Université De Montpellier - Faculté Des Sciences
Année Universitaire 2021 - 2022
M1 Informatique - GL
TER HAI823I

Rapport de projet T.E.R. M1 :

Traçabilité logicielle des GUIs d'applications web React avec l'infrastructure de traçabilité et testabilité logicielles SoftScanner

Étudiants :

Ali DABACHIL
Merwane RAKKAOUI
Lylia ZEDDAM

Encadrants :

Abdelhak-Djamel SERIAI
Bachar RIMA

Nous tenons à remercier nos encadrants M. Abdelhak-Djamel SERIAI *et* M. Bachar RIMA *pour leur accompagnement, leur bienveillance et leurs conseils précieux tout au long de ce projet qui nous ont permis de le porter à son état actuel.*

Sommaire

Partie 1

Introduction

Dans les grandes entreprises, la majorité du budget des logiciels est consacré à la maintenance et l'évolution des logiciels existants plutôt qu'au développement de nouveaux logiciels, ce dernier étant assez coûtant et risqué. Parmi les activités de maintenance et d'évolution logicielles, on note le débogage, l'optimisation, les tests, le profilage des utilisateurs, la sécurité, etc. La mise en œuvre de ces activités peut être réalisée par divers stratégies, dont l'exploitation des traces (i.e., les logs), devenant de plus en plus adoptée et systématisée dans le monde du génie logiciel. Les logs sont parfois les seules ressources disponibles pour la compréhension effective d'un système complexe à grand échelle. Assurer la qualité des informations y incluses est donc d'une importance primordiale, impactant par la suite les décisions prises par les développeurs pour la maintenance/évolution de leur système étudié. De plus, vu que les logs sont générés par des instructions de traçage (Log Printing Statements (LPSes)) instrumentant le code source du système étudié, assurer la qualité de ses dernières est également indispensable. Plusieurs solutions systématiques ont été déjà proposées pour la gestion des logs générés. Cependant, l'instrumentation du code source avec des instructions de traçage demeure coûteuse et sujette à l'erreur humaine, étant ad-hoc, majoritairement manuelle, et surtout dépendante de l'expertise des développeurs qui s'en chargent.

1.1 Contexte

Le projet SoftScanner, une collaboration entre le LIRMM et l'entreprise Berger-Levrault (BL), a été lancé comme infrastructure dédiée à l'automatisation partielle de l'instrumentation de code source avec des LPSes, selon une/plusieurs objectifs de traçage. L'infrastructure de SoftScanner est constituée d'un ensemble de petits services, appelés microservices, servant chacun comme un mini-projet. Un client interagit ainsi avec SoftScanner en fournissant un projet à analyser et en choisissant les objectifs de traçage désirés, pour ensuite lancer le processus d'instrumentation. La requête est traitée par un sous-ensemble des microservices. À la fin, le projet instrumenté est retourné au client. L'un des microservices de SoftScanner possède comme objectif le traçage des widgets graphiques d'une application web. Le microservice a été implémenté pour les applications SPA (Single-Page Applications)

créées par ***Angular*** [1].

1.2 Intérêt

Adapter l'infrastructure de traçage au ***React*** [2] représente un intérêt multiple quand on place Angular en contraste. Travailler à l'adaptation d'un tel module sur React, c'est adapter une solution prévue pour un framework rigide qui a aspiration à être autosuffisant à une librairie souple qui se complémente d'autres librairies. De plus, la technologie React est la seule qui fonctionne seulement avec du ***JSX*** [3] (extension syntaxique du Javascript permettant d'utiliser une structure similaire à l'HTML pour le rendu de composants) et prendre en charge ce langage faciliterait absolument le support de toute technologie qui utiliserait partiellement le JSX (comme Vue.js) ou l'utiliserait exclusivement. Le React est en plus la librairie la plus utilisée selon une étude du site ***stateofjs*** [4] qui publie de façon régulière des statistiques d'usage et de satisfaction des différentes technologies JavaScript.

