

# Requirements

- an internet connection
- Chrome: <https://www.google.com/chrome/>
- a GitHub account

# Eclipse Theia

Building Cloud & Desktop IDEs



Type**Fox**

# Outline

- **Theia Architecture**
  - Application Shell
  - React Integration
  - JSON-Form Extension

# SCOPE OF THEIA

- Support Desktop and Browser Apps
- Build engineering tools
- Allow fine-grained customization



# SUPPORT DESKTOP AND BROWSER APPS



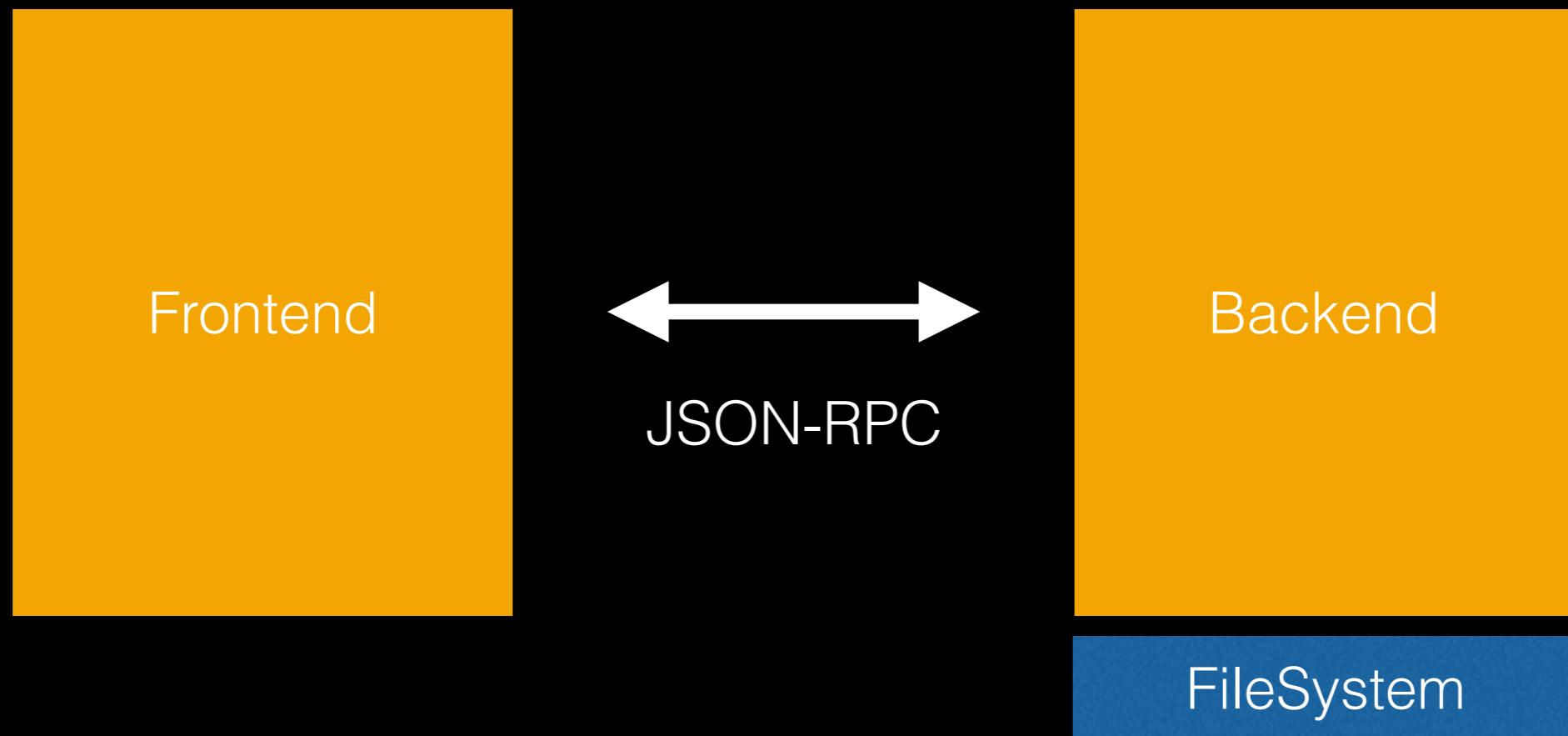
ELECTRON

Docs Blog Community Apps Userland Releases Contact

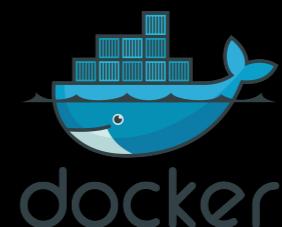
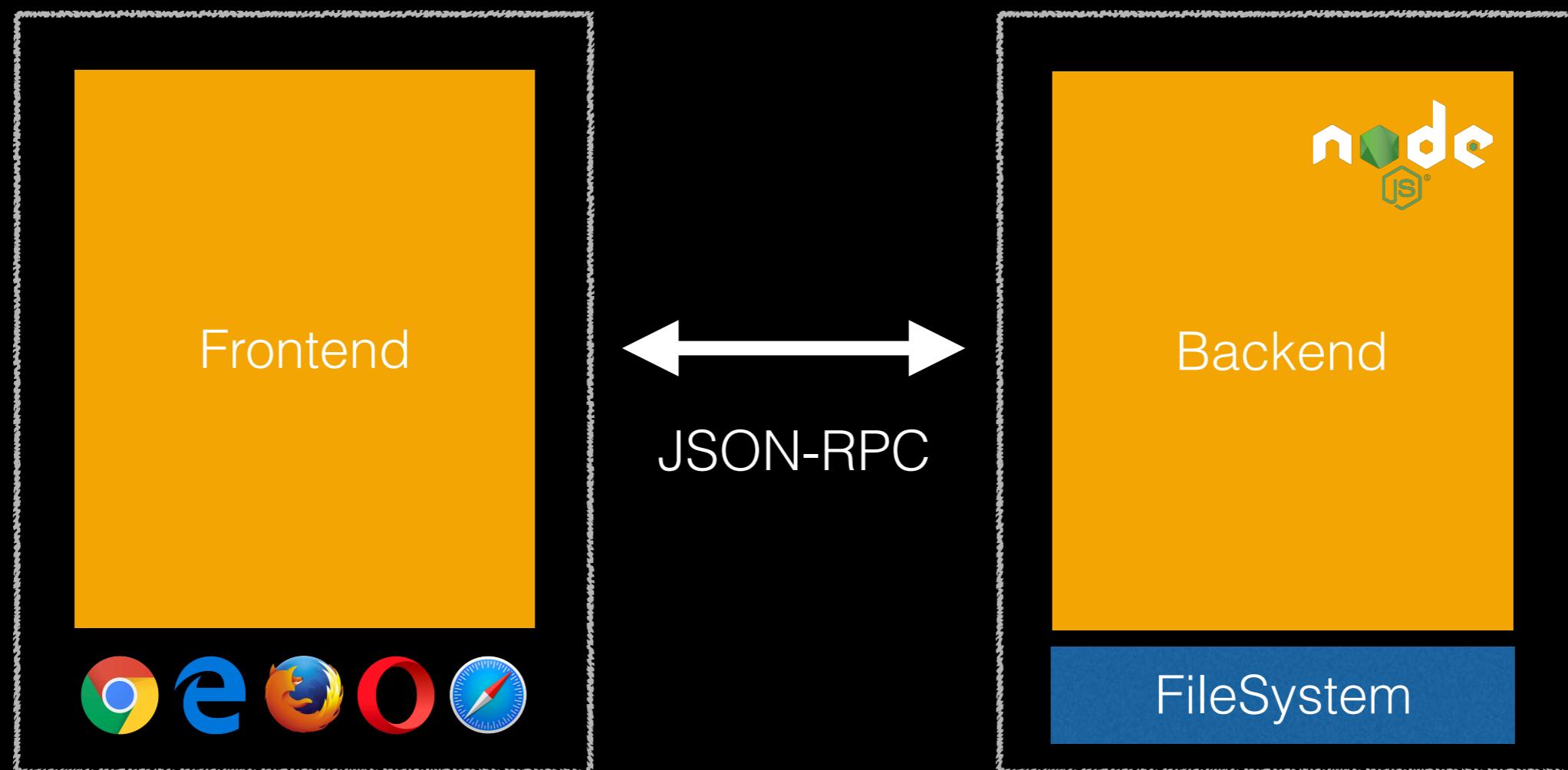


