

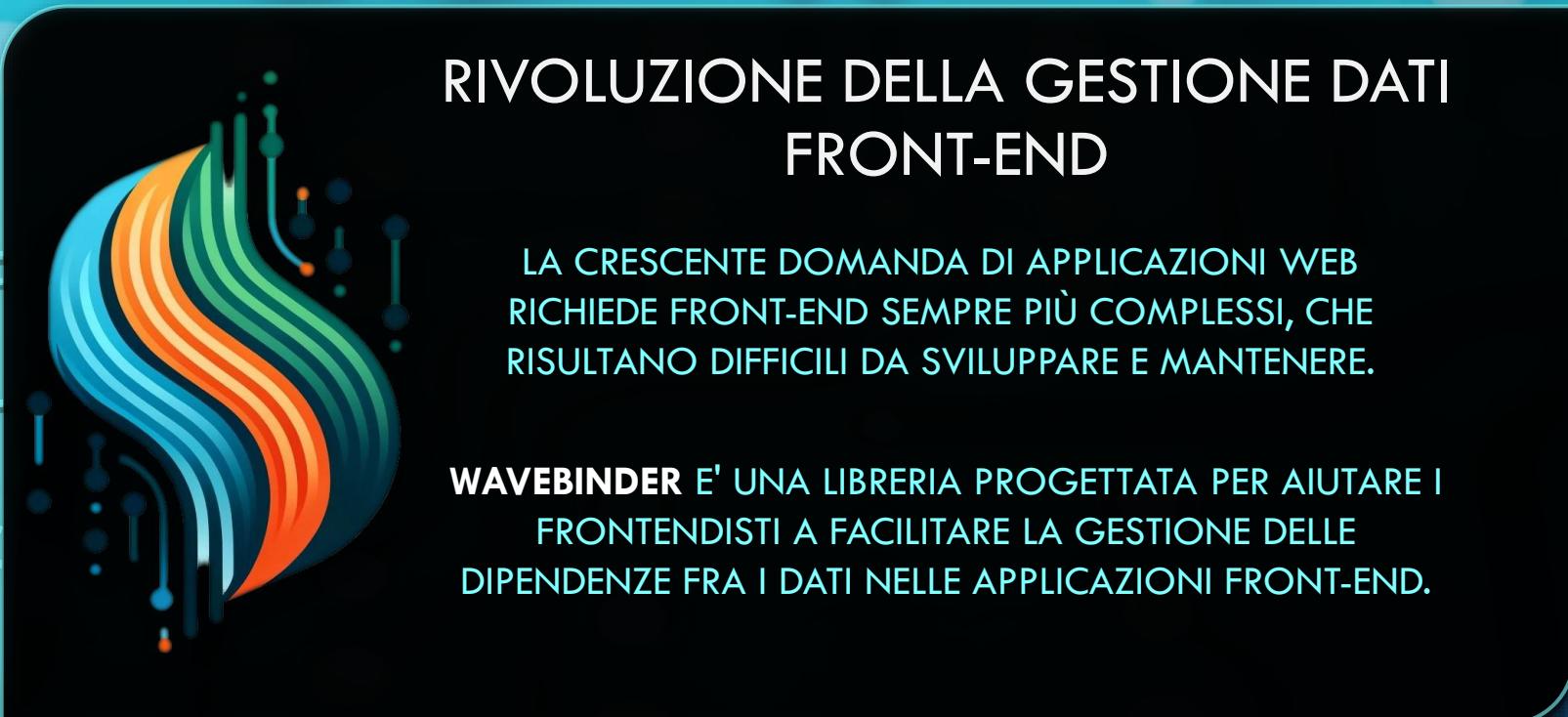


WAVEBINDER

RIVOLUZIONE DELLA GESTIONE DATI FRONT-END

LA CRESCENTE DOMANDA DI APPLICAZIONI WEB RICHIENDE FRONT-END SEMPRE PIÙ COMPLESSI, CHE RISULTANO DIFFICILI DA SVILUPPARE E MANTENERE.