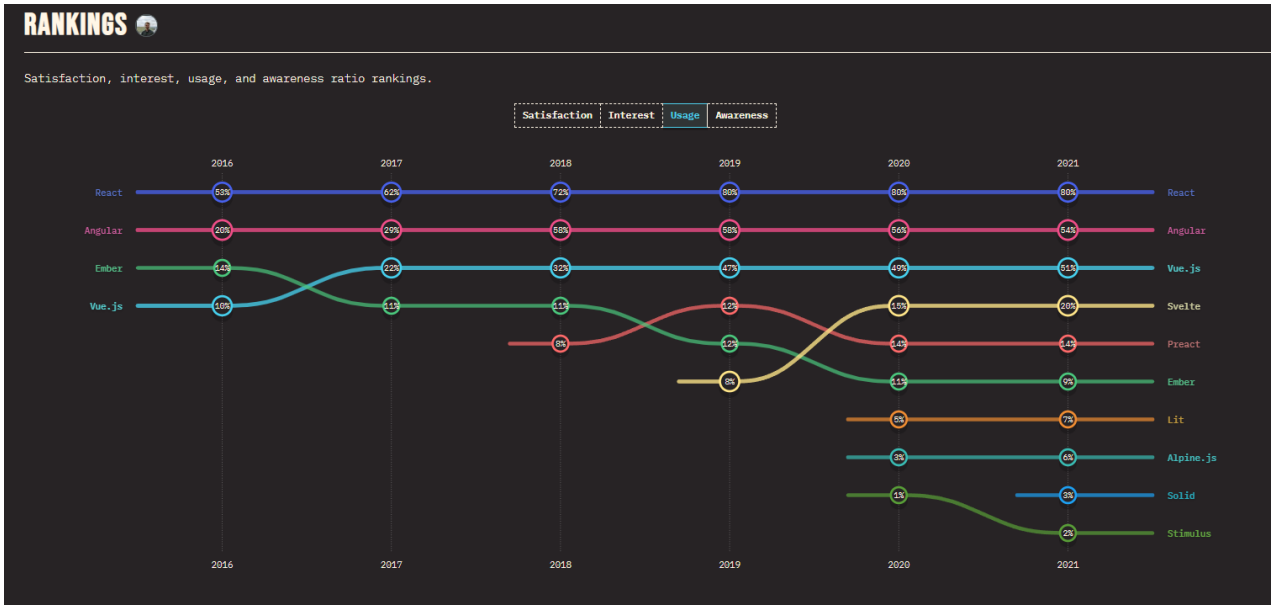

FIGURE 1.1 – Résultats de l'enquête portant sur le taux d'utilisation de différents framework front-end JavaScript

1. Framework front-end JavaScript: https://angular.io/
2. Librairie front-end JavaScript: https://reactjs.org/
3. Présentation JSX: https://reactjs.org/docs/introducing-jsx.html
4. Site web stateofjs: https://2021.stateofjs.com/en-US/libraries/front-end-frameworks

Partie 2

Problématique

Il s'agit lors de ce TER d'adapter l'infrastructure du micro-service de traçage de GUI Angular pour tracer une application écrite avec React, ce qui suppose de conserver le plus possible l'architecture logicielle originale de façon à garder les mêmes comportements de traçage qu'on avait pour du Angular. On se confrontera cependant à des problématiques inhérentes au fonctionnement d'Angular face à React.

Dichotomie logique métier/présentation

Le framework Angular impose une séparation entre la logique métier qui est codée dans un fichier TypeScript et la présentation codée dans un fichier HTML. React qui se veut une librairie permettant le plus de libertés possible n'impose pas cette façon de faire.

(a) Structure composant Angular (b) Structure composant React

FIGURE 2.1 – Comparaison structure composant Angular vs. React

Framework vs. librairie

Angular est un framework qui contient tous les outils nécessaires à la réalisation d'une application front-end, en proposant des solutions de gestion des données, de routage, de génération de composants et bien d'autres. React n'a pas été conçu dans le but d'être auto suffisant bien au contraire, il a été fait pour être complétés par d'autres librairies qui restent au choix du développeur.

TypeScript+HTML vs. JSX

Angular fonctionne à l'aide de TypeScript et d'HTML. Les solutions proposées pour le traçage de GUI Angular ont été pensées autour de l'utilisation d'outils spécifiques permettant l'analyse statique de ces langages. React fonctionne nativement en JSX, il faudra donc relever le défi d'adapter l'architecture à un nouveau langage.

Rigidité d'Angular vs. souplesse du React

Angular propose une méthode bien définie de développer une application, de la structuration du projet au contenu des fichiers. React offre une liberté qui permet par exemple de définir un composant à travers une fonction ou une classe, d'utiliser différentes solutions pour la gestion de données qui passent par attribut de classe, ou par ***Hook*** [1] pour les composants fonctionnels ou de complètement déléguer cette partie à d'autres librairies comme Redux. Ces différentes variations amenant au même résultat devront être prises en compte pour le traçage.

```
const FunctionalComponent = () => {
 const [count, setCount] = React.useState(0);

 return (
   <div>
     <p>count: {count}</p>
     <button onClick={() => setCount(count + 1)}>Click</button>
   </div>
 );
};
```