Build cross platform desktop apps with JavaScript, HTML,  
and CSS

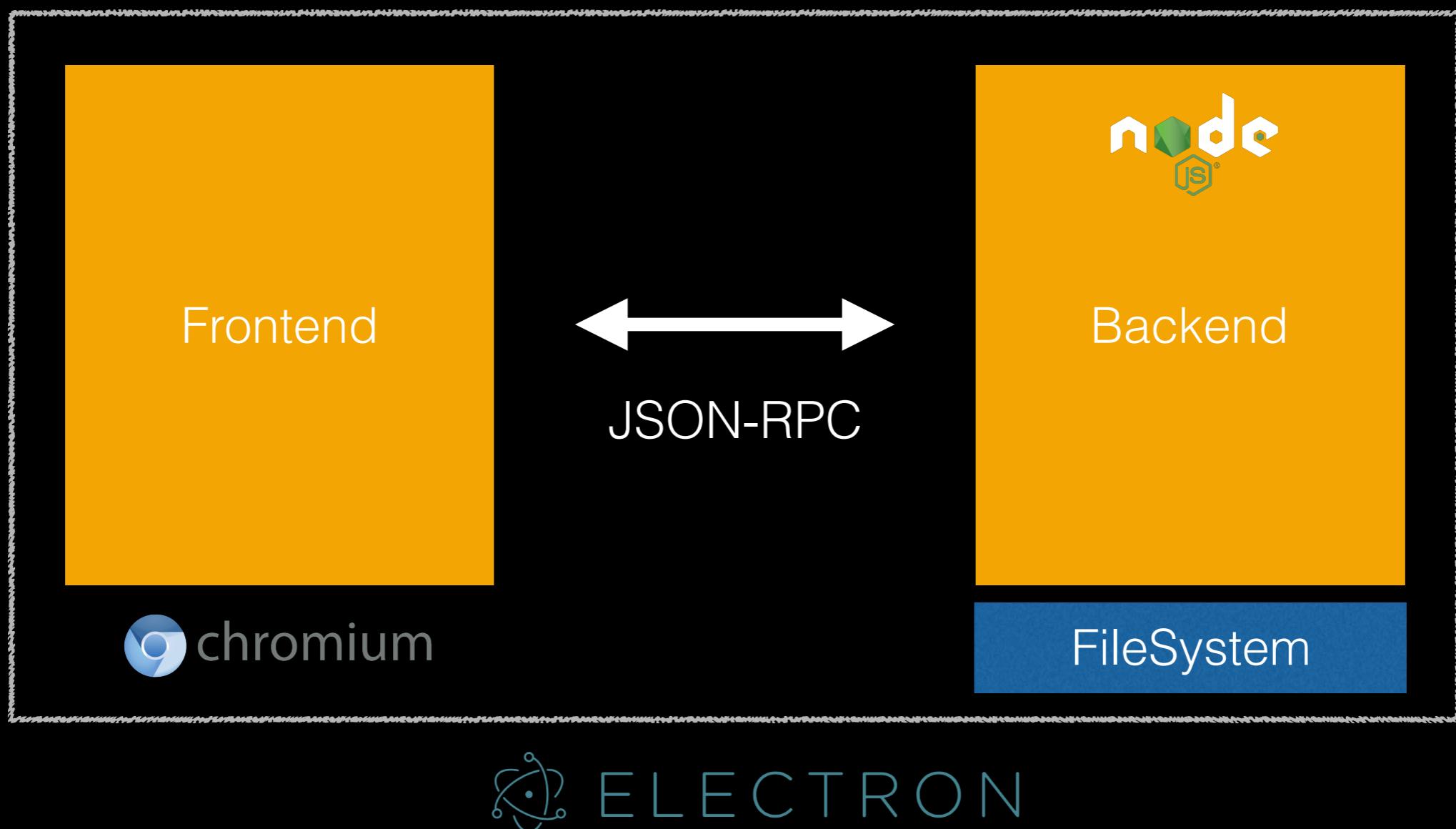
# Architecture of THEIA



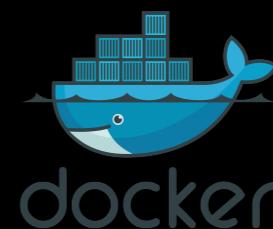
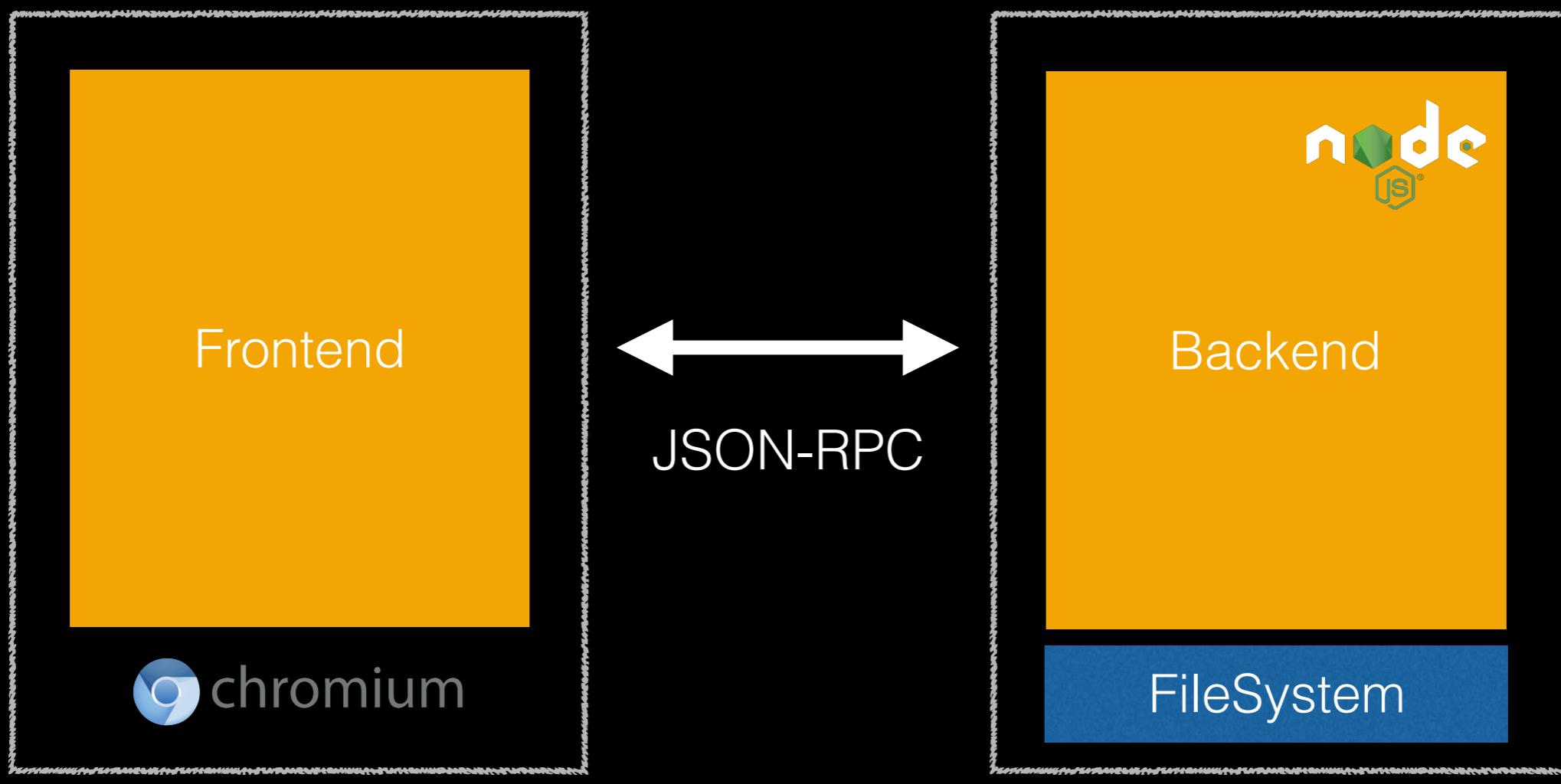
# Architecture of THEIA



# Architecture of THEIA



# Architecture of THEIA



# Demo

The screenshot shows a Theia-based workspace interface with two open files:

- Form simple-data.json**: A registration form with fields for First name, Last name, Age, Bio, Password, and Telephone.
- simple-schema.json**: The JSON schema defining the form's structure and properties.

**simple-schema.json** content:

```
1  {
2      "title": "A registration form",
3      "description": "A simple form example.",
4      "type": "object",
5      "required": [
6          "firstName",
7          "lastName"
8      ],
9      "properties": {
10         "firstName": {
11             "type": "string",
12             "title": "First name"
13         }
14     }
15 }
```

**simple-data.json** content (partial view):

```
1  {
2      "firstName": "Chuck",
3      "lastName": "Norris",
4      "age": 75,
5      "bio": "Roundhouse kicking asses since 1940",
6      "password": "noneed",
7      "telephone": null
8  }
```