WAVEBINDER È UNA LIBRERIA PROGETTATA PER AIUTARE I FRONTENDISTI A FACILITARE LA GESTIONE DELLE DIPENDENZE FRA I DATI NELLE APPLICAZIONI FRONT-END.



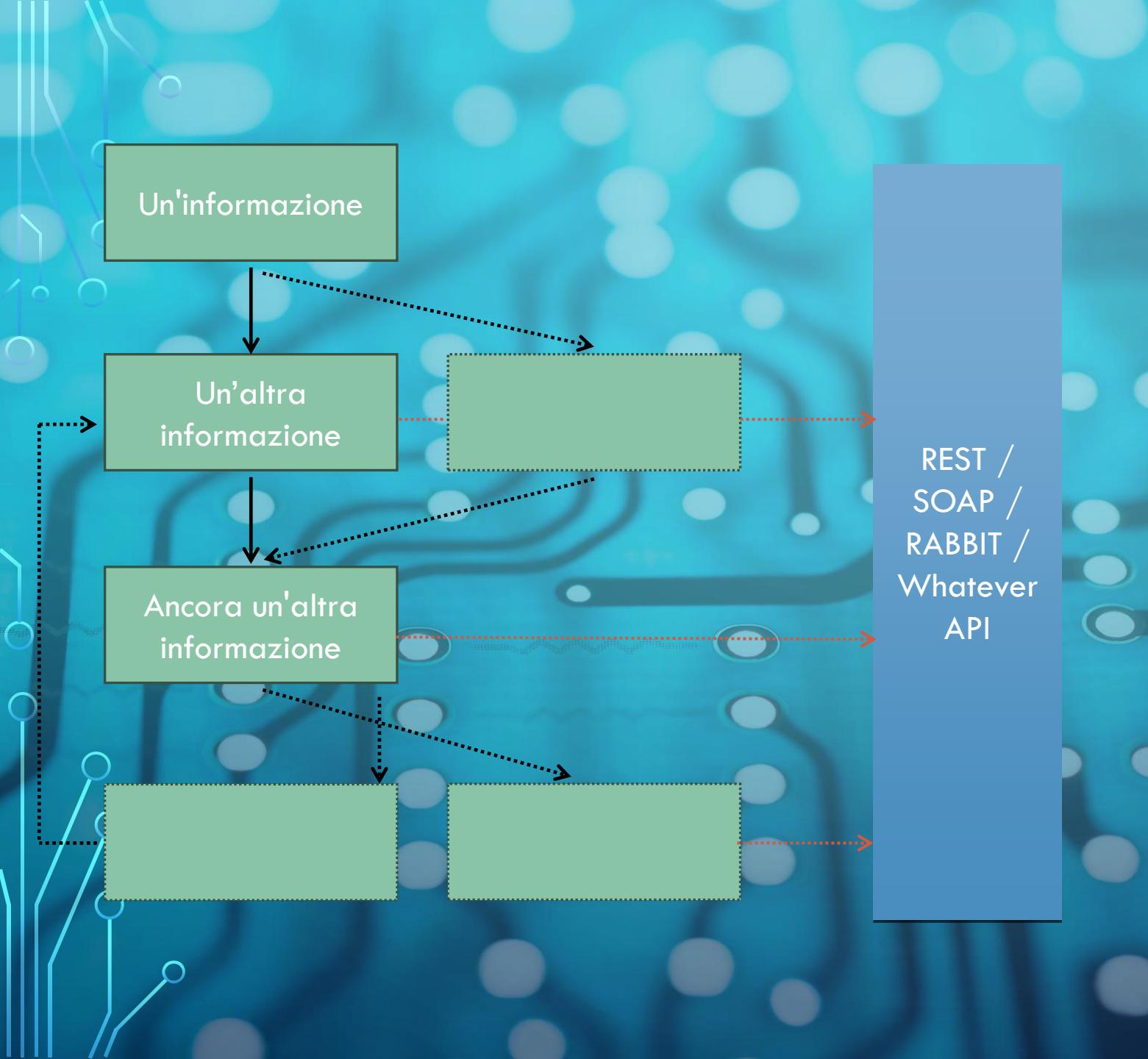
IL PROBLEMA DEI FLUSSI DATI

PERCHÉ LA GESTIONE DEI DATI È COSÌ COMPLESSA?