FIGURE 2.2 – Composant fonctionnel + utlisation de hook compteur de clique

```
class ClassComponent extends React.Component {
 constructor(props) {
   super(props);
   this.state = {count: 0};
 }
 render() {
   return (
     <div>
       <p>count: {this.state.count} times</p>
       <button onClick={() => this.setState({ count: this.state.count + 1 })}>
         Click
       </button>
     </div>
   );
 }
}
```

FIGURE 2.3 – Composant classe compteur de clique

1. Introduction aux Hooks: https://reactjs.org/docs/hooks-intro.html

Partie 3

Conception

3.1 Workflow

Le client, par le biais d'un fichier de configuration, choisit les évènements qu'il souhaite tracer ou non. Ensuite, tous les widgets qui comportent un traitement pour cet évènement sont extraits. Une fois extraits, pour chaque widget, on vérifie qu'il comporte un ID, si ce n'est pas le cas, un générateur d'ID se chargera de lui en attribuer un dépendamment de son type et de son contexte. Une fois que tous les widgets ont été identifiés(càd qu'on leur a attribué un ID), pour chacun d'entre eux, une instruction de traçage sera injectée dans la fonction permettant de gérer l'évènement. La trace construite consiste en l'affichage d'informations à propos de la nature du widget, de sa position dans le DOM, de quand il a été utilisé et par quel utilisateur.

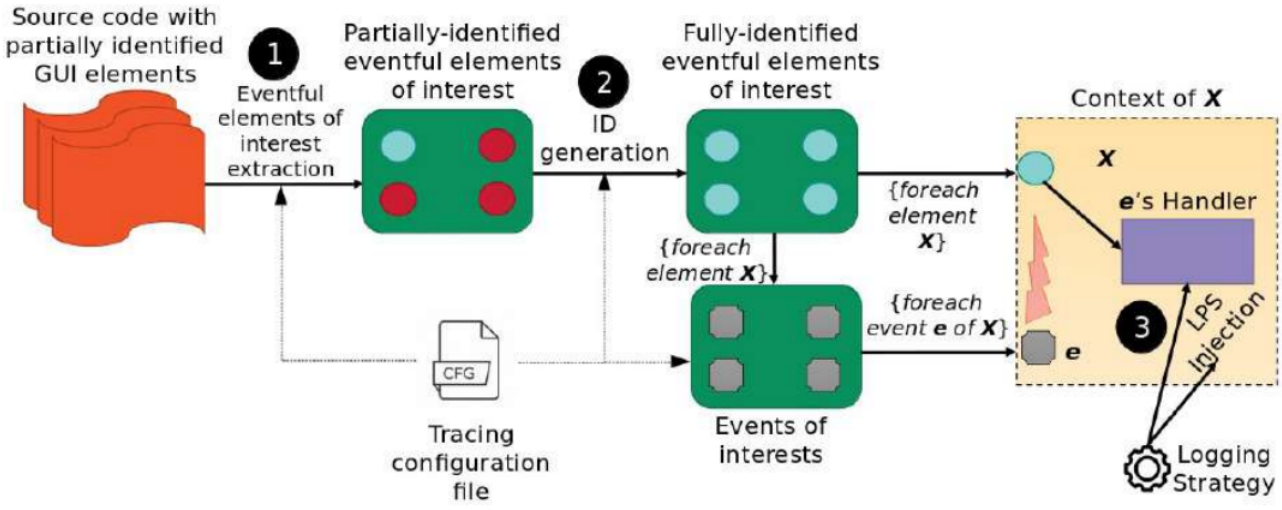

FIGURE 3.1 – Workflow général du traçage de GUI

3.2 Design

Pour instrumenter le projet, nous avons besoin de l'AST du code source du projet React car c'est par la manipulation de ce dernier que nous sommes capables d'effectuer toutes les opérations nécessaires à l'injection de code.
Afin d'intéragir avec les différents éléments du code source, chaque fichier dans lequel on aura identifié la présence de balises type HTML et contenant des évènements d'intérêt sera wrappé à la fois par un business logic wrapper et un template wrapper. Le ComponentBusinessLogicWrapper contiendra l'AST du code source produit par ***ts-morph*** [1], tandis que le parser/manipulateur d'AST HTML ***jsdom*** [2], appliqué à du code JSX, contiendra le code source en entier mais ne sera capable de manipuler que les éléments qu'il arrive à identifier comme étant du HTML.
De plus, chaque balise type HTML dans le template wrapper sera wrappé par un ElementWrapper, permettant ainsi l'utilisation de la classe ***Element*** [3] fournie par mozilla et mettant à disposition des outils de manipulation d'éléments HTML.
Enfin, tous les éléments sont visités à travers leur ElementWrapper par un ElementVisitor de telle sorte que chaque élément puisse être tracé par un visitor dédié appelé l'ElementTracer qui se chargera de le manipuler pour injecter le code de traçage dans l'évènement de cet élément.