Bottom status bar: 0 1 1 Ln 1, Col 1 LF UTF-8 Spaces: 4 JSON



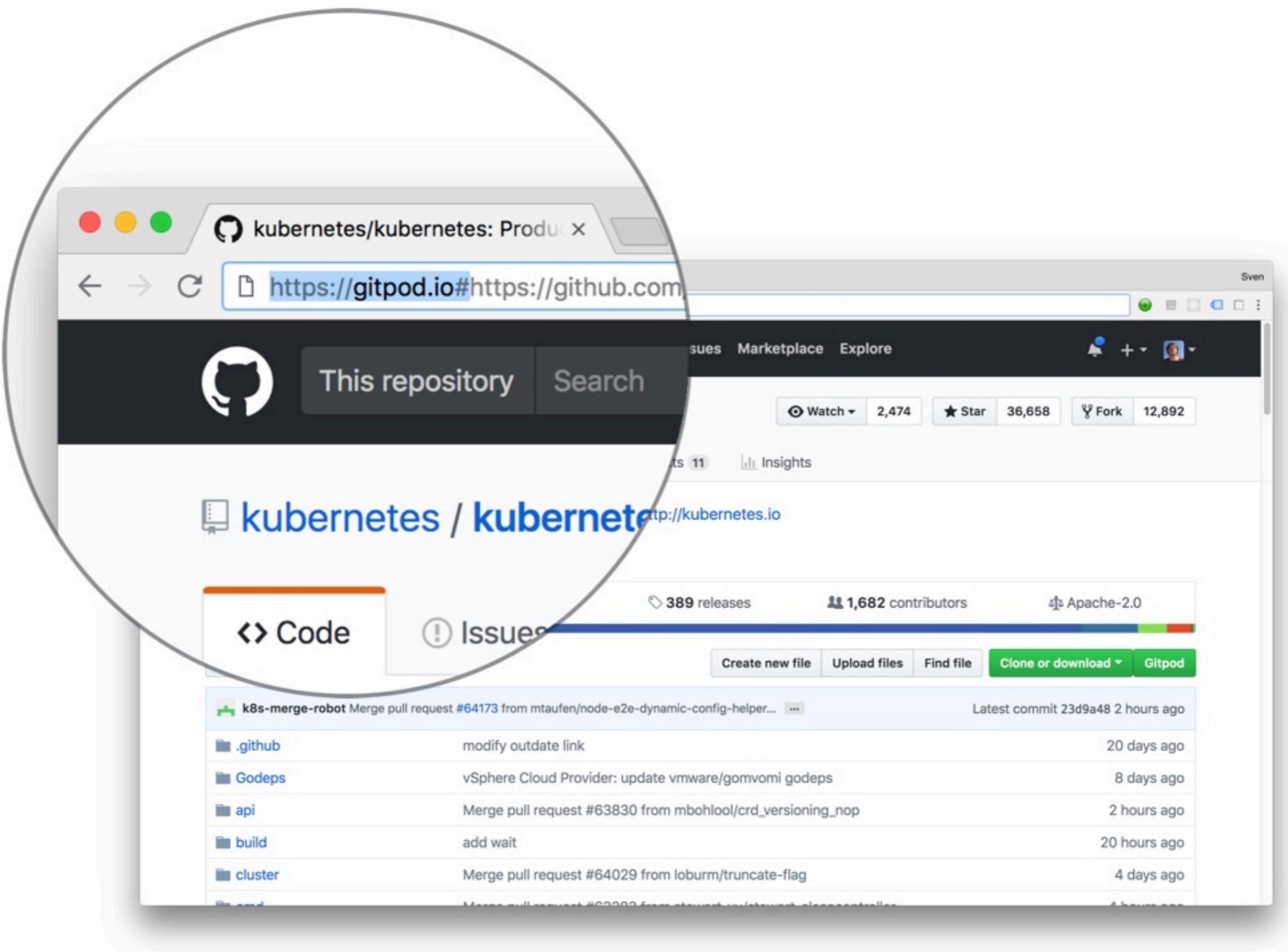
## One-Click Online IDE

for GitHub and other code hosting platforms.

<https://gitpod.io/>

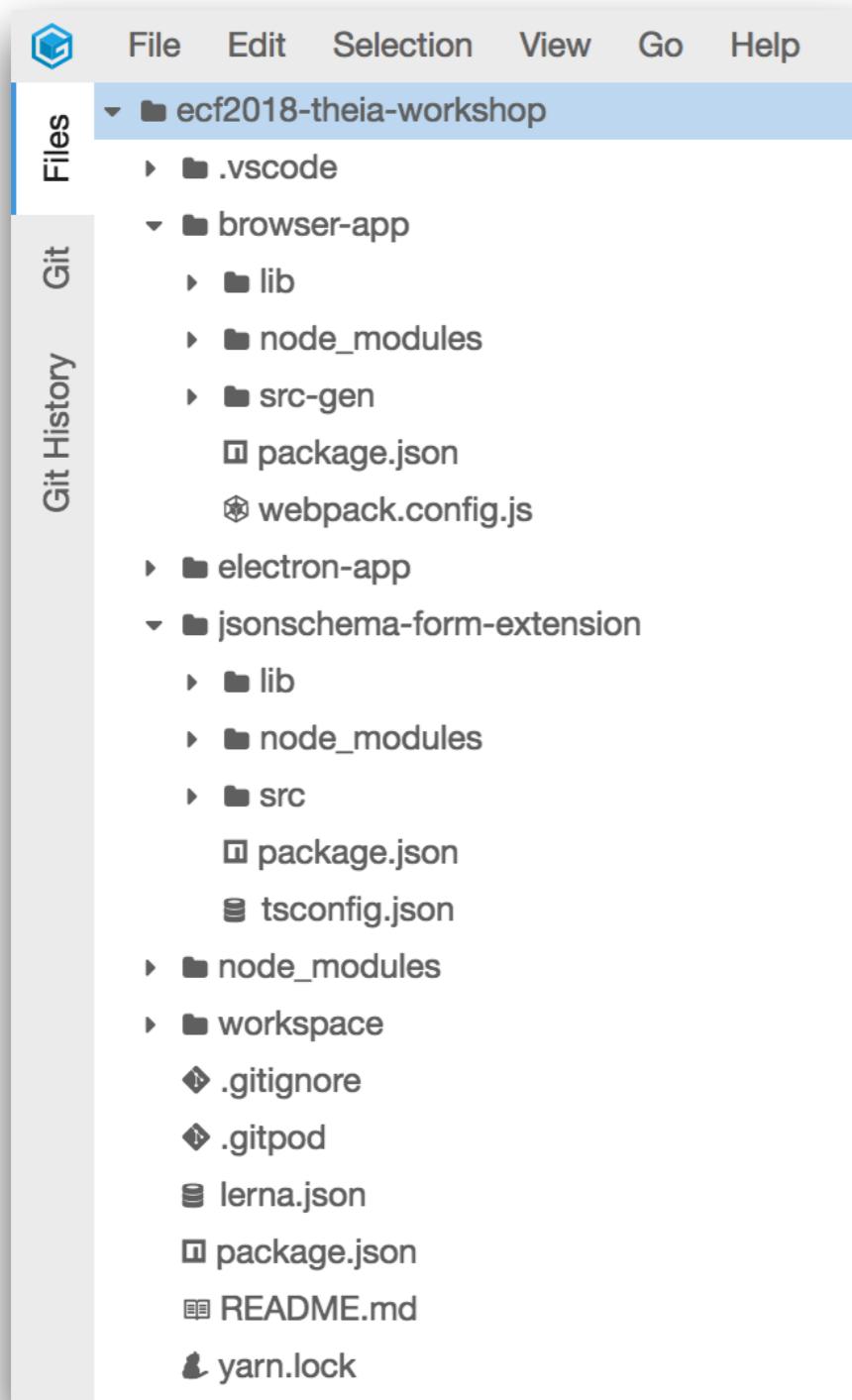


# How does Gitpod work?



**Theia applications and  
extensions are regular npm  
packages.**

# Monorepo



- root package
- extension package
- browser app package
- electron app package

# Theia Extension Generator

```
[MacKosyakov-2:my-extension kovskyakov$ yo theia-extension
[?] The extension's name my
My
  create package.json
  create lerna.json
  create .gitignore
  create README.md
  create .vscode/launch.json
  create my-extension/package.json
  create my-extension/tsconfig.json
  create my-extension/src/browser/my-frontend-module.ts
  create my-extension/src/browser/my-contribution.ts
  create browser-app/package.json
  create electron-app/package.json
yarn install v1.5.1
info No lockfile found.
[1/4] 🔎 Resolving packages...
```

<https://github.com/theia-ide/generator-theia-extension>

# Root Package



<https://lernajs.io/>

```
{  
  "private": true,  
  "scripts": {  
    "prepare": "lerna run prepare",  
    "rebuild:browser": "theia rebuild:browser",  
    "rebuild:electron": "theia rebuild:electron"  
  },  
  "devDependencies": {  
    "lerna": "2.4.0"  
  },  
  "workspaces": [  
    "jsonschema-form-extension",  
    "browser-app",  
    "electron-app"  
  ]  
}
```

# Application Package

```
{  
  "private": true,  
  "name": "browser-app",  
  "version": "0.0.0",  
  "dependencies": {  
    "@theia/core": "next",  
    "@theia/filesystem": "next",  
    "@theia/workspace": "next",  
    "@theia/preferences": "next",  
    "@theia/navigator": "next",  
    "@theia/process": "next",  
    "@theia/terminal": "next",  
    "@theia/editor": "next",  
    "@theia/languages": "next",  
    "@theia/markers": "next",  
    "@theia/monaco": "next",  
    "@theia/typescript": "next",  
    "@theia/messages": "next",  
    "jsonschema-form-extension": "0.0.0"  
  },  
  "devDependencies": {  
    "@theia/cli": "next"  
  },  
  "scripts": {  
    "prepare": "theia build --mode development",  
    "start": "theia start",  
    "watch": "theia build --watch --mode development"  
  },  
  "theia": {  
    "target": "browser"  
  }  
}
```

# Extension Package

```
{  
  "name": "jsonschema-form-extension",  
  "keywords": [  
    "theia-extension"  
,  
  "version": "0.0.0",  
  "files": [  
    "lib",  
    "src"  
,  
  "dependencies": {  
    "@theia/core": "next"  
,  
  "devDependencies": {  
    "rimraf": "latest",  
    "typescript": "latest"  
,  
  "scripts": {  
    "prepare": "yarn run clean && yarn run build",  
    "clean": "rimraf lib",  
    "build": "tsc",  
    "watch": "tsc -w"  
,  
  "theiaExtensions": [  
    {  
      "frontend": "lib/browser/jsonschema-form-frontend-module"  
    }  
,  
  ]  
}
```

# Extension Modules

- **frontend** - loaded in browser and electron frontends
- **frontendElectron** - loaded in electron frontend
  - overrides **frontend** module if present
- **backend** - loaded in browser and electron backends
- **backendElectron** - loaded in electron backend
  - overrides **backend** module if present

# Extension Module

```
export default new ContainerModule(bind => {
  bind(CommandContribution)
    .to(JsonschemaFormCommandContribution)
    .inSingletonScope();

  bind(MenuContribution)
    .to(JsonschemaFormMenuContribution)
    .inSingletonScope();
});
```

An extension module should have a default export of  
**ContainerModule | Promise<ContainerModule>** type.