Nelle applicazioni moderne, ogni campo e componente dipende da altri dati.

I dati non si aggiornano in modo lineare

Le complessità aumentano quando si utilizzano API lente e asincrone



IL PROBLEMA DEI FLUSSI DATI

PERCHÉ LA GESTIONE DEI
DATI È COSÌ COMPLESSA?

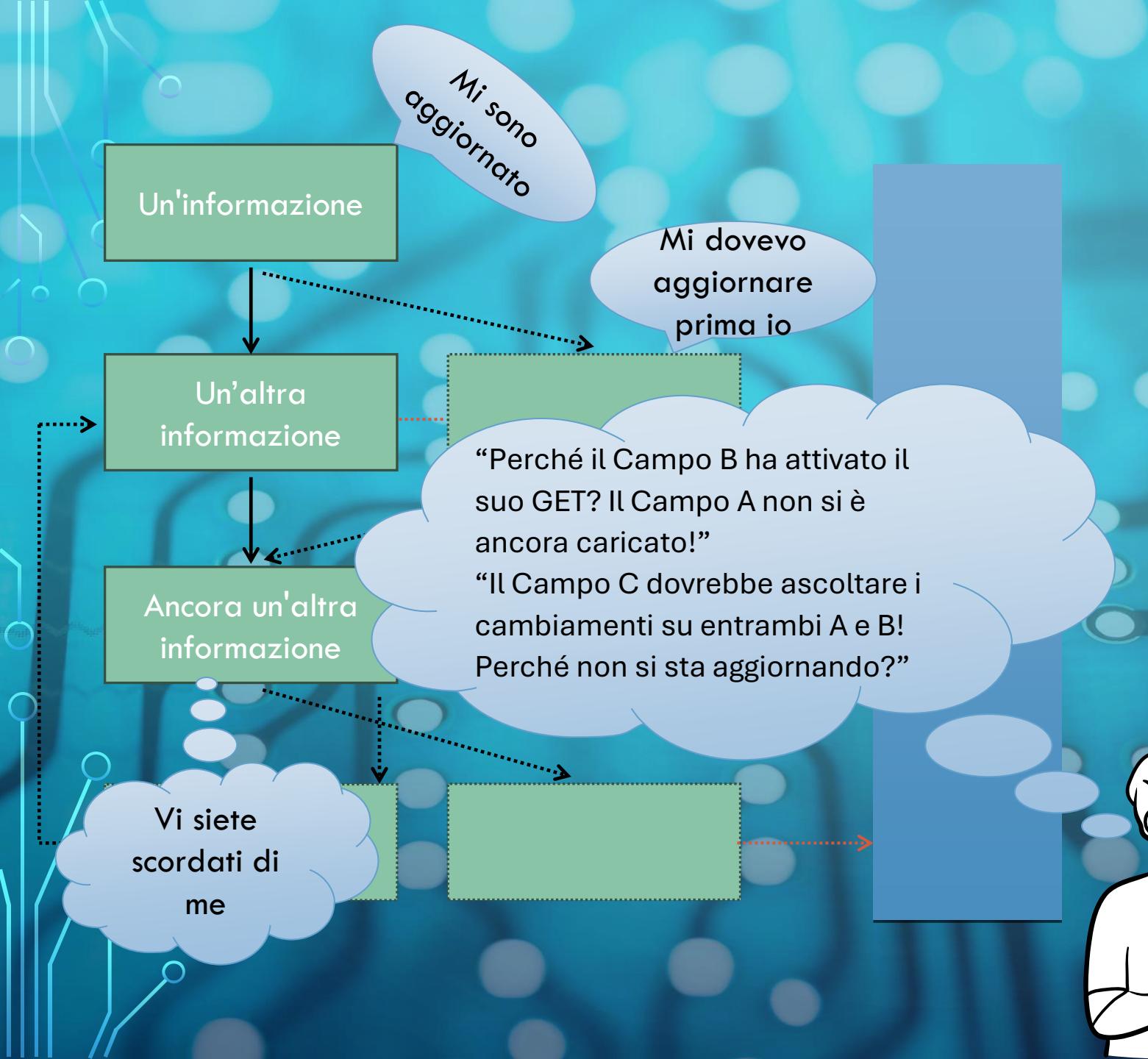
Le interfacce sono sempre più
complesse e dinamiche



IL PROBLEMA DEI FLUSSI DATI

QUALI SONO LE CONSEGUENZE?

