WAVEBINDER

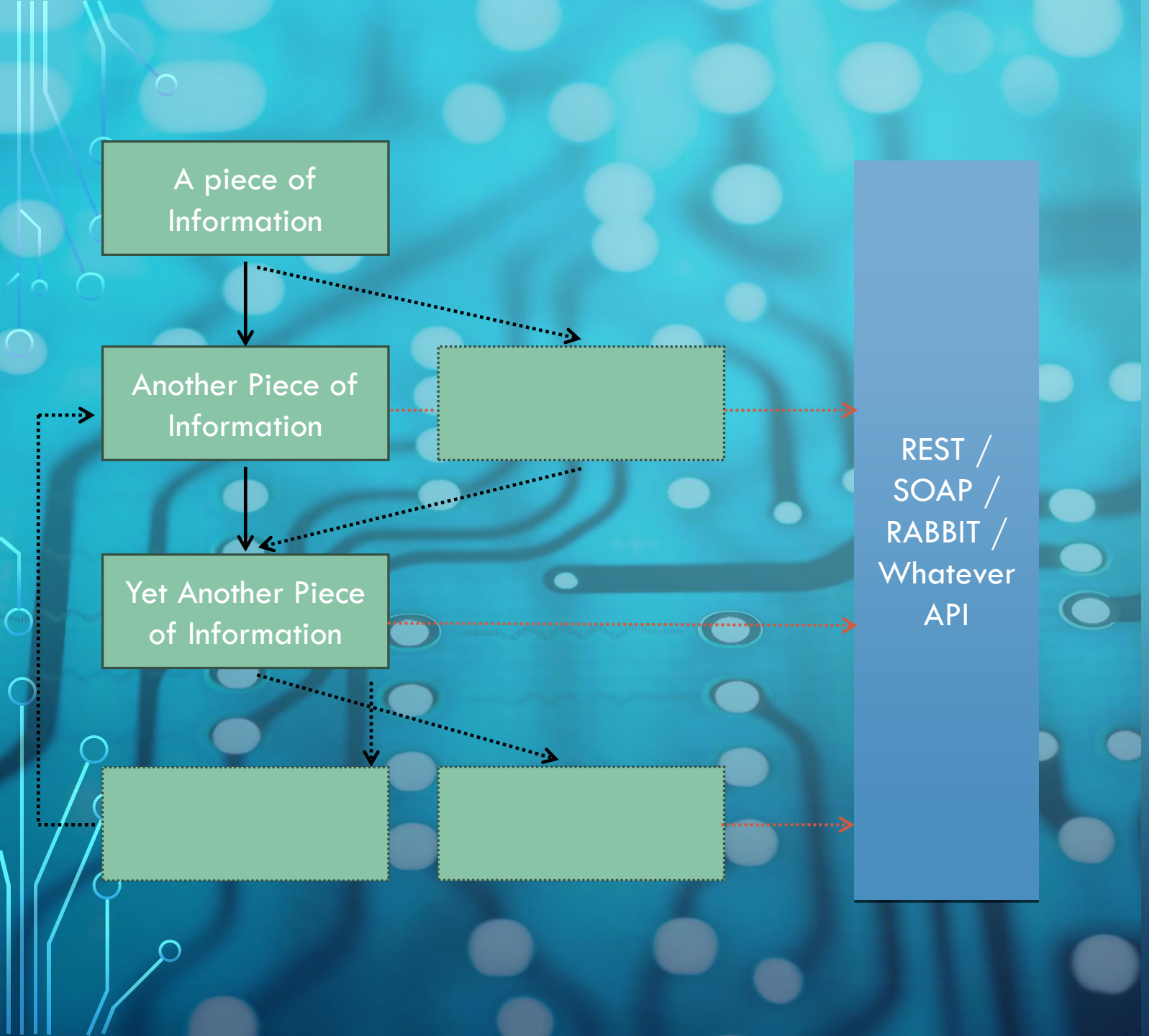
REVOLUTIONIZING FRONT-END DATA MANAGEMENT

WAVEBINDER

REVOLUTIONIZING FRONT-END DATA MANAGEMENT

THE GROWING DEMAND FOR WEB APPLICATIONS REQUIRES INCREASINGLY COMPLEX FRONT-ENDS THAT ARE CHALLENGING TO DEVELOP AND MAINTAIN.