# InversifyJS

A powerful and lightweight inversion of control container  
for JavaScript & Node.js apps powered by TypeScript.

<http://inversify.io/>

# Dependency Injection

```
export default new ContainerModule(bind => {
    bind(CommandContribution)
        .to(JsonschemaFormCommandContribution)
        .inSingletonScope();
});
```

---

```
@injectable()
export class JsonschemaFormCommandContribution implements CommandContribution {

    @inject(MessageService) private readonly messageService: MessageService;

    registerCommands(registry: CommandRegistry): void {
        registry.registerCommand(JsonschemaFormCommand, {
            execute: () => this.messageService.info('Hello World!')
        });
    }
}
```

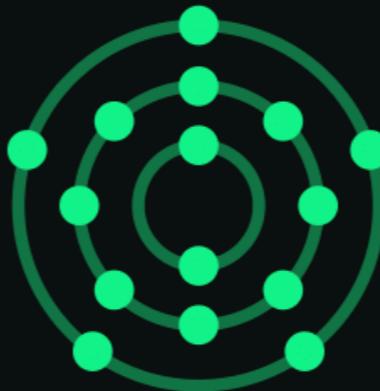
# Exercise 0

Build Theia Application

[https://github.com/TypeFox/theia-workshop/tree/exercise-0](https://github.com>TypeFox/theia-workshop/tree/exercise-0)

# Outline

- Theia Architecture
- **Application Shell**
- React Integration
- JSON-Form Extension



# PhosphorJS

a collection of libraries  
designed to make it easy  
to build high-performance,  
pluggable, desktop-style  
web applications

<https://phosphorjs.github.io/>

# Application Shell

The screenshot shows the Theia application shell interface. At the top, there's a browser-like header with tabs, showing the URL `d329f7de-cf09-463a-9cef-e9979f4e60ca.ws-eu0.gitpod.io/#/workspace/theia-workshop`. Below the header is a menu bar with File, Edit, Selection, View, Go, Debug, Terminal, and Help.

The left side features an Explorer sidebar with a tree view of the workspace files, including .vscode, browser-app, electron-app, jsonschema-form-extension, lib, node\_modules, and src. Under src, there are browser, package.json, tsconfig.json, node\_modules, workspace, .gitignore, .gitpod.dockerfile, .gitpod.yml, Development.md, lerna.json, package.json, README.md, and yarn.lock.

The main area contains several panes:

- A code editor pane showing `README.md` with content related to implementing a JSON-Form Widget Open Handler.
- A preview pane showing the rendered content of `README.md`.
- A terminal pane at the bottom right showing command-line output for tasks like `gp sync-await init`, `yarn start ..`, and `theia start ..`.
- A status bar at the bottom with various icons and text indicating the current state of the application.

# **What is a widget?**

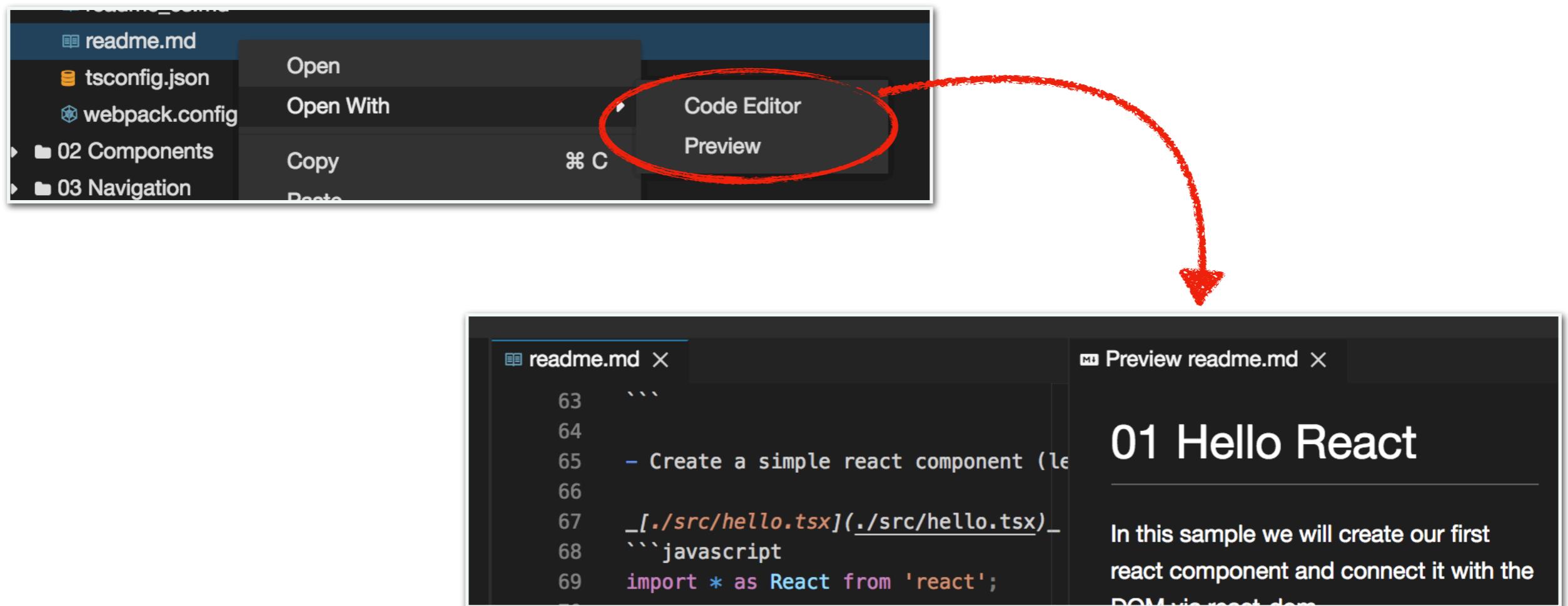
# **What is a widget?**

**An HTML element with lifecycle hooks**

# Life of a Widget

- Initialization - constructor or @postConstruct inversifyJS callback
- Attaching - on[After|Before][Attach|Detach]
- Displaying - on[After|Before][Show|Hide]
- Rendering - onUpdateRequest
- Activation - onActivateRequest
- Resizing - onResizeRequest
- Layouting - onChild[Added|Removed]
- Closing - onCloseRequest
- Destruction - dispose method

# Opening URLs



# Widget Open Handler

```
@injectable()
export class JsonschemaFormOpenHandler
  extends WidgetOpenHandler<JsonschemaFormWidget> {

  readonly id = JsonschemaFormWidget.id;
  readonly label = "Form";

  @inject(EditorManager)
  protected readonly editorManager: EditorManager;

  /**
   * Test whether this handler can open the given URI for given options.
   * Return a positive number if this handler can open; otherwise it cannot.
   *
   * A returned value indicating a priority of this handler.
   */
  canHandle(uri: URI): number {
    return this.editorManager.canHandle(uri) / 2;
  }

}
```

# URI

```
/*
 *  foo://example.com:8042/over/there?name=ferret#nose
 *  \_/   \_____/\_____/ \_____/ \____/
 *  |       authority      path        query   fragment
 *  |   / \ / \
 *  urn:example:animal:ferret:nose
 */
export class URI {
    constructor(uri?: string | Uri);
    readonly parent: URI;
    readonly scheme: string;
    readonly authority: string;
    readonly path: Path;
    readonly query: string;
    readonly fragment: string;
    resolve(path: string | Path): URI;
    toString(skipEncoding?: boolean): string;
    withPath(path: string | Path): URI;
    withoutPath(): URI;
}
```

# Path

```
/**  
 *  *      dir          base  
 *  *      |           |  
 *  *      +-----+  
 *  *      root       home/user/dir / name file ext  
 *  *      |           |   | .txt  
 *  */  
export class Path {  
    readonly isAbsolute: boolean;  
    readonly isRoot: boolean;  
    readonly root: Path | undefined;  
    readonly base: string;  
    readonly name: string;  
    readonly ext: string;  
    constructor(raw: string);  
    readonly dir: Path;  
    join(...paths: string[]): Path;  
    toString(): string;  
}
```

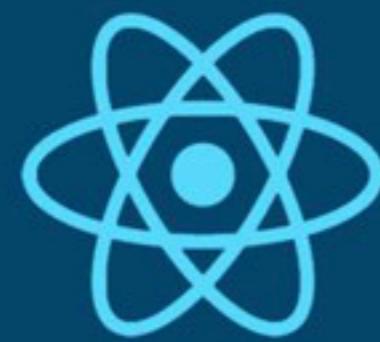
# Exercise 1

Implement Widget Open Handler

[https://github.com/TypeFox/theia-workshop/tree/exercise-1](https://github.com>TypeFox/theia-workshop/tree/exercise-1)

# Outline

- Theia Architecture
- Application Shell
- **React Integration**
- JSON-Form Extension