1. Tempi lunghi e ritardi nello sviluppo delle interfacce
2. Errori e comportamenti indesiderati
3. Esperienza utente non ottimale
4. Difficoltà nel mantenimento e nell'aggiornamento delle logiche



IL MERCATO DEI FRONTENDISTI E LE LORO SFIDE

COSA CERCANO GLI SVILUPPATORI FRONT-END?

Gli sviluppatori front-end cercano strumenti che semplifichino la loro vita e ottimizzino il flusso di dati tra componenti.

La gestione manuale delle dipendenze dati diventa insostenibile in applicazioni complesse.

WAVEBINDER è stato sviluppato per rispondere a queste esigenze, offrendo automazione, flessibilità e controllo.



CI PENSO
IO!



WAVEBINDER™

WAVEBINDER

WAVEBINDER offre una compatibilità e una facilità d'uso che lo rendono ideale per l'integrazione nei tuoi progetti front-end.

Ecco alcune delle sue caratteristiche principali:

Disponibile come pacchetto NPM

Basato su RXJS

Funziona perfettamente con i principali framework:

- Vue.js
- Angular
- React



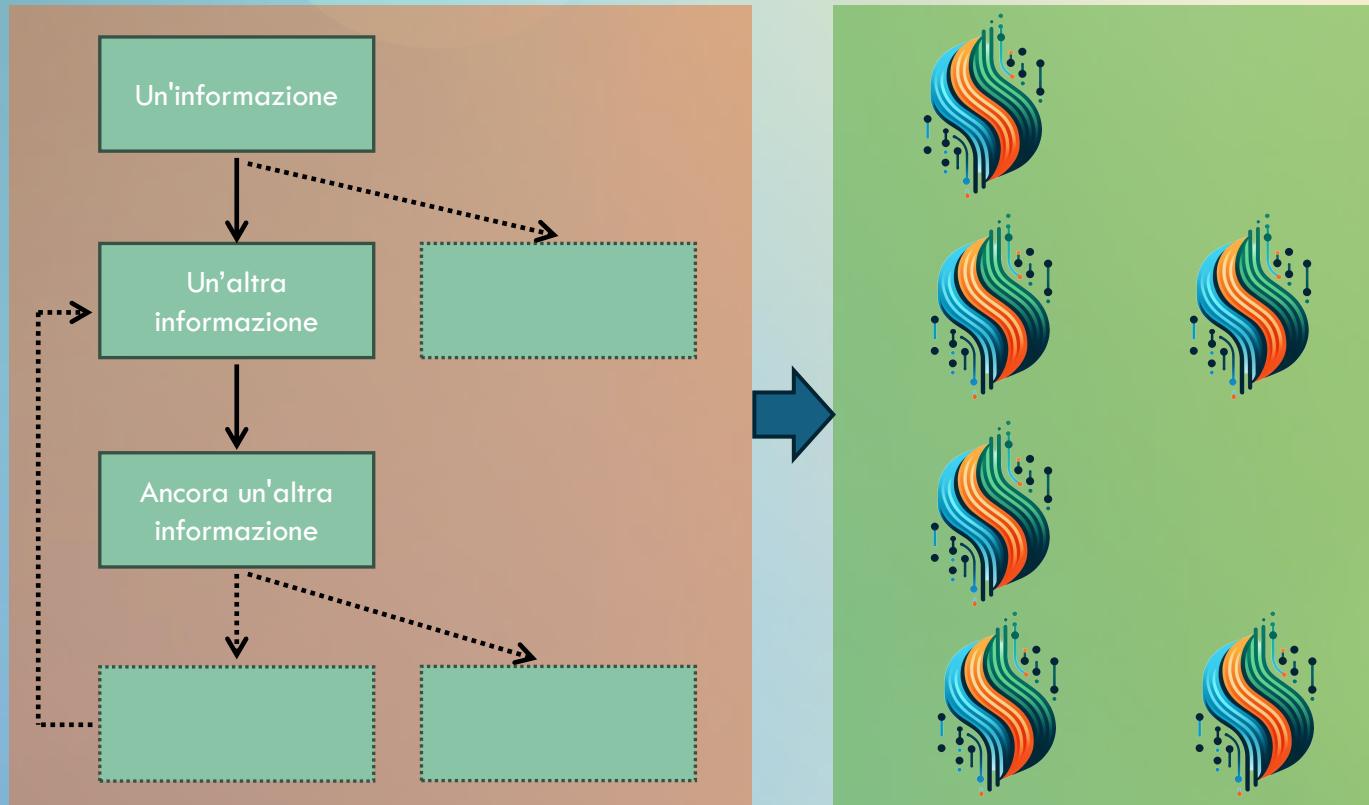
COME FUNZIONA WAVEBINDER

IL POTERE DELLA MODELLAZIONE A NODI

I dati sono modellati come se fossero nodi di un grafo.

Definisci i nodi in un oggetto JSON come questo:

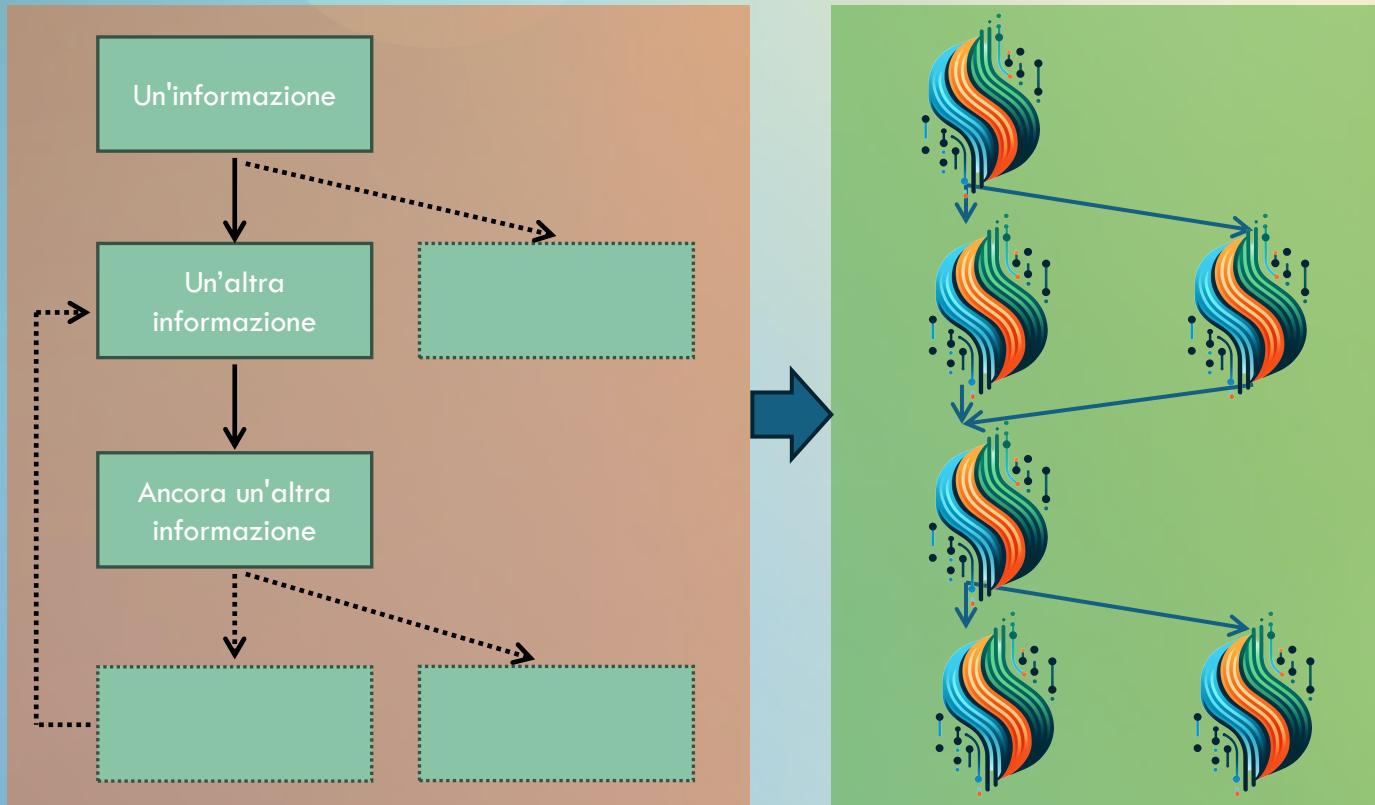
```
const config : {[path: string, la: {type: str... =  
[  
  {  
    "name": "region",  
    "type": "MULTI",  
    "path": "/region",  
    "la": {  
      "type": "GET",  
      "addr": "/region/list",  
      "serviceName": "RETRIEVE_DATA"  
    },  
    "dep": []  
  },  
  {  
    "name": "province",  
    "type": "MULTI",  
    "path": "/province",  
    "la": {  
      "type": "GET",  
      "addr": "/{region}/province/list",  
      "serviceName": "RETRIEVE_DATA"  
    },  
    "dep": []  
  }  
];
```