WAVEBINDER IS A LIBRARY DESIGNED TO HELP FRONT-END DEVELOPERS EASILY MANAGE DATA DEPENDENCIES IN FRONT-END APPLICATIONS.



THE DATA FLOW CHALLENGE

WHY IS DATA MANAGEMENT SO CHALLENGING?

In modern applications, each field and component relies on other data.

Data does not update linearly.

Complexity grows when using slow, asynchronous APIs

THE DATA FLOW CHALLENGE

WHY IS DATA MANAGEMENT SO CHALLENGING?

Interfaces are increasingly complex and dynamic

THE DATA FLOW CHALLENGE

WHAT ARE THE CONSEQUENCES?

1. Long development times and interface delays
2. Errors and undesirable behaviors
3. Suboptimal user experience
4. Difficulty in maintaining and updating logic

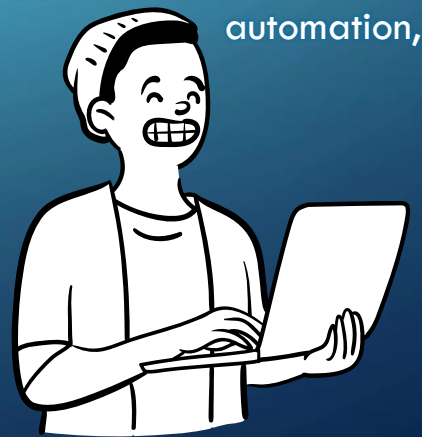
THE FRONT-END MARKET AND ITS CHALLENGES

WHAT DO FRONT-END DEVELOPERS NEED?

Front-end developers look for tools that simplify their work and optimize data flow between components.

Manual management of data dependencies becomes unsustainable in complex applications.

WAVEBINDER was developed to meet these needs, offering automation, flexibility, and control.



WAVEBINDER™

WAVEBINDER

WAVEBINDER offers compatibility and ease of use, making it ideal for integration into your front-end projects.

Here are some of its key features:

Available as an NPM package

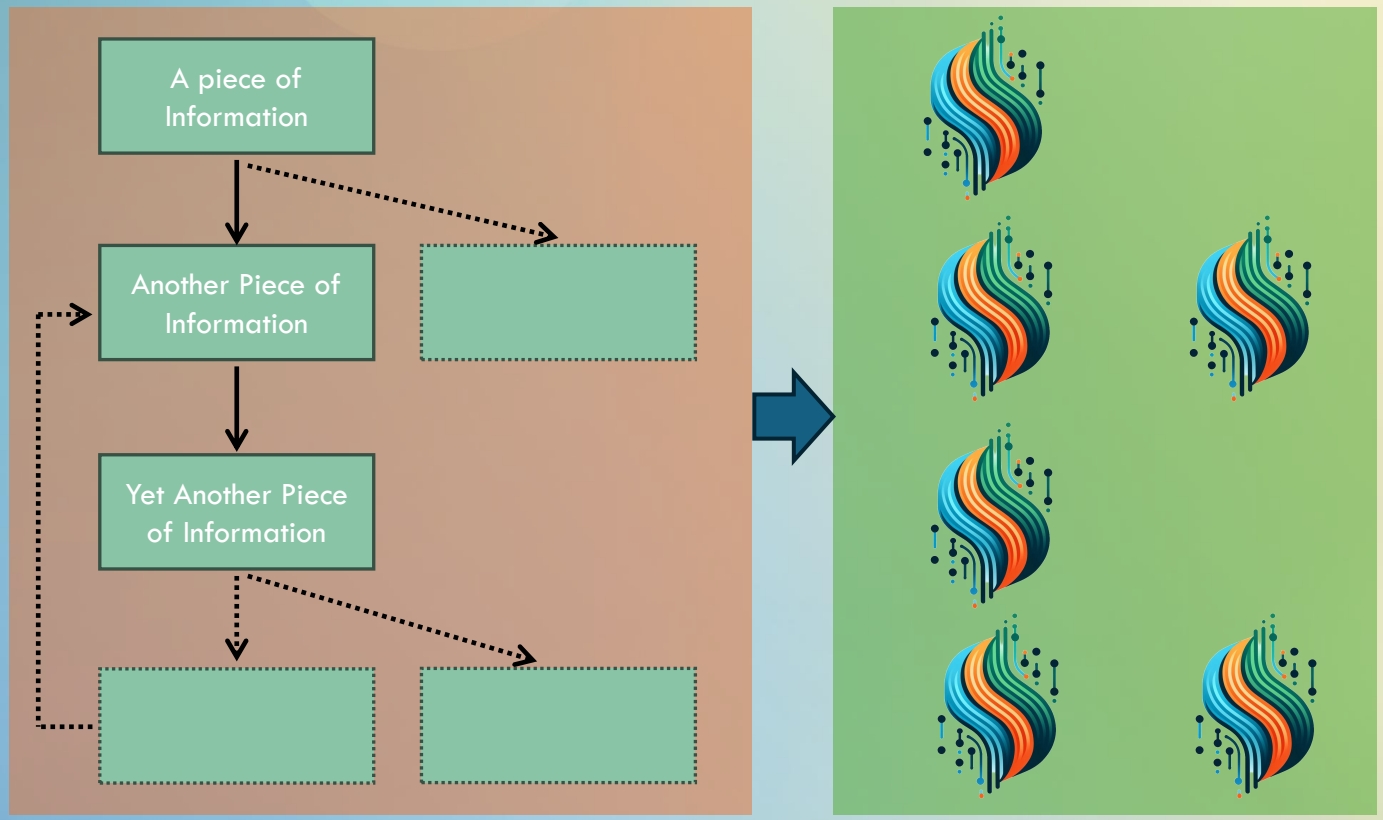
Based on RXJS

Works seamlessly with major frameworks:

- Vue.js
- Angular
- React

HOW WAVEBINDER WORKS

THE POWER OF NODE-BASED MODELING



Data is modeled as nodes in a graph.