# React

A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

<https://reactjs.org/>

# Rendering with JSX

```
const element = <div>
  <h1>Hello World!</h1>
</div>;  
  
ReactDOM.render(
  element,
  document.getElementById( 'root' )
);
```

# Rendering with JSX

```
const name = 'World';
const element = <div>
  <h1>Hello {name}</h1>
</div>;
```

```
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

# Component & Props

```
export interface Greeting {  
    name: string  
}  
  
export class GreetingView extends React.Component<Greeting> {  
    render(): JSX.Element {  
        return <h1>Hello {this.props.name}!</h1>;  
    }  
}  
  
ReactDOM.render(  
    <Greeting name='World' />,  
    document.getElementById('root')  
);
```

# State

```
export class JsonschemaFormView extends React.Component<{}, Greeting> {

    constructor(props: {}) {
        super(props);
        this.state = {
            name: 'World'
        }
    }

    render(): JSX.Element {
        return <React.Fragment>
            <GreetingView name={this.state.name} />
            Greet <input value={this.state.name} />
        </React.Fragment>;
    }

}
```

# Event Handling

```
export class JsonschemaFormView extends React.Component<{}, Greeting> {

    constructor(props: {}) {
        super(props);
        this.state = {
            name: 'World'
        }
    }

    render(): JSX.Element {
        return <React.Fragment>
            <GreetingView name={this.state.name} />
            Greet <input value={this.state.name} onChange={this.updateName} />
        </React.Fragment>;
    }

    protected updateName = (e: React.ChangeEvent<HTMLInputElement>) =>
        this.setState({
            name: e.currentTarget.value
        });
}
```

# Component Lifecycle

```
export class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = { date: new Date() };
  }

  componentDidMount() {
    this.timerID = setInterval(() => this.tick(), 1000);
  }
  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick = () => this.setState({
    date: new Date()
  });
  render() {
    return <div>
      <h1>Hello, world!</h1>
      <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
    </div>;
  }
}
```

# Widget Integration

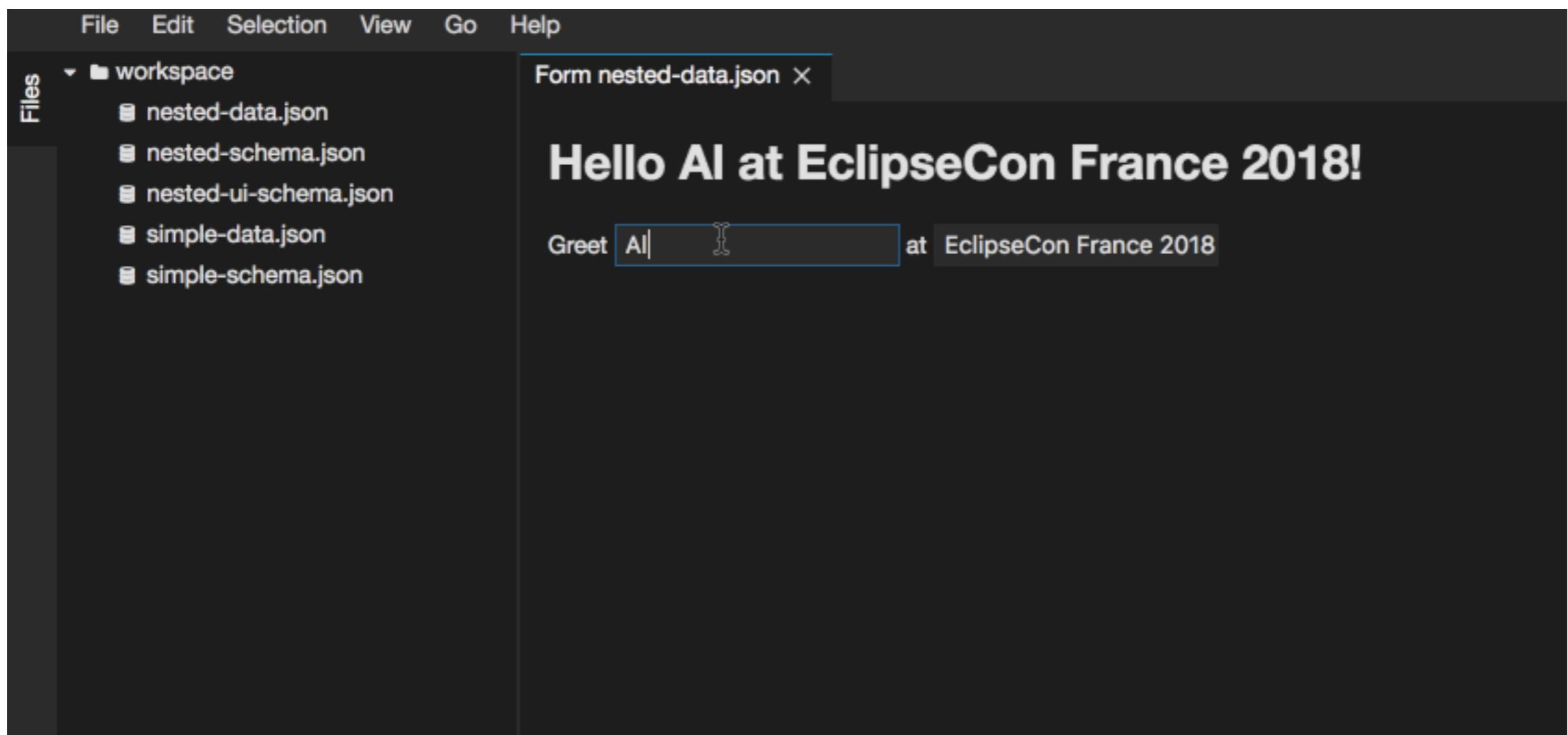
```
@injectable()
export class JsonschemaFormWidget extends BaseWidget {

    @postConstruct()
    protected async init(): Promise<void> {
        this.toDispose.push(Disposable.create(() =>
            ReactDOM.unmountComponentAtNode(this.node)
        ));
        ReactDOM.render(<JsonschemaFormView />, this.node);
    }

}
```

# Exercise 2

## Improve React Component



<https://github.com/TypeFox/theia-workshop/tree/exercise-2>

# Outline

- Theia Architecture
- Application Shell
- React Integration
- **JSON-Form Extension**

# Disposable

```
export interface Disposable {  
    dispose(): void;  
}
```

---

```
const disposable = commands.registerCommand(command);  
disposable.dispose();
```

# Disposable

```
export interface Disposable {  
    dispose(): void;  
}
```

---

```
const disposable = commands.registerCommand(command);  
disposable.dispose();
```

Unregister a command

# Disposable Collection

```
toDispose: DisposableCollection;  
  
componentWillMount(): void {  
    this.toDispose.push(this.schemaStorage);  
}  
  
componentWillUnmount(): void {  
    this.toDispose.dispose();  
}
```

# Disposable Collection

```
toDispose: DisposableCollection;
```

```
componentWillMount(): void {  
    this.toDispose.push(this.schemaStorage);  
}  
}
```

```
componentWillUnmount(): void {  
    this.toDispose.dispose();  
}  
}
```

Collect all widget services and listeners!

# Disposable Collection

```
toDispose: DisposableCollection;
```

```
componentWillMount(): void {  
    this.toDispose.push(this.schemaStorage);  
}
```

```
componentWillUnmount(): void {  
    this.toDispose.dispose();  
}
```

Exceptions are handled

# Event Handling

```
protected readonly onDidChangeEmitter = new Emitter<T>();
readonly onDidChange: Event<T> = this.onDidChangeEmitter.event;
```

---

```
protected reconcile(): void {
  const schema = this.parse();
  this.onDidChangeEmitter.fire(schema);
}
```

---

```
this.schemaStorage.onDidChange(schema =>
  this.setState({ schema })
)
```

# Event Handling

```
protected readonly onDidChangeEmitter = new Emitter<T>();
readonly onDidChange: Event<T> = this.onDidChangeEmitter.event;
```

---

```
protected reconcile(): void {
  const schema = this.parse();
  this.onDidChangeEmitter.fire(schema);
}
```

Emitting

```
this.schemaStorage.onDidChange(schema =>
  this.setState({ schema })
)
```

# Event Handling

```
protected readonly onDidChangeEmitter = new Emitter<T>();
readonly onDidChange: Event<T> = this.onDidChangeEmitter.event;
```

---

```
protected reconcile(): void {
  const schema = this.parse();
  this.onDidChangeEmitter.fire(schema);
}
```

---

```
this.schemaStorage.onDidChange(schema => Listening
  this.setState({ schema })
)
```

# Data Flow

The screenshot shows a Theia-based IDE interface with a dark theme. On the left, there's a registration form titled "A registration form". It contains fields for First name, Last name, Age, Bio, Password, and Telephone. The "First name" field has "Chuck" entered. The "Last name" field has "Norris" entered. The "Age" field has "75" entered. The "Bio" field has "Roundhouse kicking asses since 1940" entered. The "Password" field has "noneed" entered. The "Telephone" field is empty. On the right, there are two code editors. The top one is titled "simple-data.json" and contains the following JSON:

```
1 {  
2   "firstName": "Chuck",  
3   "lastName": "Norris",  
4   "age": 75,  
5   "bio": "Roundhouse kicking asses since 1940",  
6   "password": "noneed",  
7   "$schema": "simple-schema.json"  
8 }
```

The bottom code editor is titled "simple-schema.json" and contains the following JSON schema:

```
1 {  
2   "title": "A registration form",  
3   "description": "A simple form example.",  
4   "type": "object",  
5   "required": [  
6     "firstName",  
7     "lastName"  
8   ],  
9   "properties": {  
10     "firstName": {  
11       "type": "string",  
12       "title": "First name"  
13     },  
14     "lastName": {  
15       "type": "string",  
16       "title": "Last name"  
17     },  
18     "age": {  
19       "type": "integer",  
20       "title": "Age"  
21     },  
22     "bio": {  
23       "type": "string",  
24       "title": "Bio"  
25     },  
26     "password": {  
27   }
```

At the bottom of the interface, there are status indicators: "0 1" on the left, "Ln 7, Col 34 LF UTF-8 Spaces: 4 JSON" in the center, and a bell icon and other small icons on the right.

# Data Flow

The screenshot shows a code editor interface with two main panes and a sidebar.

**Left Pane:** A registration form titled "A registration form". It contains fields for First name, Last name, Age, Bio, Password, and Telephone. The "First name" field has "Chuck" entered, and the "Bio" field has "Roundhouse kicking asses since 1940".

**Right Pane:** Two JSON files are displayed.

- simple-data.json:** Contains the following JSON data:

```
1 {  
2   "firstName": "Chuck",  
3   "lastName": "Norris",  
4   "age": 75,  
5   "bio": "Roundhouse kicking asses since 1940",  
6   "password": "noneed",  
7   "$schema": "simple-schema.json"  
8 }
```

A red oval highlights the word "Form Data" in the text area.
- simple-schema.json:** Contains the following JSON schema:

```
1 {  
2   "title": "A registration form",  
3   "description": "A simple form example.",  
4   "type": "object",  
5   "required": [  
6     "firstName",  
7     "lastName"  
8   ],  
9   "properties": {  
10     "firstName": {  
11       "type": "string",  
12       "title": "First name"  
13     },  
14     "lastName": {  
15       "type": "string",  
16       "title": "Last name"  
17     },  
18     "age": {  
19       "type": "integer",  
20       "title": "Age"  
21     },  
22     "bio": {  
23       "type": "string",  
24       "title": "Bio"  
25     },  
26     "password": {  
27   }
```

**Bottom Status Bar:** Shows file status (0 0 1), line/col info (Ln 7, Col 34), encoding (LF), character set (UTF-8), spaces (Spaces: 4), and file type (JSON).

# Data Flow

The screenshot shows a code editor interface with two JSON files and a registration form.

**Form simple-data.json**

```
A registration form

A simple form example.

First name*
Chuck

Last name*
Norris

Age
75

Bio
Roundhouse kicking asses since 1940

Password
noneed

Telephone
```

**simple-data.json**

```
{
  "firstName": "Chuck",
  "lastName": "Norris",
  "age": 75,
  "bio": "Roundhouse kicking asses since 1940",
  "password": "noneed",
  "$schema": "simple-schema.json"
}
```

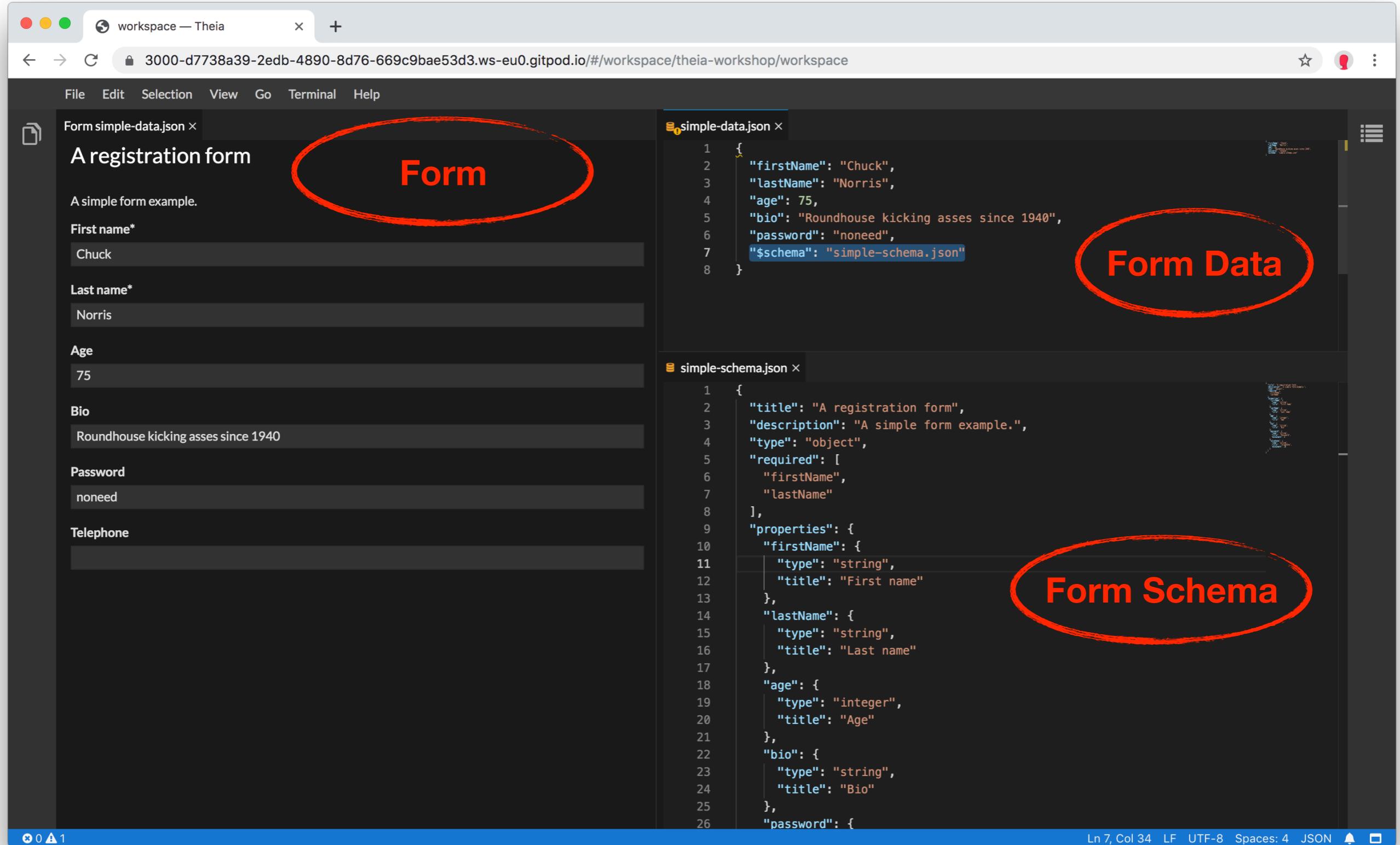
**simple-schema.json**

```
{
  "title": "A registration form",
  "description": "A simple form example.",
  "type": "object",
  "required": [
    "firstName",
    "lastName"
  ],
  "properties": {
    "firstName": {
      "type": "string",
      "title": "First name"
    },
    "lastName": {
      "type": "string",
      "title": "Last name"
    },
    "age": {
      "type": "integer",
      "title": "Age"
    },
    "bio": {
      "type": "string",
      "title": "Bio"
    },
    "password": {
      "type": "string"
    }
  }
}
```

**Form Data** (Red oval)

**Form Schema** (Red oval)

# Data Flow



The screenshot shows a code editor interface with three main components highlighted by hand-drawn red circles:

- Form**: A registration form on the left, showing fields for First name, Last name, Age, Bio, Password, and Telephone. The word "Form" is written in red inside the circle.
- Form Data**: A JSON file "simple-data.json" on the right, containing a single object with properties: firstName, lastName, age, bio, password, and \$schema. The word "Form Data" is written in red inside the circle.
- Form Schema**: Another JSON file "simple-schema.json" on the right, defining the structure for the registration form. It includes properties for title, description, type, required fields (firstName, lastName), and detailed definitions for each field like type and title. The word "Form Schema" is written in red inside the circle.

**simple-data.json**

```
1 {  
2   "firstName": "Chuck",  
3   "lastName": "Norris",  
4   "age": 75,  
5   "bio": "Roundhouse kicking asses since 1940",  
6   "password": "noneed",  
7   "$schema": "simple-schema.json"  
8 }
```

**simple-schema.json**

```
1 {  
2   "title": "A registration form",  
3   "description": "A simple form example.",  
4   "type": "object",  
5   "required": [  
6     "firstName",  
7     "lastName"  
8   ],  
9   "properties": {  
10     "firstName": {  
11       "type": "string",  
12       "title": "First name"  
13     },  
14     "lastName": {  
15       "type": "string",  
16       "title": "Last name"  
17     },  
18     "age": {  
19       "type": "integer",  
20       "title": "Age"  
21     },  
22     "bio": {  
23       "type": "string",  
24       "title": "Bio"  
25     },  
26     "password": {  
27   }
```

File Edit Selection View Go Terminal Help

Form simple-data.json ×

A registration form

A simple form example.

First name\*

Chuck

Last name\*

Norris

Age

75

Bio

Roundhouse kicking asses since 1940

Password

noneed

Telephone

(workspace — Theia) 3000-d7738a39-2edb-4890-8d76-669c9bae53d3.ws-eu0.gitpod.io/#/workspace/theia-workshop/workspace

Ln 7, Col 34 LF UTF-8 Spaces: 4 JSON

# Data Flow

- Files -> UI State
  - **Form Data File -> Form Data**
  - Form Schema File -> Form Schema
- UI State -> UI
- UI -> Form Data File

# Data Flow

The screenshot shows a code editor interface with two main panes and several annotations.

**Annotations:**

- A red oval highlights the word "Form" in the title bar of the left pane.
- A red oval highlights the word "Form Data" in the title bar of the right pane.

**Panels:**

- Left Panel (Form):** Displays a registration form with fields:
  - First name\*: Chuck
  - Last name\*: Norris
  - Age: 75
  - Bio: Roundhouse kicking asses since 1940
  - Password: noneed
  - Telephone: (empty)
- Right Panel (Data):** Displays two JSON files:
  - simple-data.json:**