COME FUNZIONA WAVEBINDER

IL POTERE DELLA MODELLAZIONE A NODI

Anche le dipendenze tra i nodi sono specificate lì.



```
{  
  "name": "city",  
  "type": "MULTI",  
  "path": "/city",  
  "la": {  
    "type": "GET",  
    "addr": "/{region}/{province}/city/list",  
    "serviceName": "RETRIEVE_DATA"  
  },  
  "dep": [  
    {  
      "nodeName": "region",  
      "isOptional": false,  
      "onUpdate": true,  
      "type": "PATH_VARIABLE"  
    },  
    {  
      "nodeName": "province",  
      "isOptional": false,  
      "onUpdate": true,  
      "type": "PATH_VARIABLE"  
    }  
  ]  
},
```

COME FUNZIONA WAVEBINDER

IL POTERE DELLA MODELLAZIONE A NODI

Puoi specificare molte proprietà per ogni nodo:

Il tipo di dato che contiene (es. un singolo valore, una lista, un oggetto, ...)

Il percorso in cui memorizzarlo su un modello

L'azione necessaria per caricare il suo valore (es. una chiamata API, una selezione da parte dell'utente, ...)

Le sue dipendenze da altri nodi

```
const config :[{path: string, la: {type: str...  =  
[  
  {  
    "name": "region",  
    "type": "MULTI",  
    "path": "/region",  
    "la": {  
      "type": "GET",  
      "addr": "/region/list",  
      "serviceName": "RETRIEVE_DATA"  
    },  
    "dep": []  
  },  
  {  
    "name": "province",  
    "type": "MULTI",  
    "path": "/province",  
    "la": {  
      "type": "GET",  
      "addr": "/{region}/province/list",  
      "serviceName": "RETRIEVE_DATA"  
    }  
  }  
]
```

COME FUNZIONA WAVEBINDER

IL POTERE DELLA MODELLAZIONE A NODI

Puoi specificare molte proprietà per ogni nodo:

Il tipo di dato che contiene (es. un singolo valore, una lista, un oggetto, ...)

Il percorso in cui memorizzarlo su un modello

L'azione necessaria per caricare il suo valore (es. una chiamata API, una selezione da parte dell'utente, ...)