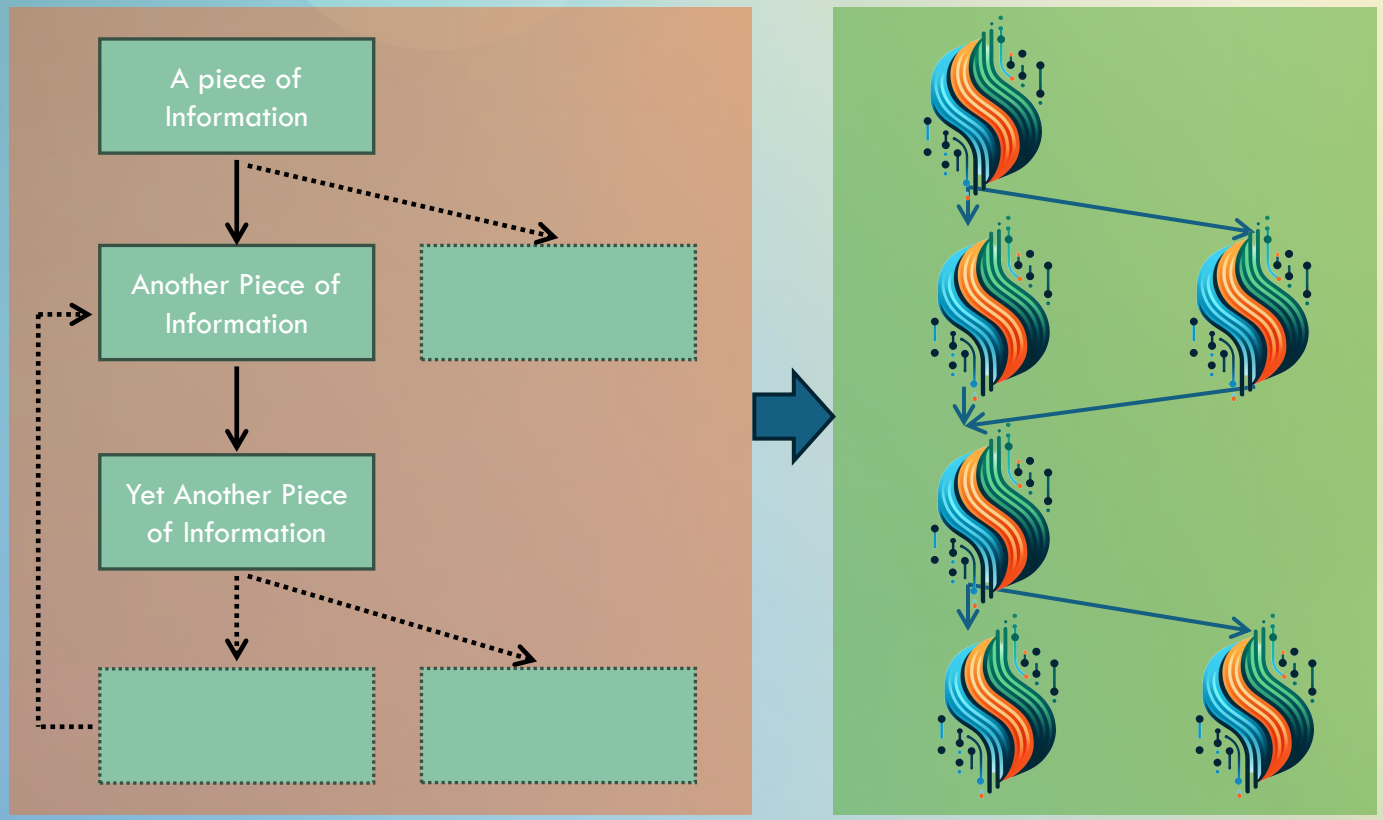
Define nodes in a JSON object like this:

```
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"
        },
        ...
```

HOW WAVEBINDER WORKS

THE POWER OF NODE-BASED MODELING

Dependencies between nodes are also specified here.




```
{
    "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"
        }
    ]
},
```

HOW WAVEBINDER WORKS

THE POWER OF NODE-BASED MODELING

```
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"
```

You can specify many properties for each node:

The type of data it contains (e.g., a single value, a list, an object, ...)

The path to store it in a model

The action required to load its value (e.g., an API call, a user selection, ...)

Its dependencies on other nodes

HOW WAVEBINDER WORKS

THE POWER OF NODE-BASED MODELING

You can specify many properties for each node:

The type of data it contains (e.g., a single value, a list, an object, ...)

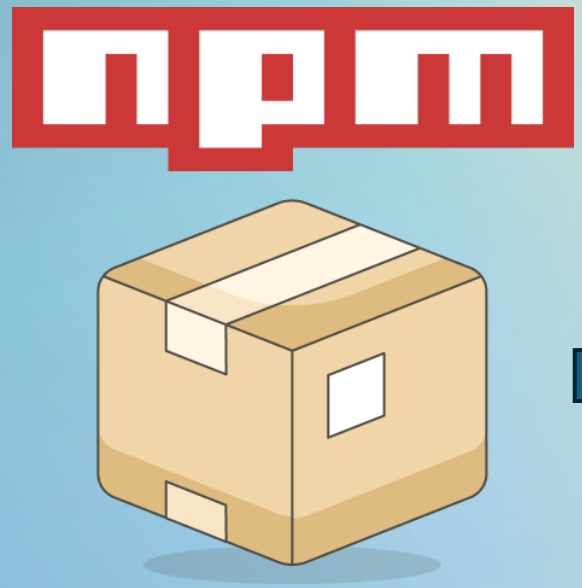
The path to store it in a model

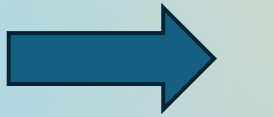
The action required to load its value (e.g., an API call, a user selection, ...)