```
1 {  
2   "firstName": "Chuck",  
3   "lastName": "Norris",  
4   "age": 75,  
5   "bio": "Roundhouse kicking asses since 1940",  
6   "password": "noneed",  
7   "$schema": "simple-schema.json"  
8 }
```
  - simple-schema.json:**

```
1 {  
2   "title": "A registration form",  
3   "description": "A simple form example.",  
4   "type": "object",  
5   "required": [  
6     "firstName",  
7     "lastName"  
8   ],  
9   "properties": {  
10     "firstName": {  
11       "type": "string",  
12       "title": "First name"  
13     },  
14     "lastName": {  
15       "type": "string",  
16       "title": "Last name"  
17     },  
18     "age": {  
19       "type": "integer",  
20       "title": "Age"  
21     },  
22     "bio": {  
23       "type": "string",  
24       "title": "Bio"  
25     },  
26     "password": {  
27       "type": "string",  
28       "title": "Password"  
29     }  
30   }  
31 }
```

**Status Bar:** Shows file status (0 1), line/col (Ln 7, Col 34), encoding (LF), character set (UTF-8), spaces (Spaces: 4), and JSON.

# Data Flow

The screenshot shows a code editor interface with two main panes. On the left, a file named `Form simple-data.json` displays a registration form with fields for First name, Last name, Age, Bio, Password, and Telephone. The First name field contains "Chuck", the Last name field contains "Norris", and the Age field contains "75". A red oval highlights the word "Form" in the center of the form fields. On the right, there are two files: `simple-data.json` and `simple-schema.json`. `simple-data.json` contains a JSON object with properties: `firstName: "Chuck", lastName: "Norris", age: 75, bio: "Roundhouse kicking asses since 1940", password: "noneed", $schema: "simple-schema.json"`. A red arrow points from the "Form" label in the left pane to the `$schema` field in the right pane. `simple-schema.json` defines the schema for the form, specifying types and titles for each field. A red oval highlights the word "Form Data" in the center of the schema definitions.

File Edit Selection View Go Terminal Help

Form simple-data.json ×

A registration form

A simple form example.

First name\*

Chuck

Last name\*

Norris

Age

75

Bio

Roundhouse kicking asses since 1940

Password

noneed

Telephone

Form

simple-data.json ×

```
1 {  
2   "firstName": "Chuck",  
3   "lastName": "Norris",  
4   "age": 75,  
5   "bio": "Roundhouse kicking asses since 1940",  
6   "password": "noneed",  
7   "$schema": "simple-schema.json"  
8 }
```

Form Data

simple-schema.json ×

```
1 {  
2   "title": "A registration form",  
3   "description": "A simple form example.",  
4   "type": "object",  
5   "required": [  
6     "firstName",  
7     "lastName"  
8   ],  
9   "properties": {  
10     "firstName": {  
11       "type": "string",  
12       "title": "First name"  
13     },  
14     "lastName": {  
15       "type": "string",  
16       "title": "Last name"  
17     },  
18     "age": {  
19       "type": "integer",  
20       "title": "Age"  
21     },  
22     "bio": {  
23       "type": "string",  
24       "title": "Bio"  
25     },  
26     "password": {  
27       "type": "string",  
28       "title": "Password"  
29     }  
30   }  
31 }
```

0 1

Ln 7, Col 34 LF UTF-8 Spaces: 4 JSON

# Updating Form Data

```
componentWillMount(): void {
  this.reconcileFormData();
  this.toDispose.push(
    this.props.model.onDidChangeContent(() =>
      this.reconcileFormData()
    )
  );
}
```

# Updating Form Data

```
componentWillMount(): void {
  this.reconcileFormData();
  this.toDispose.push(
    this.props.model.onDidChangeContent(() =>
      this.reconcileFormData()
    )
  );
}
```

Update the form data from initial file content.

# Updating Form Data

```
componentWillMount(): void {
  this.reconcileFormData();
  this.toDispose.push(
    this.props.model.onDidChangeContent(() =>
      this.reconcileFormData()
    )
  );
}
```

Whenever the content of form data file is changed...

# Updating Form Data

```
componentWillMount(): void {  
    this.reconcileFormData();  
    this.toDispose.push(  
        this.props.model.onDidChangeContent(() =>  
            this.reconcileFormData()  
        )  
    );  
}
```

...update the form data.

# Updating Form Data

```
protected async reconcileFormData(): Promise<void> {
  const formData = parse(this.props.model.getText()) || {};
  this.setState({ formData });
}
```

# Updating Form Data

```
protected async reconcileFormData(): Promise<void> {
  const formData = parse(this.props.model.getText()) || {};
  this.setState({ formData });
}
```

Update the state with new form data.

# Data Flow

- Files -> UI State
  - Form Data File -> Form Data
  - Form Schema File -> Form Schema
- UI State -> UI
- UI -> Form Data File

# Rendering Form Data

```
render(): JSX.Element | null {
  const { schema, formData } = this.state;
  return <Form
    schema={schema}
    formData={formData}>
    <div />
  </Form>;
}
```

Render new form data from the state.

# Data Flow

- Files -> UI State
  - Form Data File -> Form Data
  - Form Schema File -> Form Schema
- UI State -> UI
- UI -> Form Data File

# Data Flow

The screenshot shows a code editor interface with two main panes and a sidebar.

**Left Pane:** A registration form titled "A registration form". It contains fields for First name, Last name, Age, Bio, Password, and Telephone. The "First name" field has "Chuck" entered. The "Last name" field has "Norris" entered. The "Age" field has "75" entered. The "Bio" field has "Roundhouse kicking asses since 1940" entered. The "Password" field has "noneed" entered. The "Telephone" field is empty. A red oval highlights the word "Form" in the center of the form area.

**Right Pane:** Two JSON files are shown.

- simple-data.json:** Contains the following JSON data:

```
1 {  
2   "firstName": "Chuck",  
3   "lastName": "Norris",  
4   "age": 75,  
5   "bio": "Roundhouse kicking asses since 1940",  
6   "password": "noneed",  
7   "$schema": "simple-schema.json"  
8 }
```

A red oval highlights the word "Form Data" in the center of the data area.
- simple-schema.json:** Contains the following JSON schema:

```
1 {  
2   "title": "A registration form",  
3   "description": "A simple form example.",  
4   "type": "object",  
5   "required": [  
6     "firstName",  
7     "lastName"  
8   ],  
9   "properties": {  
10     "firstName": {  
11       "type": "string",  
12       "title": "First name"  
13     },  
14     "lastName": {  
15       "type": "string",  
16       "title": "Last name"  
17     },  
18     "age": {  
19       "type": "integer",  
20       "title": "Age"  
21     },  
22     "bio": {  
23       "type": "string",  
24       "title": "Bio"  
25     },  
26     "password": {  
27   }
```

**Bottom Status Bar:** Shows file status (0 1), line/col (Ln 7, Col 34), encoding (LF), character set (UTF-8), spaces (Spaces: 4), and JSON.

# Data Flow

The diagram illustrates the data flow between a registration form and its corresponding JSON data representation.

**Form** (highlighted by a red oval): A registration form containing fields for First name, Last name, Age, Bio, Password, and Telephone. The First name field contains "Chuck", the Last name field contains "Norris", and the Age field contains "75".

**Form Data** (highlighted by a red oval): The JSON representation of the form data, showing the following structure:

```
simple-data.json
1 {
2   "firstName": "Chuck",
3   "lastName": "Norris",
4   "age": 75,
5   "bio": "Roundhouse kicking asses since 1940",
6   "password": "noneed",
7   "$schema": "simple-schema.json"
8 }
```

**simple-schema.json**: The schema definition for the form data, defining the properties and their types:

```
simple-schema.json
1 {
2   "title": "A registration form",
3   "description": "A simple form example.",
4   "type": "object",
5   "required": [
6     "firstName",
7     "lastName"
8   ],
9   "properties": {
10     "firstName": {
11       "type": "string",
12       "title": "First name"
13     },
14     "lastName": {
15       "type": "string",
16       "title": "Last name"
17     },
18     "age": {
19       "type": "integer",
20       "title": "Age"
21     },
22     "bio": {
23       "type": "string",
24       "title": "Bio"
25     },
26     "password": {
27       "type": "string"
28     }
29 }
```

Annotations: A red arrow points from the Form Data JSON to the \$schema field in simple-data.json, indicating that the schema is used to validate the data.

# Updating Form Data File

```
render(): JSX.Element | null {
  const { schema, formData } = this.state;
  return <Form
    schema={schema}
    formData={formData}
    onChange={this.submit}>
    <div />
  </Form>;
}
```

Listen to the form data changes...

# Updating Form Data File

```
render(): JSX.Element | null {
  const { schema, formData } = this.state;
  return <Form
    schema={schema}
    formData={formData}
    onChange={this.submit}>
    <div />
  </Form>;
}
```

...and submit them.

# Updating Form Data File

```
protected submit = (e: IChangeEvent<any>) => {
  const model = this.props.model.textEditorModel;
  const content = model.getValue();
  const edits = jsoncparser.modify(content, [], e.formData);
  model.applyEdits(edits.map(e => {
    const start = model.getPositionAt(e.offset);
    const end = model.getPositionAt(e.offset + e.length);
    return {
      range: monaco.Range.fromPositions(start, end),
      text: e.content
    }
  }));
}
```

Compute minimal edits for original content...

# Updating Form Data File

```
protected submit = (e: IChangeEvent<any>) => {
  const model = this.props.model.textEditorModel;
  const content = model.getValue();
  const edits = jsoncparser.modify(content, [], e.formData);
  model.applyEdits(edits.map(e => {
    const start = model.getPositionAt(e.offset);
    const end = model.getPositionAt(e.offset + e.length);
    return {
      range: monaco.Range.fromPositions(start, end),
      text: e.content
    }
  }));
}
```

...apply edits to the file.

# Data Flow

- Files -> UI State
  - Form Data File -> Form Data
  - **Form Schema File -> Form Schema**
- UI State -> UI
- UI -> Form Data File

# Data Flow

The screenshot shows a code editor interface with two JSON files and a registration form.

**Form simple-data.json**

```
A registration form

A simple form example.