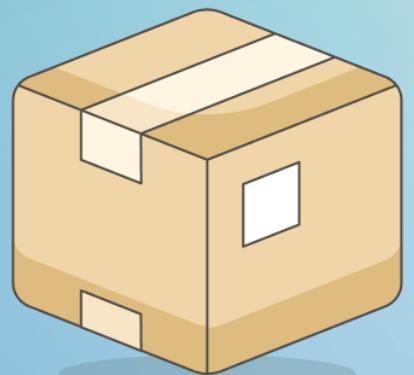
Le sue dipendenze da altri nodi

Abbiamo sviluppato
un tool CLI che lo
renderà
semplicissimo.

FANTASTICO!



```
const config :[{path: string, la: {type: str...  =  
    [ {  
        "path": "/region/list",  
        "la": {  
            "type": "GET",  
            "addr": "/region/list",  
            "serviceName": "RETRIEVE_DATA"  
        },  
        "dep": []  
    },  
    {  
        "path": "/region/{region}/province/  
        "serviceName": "RETRIEVE_DA
```



COME FUNZIONA WAVEBINDER

INSTALLAZIONE E CONFIGURAZIONE FACILI

WAVEBINDER si integra facilmente nei progetti esistenti.

Basta eseguire «`npm install wave-binder`» per iniziare.

Con pochi passaggi, puoi configurare i nodi e le loro dipendenze usando JSON.

Anche per gli sviluppatori alle prime armi con l'uso di nodi, il tool CLI semplifica la creazione e la gestione delle dipendenze dati.

COME FUNZIONA WAVEBINDER

FUNZIONALITÀ

Una volta installato WAVEBINDER e modellato il tuo problema nella struttura di configurazione, sei quasi pronto per partire.

- Fornisci la configurazione a una nuova istanza di WAVEBINDER.
- Indica a WAVEBINDER di costruire la rete di dipendenze
- Fatto

Come?! Fatto?!
Fatto cosa?



```
const wb :WaveBinder = new m.WaveBinder(config);  
  
wb.tangleNodes();
```

COME FUNZIONA WAVEBINDER

FUNZIONALITÀ

WAVEBINDER espone:

- I metodi necessari per recuperare i nodi
- Vari modi per interagire con essi.

```
const regionNode :null = wb.getNodeByName( name: 'region')
regionNode.setSelection(1);
```

```
const listNode :null = wb.getNodeByName( name: 'myListNode')
listNode.next( value: 3);
```

```
const regionsListNode :null = wb.getNodeByNameAndType( name: 'regionsList' , type: 'LIST')
regionsListNode.children[1].setSelection(1);
```

```
public constructor(protoNodes: ProtoNode[], Show usages new*
                  extApis: Map<string, HttpServiceSetting>,
                  customFunctions: FunctionInstance[]) {
}

public tangleNodes():void { no usages new*

}

public getNodes(): WaveBinderNode[] { no usages new *

}

public getDataPool():void { no usages new *

}

public getNodeInfo(logDepth?: number): WaveBinderNodeView[] { no usages new *

}

public getNodeByName(name: string): WaveBinderNode { no usages new *

}

public getNodeByNameAndType(name: string, no usages new*
                           type: string): WaveBinderNode {

}

public getNodeByNameAmongNodes(name: string, no usages new*
                               nodes: WaveBinderNode[]): WaveBinderNode {

}

public getNodeByNameAndTypeAmongNodes(name: string, no usages new*
                                       type: string,
                                       nodes: WaveBinderNode[]): WaveBinderNode {

}
```



The screenshot shows the homepage of wavebinder™. At the top, there's a decorative graphic of colorful, flowing lines resembling waves or data streams. Below it, the brand name "wavebinder™" is prominently displayed in white. A subtitle reads: "A dependency manager to simplify data relationships, so you can focus on business logic." Underneath, a section titled "Why Choose wavebinder™?" is shown with three cards: "Data as Graph Nodes", "Powered by RXJS", and "Multi-framework Support". At the bottom of this section are two buttons: "Documentation" (highlighted with a red oval) and "Example #1".