Its dependencies on other nodes

HOW WAVEBINDER WORKS

EASY INSTALLATION AND CONFIGURATION





WAVEBINDER integrates easily into existing projects.

Simply run `npm install wave-binder` to get started.

In a few steps, you can configure nodes and their dependencies using JSON.

Even for developers new to using nodes, the CLI tool simplifies the creation and management of data dependencies.

HOW WAVEBINDER WORKS

FEATURES

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

Once WAVEBINDER is installed and you've modeled your problem within the configuration structure, you're almost ready to go.

- Provide the configuration to a new instance of WAVEBINDER.
- Instruct WAVEBINDER to build the dependency network.
- Done

```
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 getNodesInfo(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 {
}
```

HOW WAVEBINDER WORKS

FEATURES

WAVEBINDER espone:

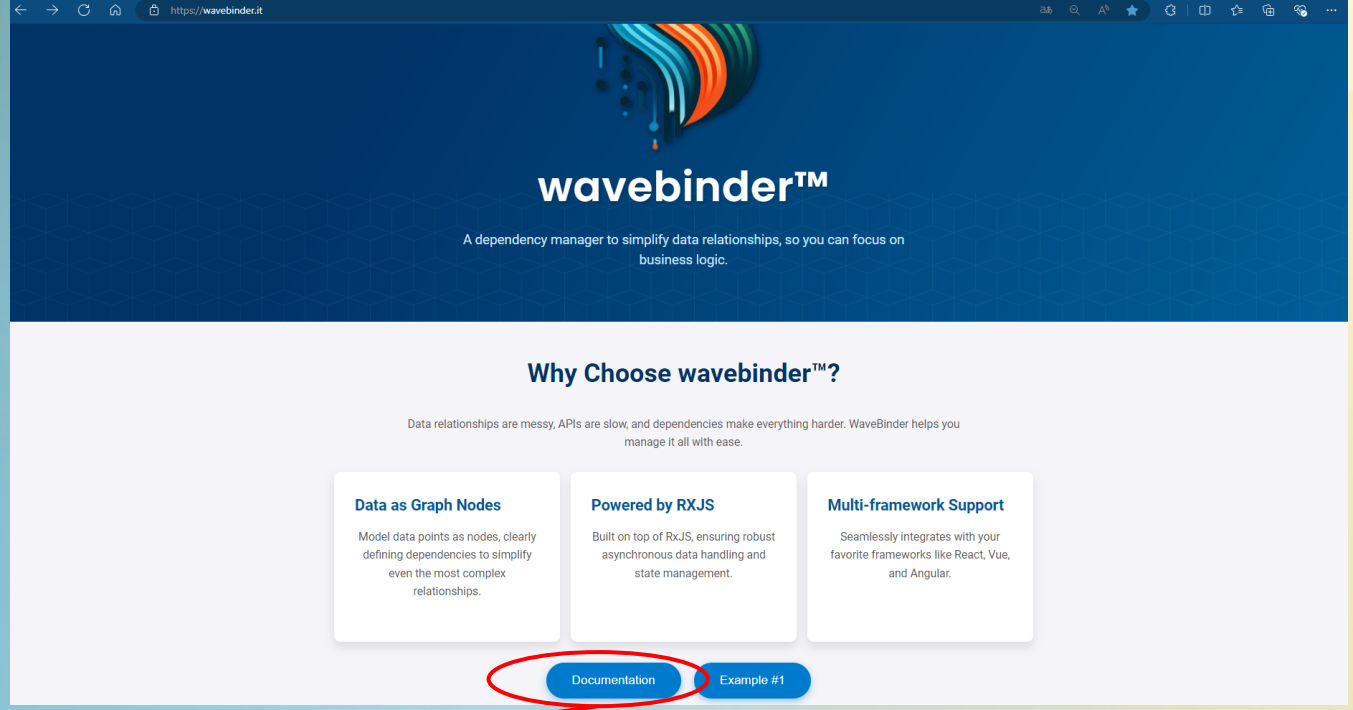
- I metodi necessari per recuperare i nodi
- Various ways to interact with them.