First name*
Chuck

Last name*
Norris

Age
75

Bio
Roundhouse kicking asses since 1940

Password
noneed

Telephone
```

**simple-data.json**

```
{
  "firstName": "Chuck",
  "lastName": "Norris",
  "age": 75,
  "bio": "Roundhouse kicking asses since 1940",
  "password": "noneed",
  "$schema": "simple-schema.json"
}
```

**simple-schema.json**

```
{
  "title": "A registration form",
  "description": "A simple form example.",
  "type": "object",
  "required": [
    "firstName",
    "lastName"
  ],
  "properties": {
    "firstName": {
      "type": "string",
      "title": "First name"
    },
    "lastName": {
      "type": "string",
      "title": "Last name"
    },
    "age": {
      "type": "integer",
      "title": "Age"
    },
    "bio": {
      "type": "string",
      "title": "Bio"
    },
    "password": {
      "type": "string"
    }
  }
}
```

**Form Data** (Red oval)

**Form Schema** (Red oval)

# Data Flow

The screenshot shows a code editor interface with two files open:

- simple-data.json**:

```
1  {
2    "firstName": "Chuck",
3    "lastName": "Norris",
4    "age": 75,
5    "bio": "Roundhouse kicking asses since 1940",
6    "password": "noneed",
7    "$schema": "simple-schema.json"
8 }
```
- simple-schema.json**:

```
1  {
2    "title": "A registration form",
3    "description": "A simple form example.",
4    "type": "object",
5    "required": [
6      "firstName",
7      "lastName"
8    ],
9    "properties": {
```

A large red arrow points from the **simple-data.json** file down to the **simple-schema.json** file, indicating the flow of data from the source file to the schema file.

Annotations in red circles:

- Form Data** (circled in red) is placed over the **simple-data.json** file.
- Form Schema** (circled in red) is placed over the **simple-schema.json** file.

# Data Flow

gitpod.io/#/workspace/theia-workshop/workspace



simple-data.json ×

```
1  {  
2    "firstName": "Chuck",  
3    "lastName": "Norris",  
4    "age": 75,  
5    "bio": "Roundhouse kicking asses since 1940",  
6    "password": "noneed",  
7    "$schema": "simple-schema.json"  
8  }
```

Form Data

simple-schema.json ×

```
1  {  
2    "title": "A registration form",  
3    "description": "A simple form example.",  
4    "type": "object",  
5    "required": [  
6      "firstName",  
7      "lastName"  
8    ],  
9    "properties": {  
10      "firstName": {  
11        "type": "string",  
12        "title": "First Name",  
13        "description": "The first name of the user",  
14        "minLength": 2, "maxLength": 50,  
15        "pattern": "^[a-zA-Z ]+$",  
16        "examples": ["John", "Jane"]  
17      },  
18      "lastName": {  
19        "type": "string",  
20        "title": "Last Name",  
21        "description": "The last name of the user",  
22        "minLength": 2, "maxLength": 50,  
23        "pattern": "^[a-zA-Z ]+$",  
24        "examples": ["Doe", "Smith"]  
25      },  
26      "email": {  
27        "type": "string",  
28        "title": "Email Address",  
29        "description": "The email address of the user",  
30        "format": "email",  
31        "examples": ["john.doe@example.com", "jane.smith@example.com"]  
32      },  
33      "password": {  
34        "type": "string",  
35        "title": "Password",  
36        "description": "The password for the user",  
37        "minLength": 8, "maxLength": 20,  
38        "pattern": "^(?=.*[a-zA-Z])(?=.*[0-9]).{8,20}$",  
39        "examples": ["P@ssw0rd", "Secure123"]  
40      },  
41      "confirmPassword": {  
42        "type": "string",  
43        "title": "Confirm Password",  
44        "description": "The password for the user",  
45        "minLength": 8, "maxLength": 20,  
46        "pattern": "^(?=.*[a-zA-Z])(?=.*[0-9]).{8,20}$",  
47        "examples": ["P@ssw0rd", "Secure123"]  
48      }  
49    }  
50  }
```

Form Schema

# Data Flow

gitpod.io/#/workspace/theia-workshop/workspace

```
simple-data.json ×
1  {
2    "firstName": "Chuck",
3    "lastName": "Norris",
4    "age": 75,
5    "bio": "Roundhouse kicking asses since 1940",
6    "password": "noneed",
7    "$schema": "simple-schema.json"
8 }
```

Form Data

```
simple-schema.json ×
1  {
2    "title": "A registration form",
3    "description": "A simple form example.",
4    "type": "object",
5    "required": [
6      "firstName",
7      "lastName"
8    ],
9    "properties": {
```

Form Schema

# Form Schema Tracking

```
new ReferencedModelStorage(  
    model, modelService, '$schema', { default: {} }  
)
```

# Form Schema Storage

```
new ReferencedModelStorage(  
    model, modelService, '$schema', { default: {} }  
)
```

Form Data File

# Form Schema Storage

```
new ReferencedModelStorage(  
    model, modelService, '$schema', { default: {} })  
);
```

A property name of a reference to Form Schema File

# Form Data -> Form Schema

```
protected async reconcileFormData(): Promise<void> {
  const formData = parse(this.props.model.getText()) || {};
  this.schemaStorage.update(formData);
}
```

---

```
componentWillMount(): void {
  this.toDispose.push(this.schemaStorage.onDidChange(schema =>
    this.setState({ schema })
  ));
}
```

---

```
render(): JSX.Element | null {
  const { schema, formData } = this.state;
  return <Form schema={schema} formData={formData} />;
}
```

# Form Data -> Form Schema

```
protected async reconcileFormData(): Promise<void> {
  const formData = parse(this.props.model.getText()) || {};
  this.schemaStorage.update(formData);
}
```

---

```
componentWillMount(): void {
  this.toDispose.push(this.schemaStorage.onDidChange(schema =>
    this.setState({ schema })
  ));
}
```

---

```
render(): JSX.Element | null {
  const { schema, formData } = this.state;
  return <Form schema={schema} formData={formData} />;
}
```

# Form Schema → UI State

```
protected async reconcileFormData(): Promise<void> {
  const formData = parse(this.props.model.getText()) || {};
  this.schemaStorage.update(formData);
}
```

---

```
componentWillMount(): void {
  this.toDispose.push(this.schemaStorage.onDidChange(schema =>
    this.setState({ schema })
  ));
}
```

---

```
render(): JSX.Element | null {
  const { schema, formData } = this.state;
  return <Form schema={schema} formData={formData} />;
}
```

# UI State -> UI

```
protected async reconcileFormData(): Promise<void> {
  const formData = parse(this.props.model.getText()) || {};
  this.schemaStorage.update(formData);
}
```

---

```
componentWillMount(): void {
  this.toDispose.push(this.schemaStorage.onDidChange(schema =>
    this.setState({ schema })
  ));
}
```

---

```
render(): JSX.Element | null {
  const { schema, formData } = this.state;
  return <Form schema={schema} formData={formData} />;
}
```

# Exercise 3

Implement UI Schema Support  
for JSON-Form Widget

**[https://github.com/TypeFox/theia-workshop/tree/exercise-3](https://github.com>TypeFox/theia-workshop/tree/exercise-3)**

# Get involved!

- Join community: <https://spectrum.chat/theia>
- Contribute: <https://github.com/eclipse-theia/theia>
- Learn more: <https://www.typefox.io/trainings>
- Self-Host with Gitpod Enterprise
  - <https://www.gitpod.io/pricing/#enterprise>

 [eclipse-theia / theia](#) Used by ▾ 175 Unwatch ▾ 167 ★ Unstar 5.9k Fork 737

[Code](#) [Issues 1,114](#) [Pull requests 58](#) [Projects 1](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

Eclipse Theia is a cloud & desktop IDE framework implemented in TypeScript. <http://theia-ide.org> [Edit](#)

[ide](#) [editor](#) [language-server-protocol](#) [electron](#) [typescript](#) [cloud-ide](#) [Manage topics](#)

 3,988 commits  131 branches  40 releases  1 environment  117 contributors  View license