloning

```
// each active context can be develop separately
// with separated frontend
const UsersContext = Taon.createContext();
const UsersBackupContext = UsersContext.clone({
 // override db type/location
 // override host
})
```

10 Reasons Why Taon is THE FUTURE of Angular/NodeJS development

8. Isomorphic backend/frontend npm package:

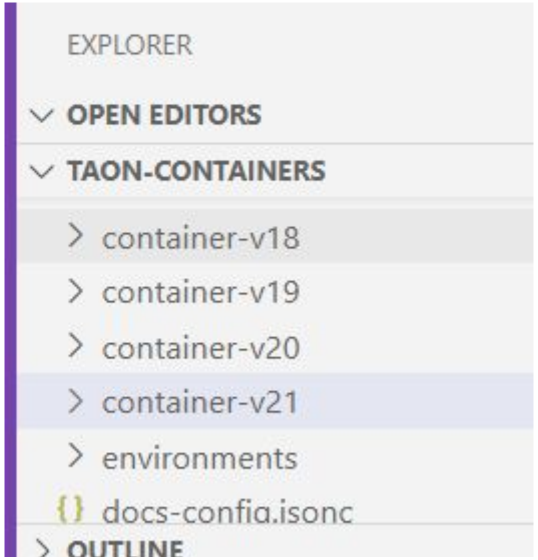
- npm libraries with backend/frontend code that actually make sense (imagine sharing whole Authentication, CMS, E-commerce and other parts BE/FE that can be easily separated)
- shared assets in npm packages
- sharde .scss in npm packages

SHARING CODE IS AWESOME AGAIN! ⭐

10 Reasons Why Taon is <u>THE FUTURE</u> of Angular/NodeJS development

9. Ultimate security for ecosystem

- no more direct npm install for each project (just core containers rebuilds)
- taon long term support for major Angular versions (starts from v21)
- Taon cloud auto updates based on core container with community selected best / stable packages
- similar projects share node_modules with zero installation time (it is just link)



10 Reasons Why Taon is THE FUTURE of Angular/NodeJS development

10. Taon cloud for perfect CI/CD:

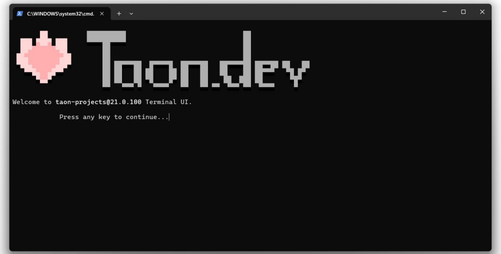
- Build local and release or release from server (your choice)

- Local taon cloud is the same as destination server taon cloud

- Terminal-first interface for development

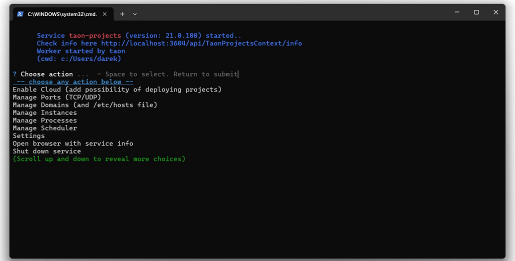
- Single command release for every artifact: **$ taon release**

- CI/CD not only for big companies



```
Welcome to taon-projects@21.0.100 Terminal UI.

          Press any key to continue...
```

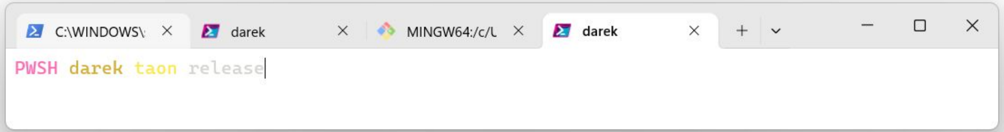


```
     Service taon-projects (version: 21.0.100) started..
     Check info here http://localhost:3604/api/TaonProjectsContext/info
     Worker started by taon
     (cwd: c:/Users/darek)

? Choose action ...  - Space to select. Return to submit
 -- choose any action below --
Enable Cloud (add possibility of deploying projects)
Manage Ports (TCP/UDP)
Manage Domains (and /etc/hosts file)
Manage Instances
Manage Processes
Manage Scheduler
Settings
Open browser with service info
Shut down service
(Scroll up and down to reveal more choices)
```

Taon cloud will do complex deployments for you

1. TRANSACTIONS WAITING MODE
2. SETTING DB / APP TO BE READONLY
3. DB REPLICATION OF CURRENT DATABASE
4. SET DB TO WRITE MODE
5. RESTART APP
6. APPLY NEW MIGRATIONS
7. SLOWLY INTRODUCE NEW APP TO USER WITH TREAFIX REVERSE PROXY
8. IF THERE IS ANYTHING WRONG AFTER E2E TEST => PROCESS WILL REVERSE ITSELF



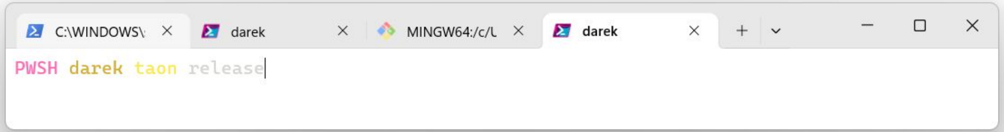
```
PWSH darek taon release
```

Taon will cut costs and responsibilities

1. Self hosted secure taon instance will saves hundred thousands of $$$ in comparison to azure or amazon.

2. Self updated secure and backed by community taon cloud instances

3. One programmer with AI will be able to do huge amount of work like never before.



```
PWSH darek taon release
```

10 Reasons Why Taon IS THE FUTURE OF Angular/NodeJS development

ADDITIONAL REASON:

AI would choose TAON.

TypeScript is not only human friendly
 => it is also language that AI understands the most

Final words..

Unless you are building software for new SR-71

JavaScript is for you…

Unless you are building software for new SR-71

TAON is for you…

Thanks for watching!

Taon.dev