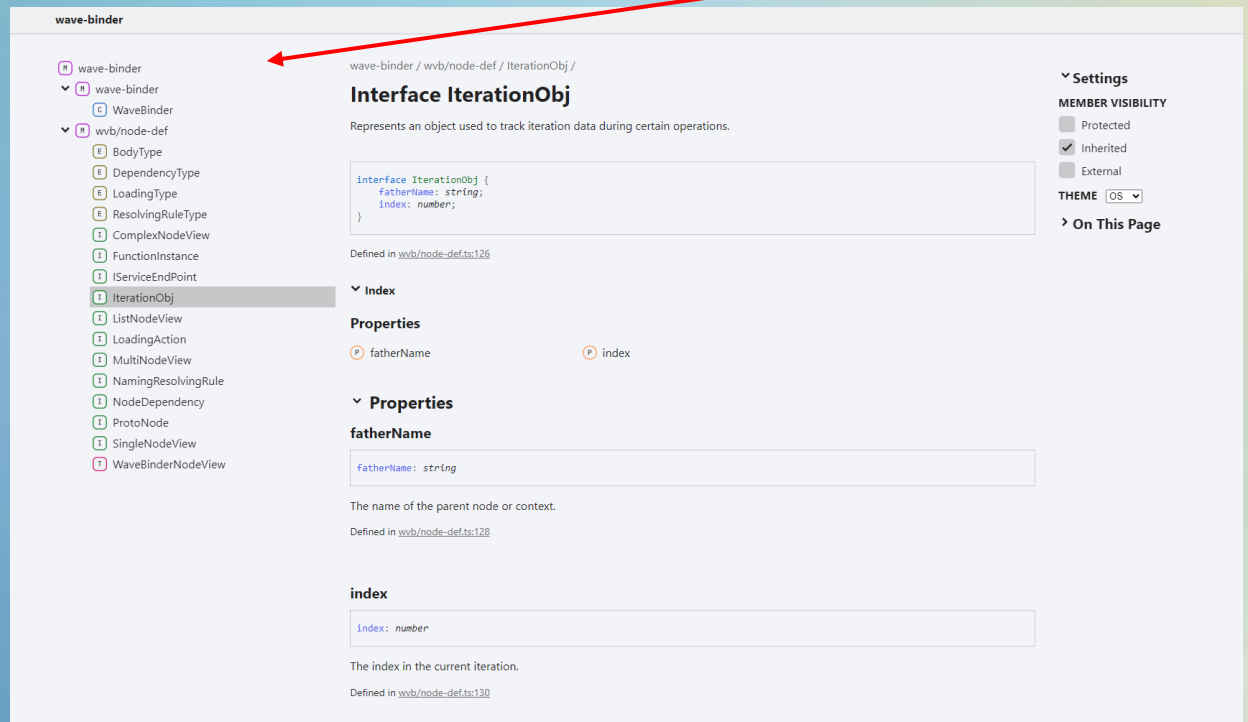
HOW WAVEBINDER WORKS

DOCUMENTATION

Every aspect of configuring and managing WAVEBINDER is documented on www.wavebinder.it .

This ensures developers have continuous guidance for advanced configurations and specific cases.





WHY DOES IT WORK?

THE SECRETS OF WAVEBINDER

A powerful library.

Years of experience.

The vision to abstract complexity once and for all, allowing everyone to focus solely on what the client truly wants: **Business Logic**.

WHY DOES IT WORK?

ORDER AND PRECISION:
DEPENDENCIES AND ACTIONS THAT ORGANIZE YOUR CODE

```
"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"
},
```

For each node, you're asked to specify two very important things:

- **Dependencies**
- **Azione di caricamento**

Dependencies are simply references to other nodes

If at any point those referenced nodes are fully and correctly loaded, the loading action is triggered

This, of course, can trigger the loading actions of other nodes

WHY DOES IT WORK?

CONSTANT CONTROL OVER EVERY ACTION IN YOUR INTERFACE

Each node keeps a record of what has happened to it

- Updates from other nodes set as its dependencies
- Loading action executed when dependencies are satisfied


```
"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"
  }
],
```