This screenshot shows the API documentation for the "IterationObj" interface. On the left, a sidebar lists various components under the "wave-binder" namespace, with "IterationObj" currently selected. The main content area displays the "Interface IterationObj" definition, which includes the interface declaration:

```
interface IterationObj {
  fatherName: string;
  index: number;
}
```

It also shows the "Properties" section, which includes "fatherName" and "index". The "fatherName" property is described as "The name of the parent node or context." and its definition is "Defined in [wvb/node-defs:126](#)". The "index" property is described as "The index in the current iteration." and its definition is "Defined in [wvb/node-defs:128](#)". To the right, there are "Settings" for member visibility (Protected, Inherited, External) and a "THEME" dropdown set to "OS". A link "On This Page" is also present.

COME FUNZIONA WAVEBINDER

DOCUMENTAZIONE

Ogni aspetto della configurazione e gestione di WAVEBINDER è documentato su www.wavebinder.it.

Questo assicura agli sviluppatori una guida costante per configurazioni avanzate e casi particolari, rendendo l'adozione ancora più semplice.



PERCHÈ FUNZIONA?

I SEGRETI DI WAVEBINDER

Una libreria potente

Anni di esperienza

La visione di astrarre la complessità
una volta per tutte, permettendo a
chiunque di concentrarsi solo su ciò che
il cliente desidera davvero: business
logic



PERCHÈ FUNZIONA?

ORDINE E PRECISIONE:

DIPENDENZE E AZIONI CHE ORGANIZZANO IL TUO CODICE

Per ogni nodo, ti viene chiesto di specificare due cose molto importanti:

- **Dipendenze**
- **Azione di caricamento**

Le dipendenze sono semplicemente riferimenti ad altri nodi

Se in qualsiasi momento quei nodi di riferimento sono completamente e correttamente caricati, l'azione di caricamento viene attivata

Questo, naturalmente, può attivare le azioni di caricamento di altri nodi

```
"dep": [  
    {  
        "nodeName": "region",  
        "isOptional": false,  
        "onUpdate": true,  
        "type": "PATH_VARIABLE"  
    },  
    {  
        "nodeName": "province",  
        "isOptional": false,  
        "onUpdate": true,  
        "type": "PATH_VARIABLE"  
    }  
]
```

```
"la": {  
    "type": "GET",  
    "addr": "/region/list",  
    "serviceName": "RETRIEVE_DATA"  
},
```

PERCHÈ FUNZIONA?

CONTROLLO COSTANTE SU OGNI AZIONE DELLA TUA INTERFAZIA

Ogni nodo mantiene un registro di ciò che gli è successo

- Aggiornamenti da altri nodi impostati come sue dipendenze
- Azione di caricamento eseguita quando le dipendenze sono soddisfatte

```
"eventsLog": [  
    {  
        "what": "INSTANTIATED",  
        "when": "2024-06-26T14:01:41.864Z"  
    },  
    {  
        "what": "RECEIVED undefined FROM region",  
        "when": "2024-06-26T14:01:41.885Z"  
    },  
    {  
        "what": "RECEIVED undefined FROM province",  
        "when": "2024-06-26T14:01:41.886Z"  
    },  
    {  
        "what": "RECEIVED null FROM region",  
        "when": "2024-06-26T14:01:41.895Z"  
    },  
    {  
        "what": "RECEIVED Toscana FROM region",  
        "when": "2024-06-26T14:01:43.898Z"  
    },  
    {  
        "what": "RECEIVED null FROM province",  
        "when": "2024-06-26T14:01:43.900Z"  
    },  
    {  
        "what": "RECEIVED Prato FROM province",  
        "when": "2024-06-26T14:01:44.893Z"  
    },  
    {  
        "what": "GET REQ SENT: http://localhost:3000/retrieve/Toscana/Prato/city/list",  
        "when": "2024-06-26T14:01:44.894Z"  
    },  
    {  
        "what": "SELECTION INVALIDATED, PROPAGATED NULL",  
        "when": "2024-06-26T14:01:44.895Z"  
    },  
    {  
        "what": "CHOICES SET TO: Poggio a Caiano, Montemurlo, Carmignano",  
        "when": "2024-06-26T14:01:44.895Z"  
    }  
]
```