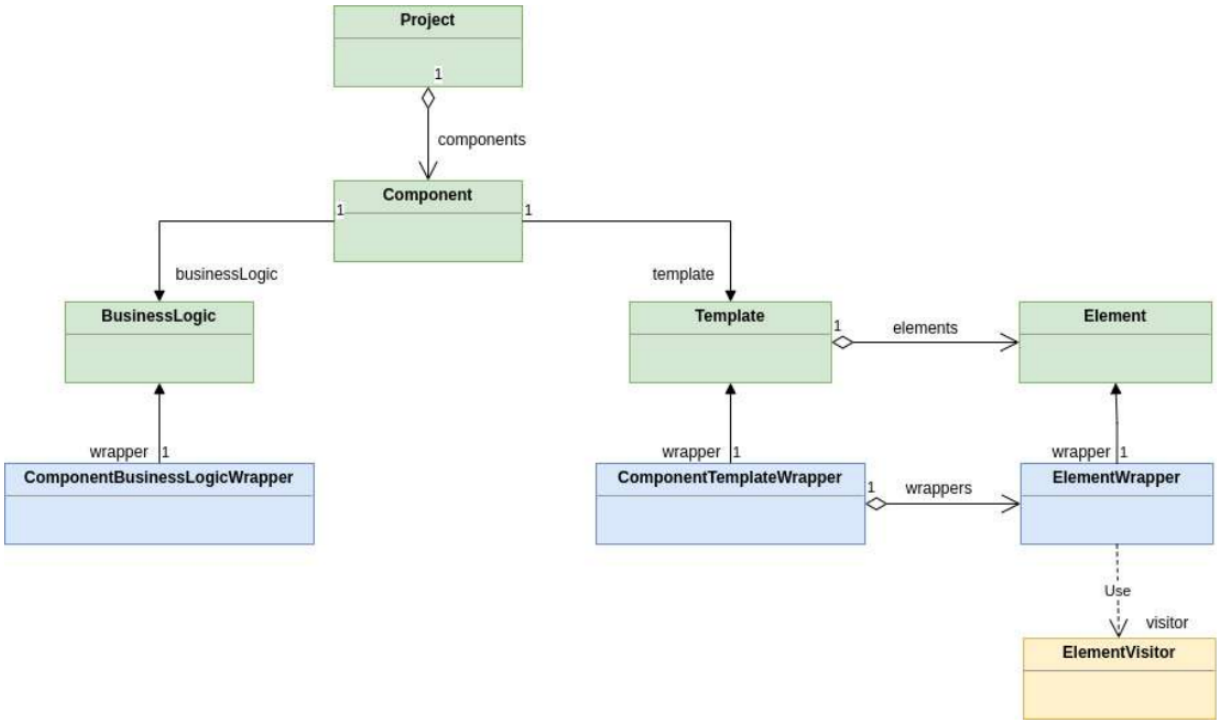

FIGURE 3.2 – Diagramme de classe UML des wrappers et de l'ElementVisitor

1. API compilateur Typescript ts-morph: https://ts-morph.com/
2. API parser jsdom: https://github.com/jsdom/jsdom
3. Element class: https://developer.mozilla.org/en-US/docs/Web/API/Element

Avant même de tracer les éléments correspondant à nos évènements d'intérêt, il faut d'abord s'assurer qu'ils soient munis d'IDs car ils feront parti de notre trace. Les éléments qui n'en ont pas seront visités par un visitor spécifique appelé ElementIDGenerator qui s'occupera de générer l'ID appropriée pour l'élément en question.
Une fois tous les éléments identifiés, on les trace. Afin de relier les widgets à leur évènement d'intérêt (par exemple, pour un bouton l'évènement de clique est à priori l'évènement d'intérêt, mais cela peut être laissé à l'appréciation de l'utilisateur pour un widget différent), il est possible de configurer l'ElementTracer avec un ElementTracerConfiguration, qui consiste en un fichier JSON obtenu d'un fichier de configuration ou de simplement utiliser le comportement par défaut.
Finalement, le processus de traçage consiste en la recherche de tous les évènements d'intérêt et à l'ajout de l'instruction correspondant à l'appel de la fonction de traçage et ce en première instruction de la fonction définissant le comportement de l'évènement.

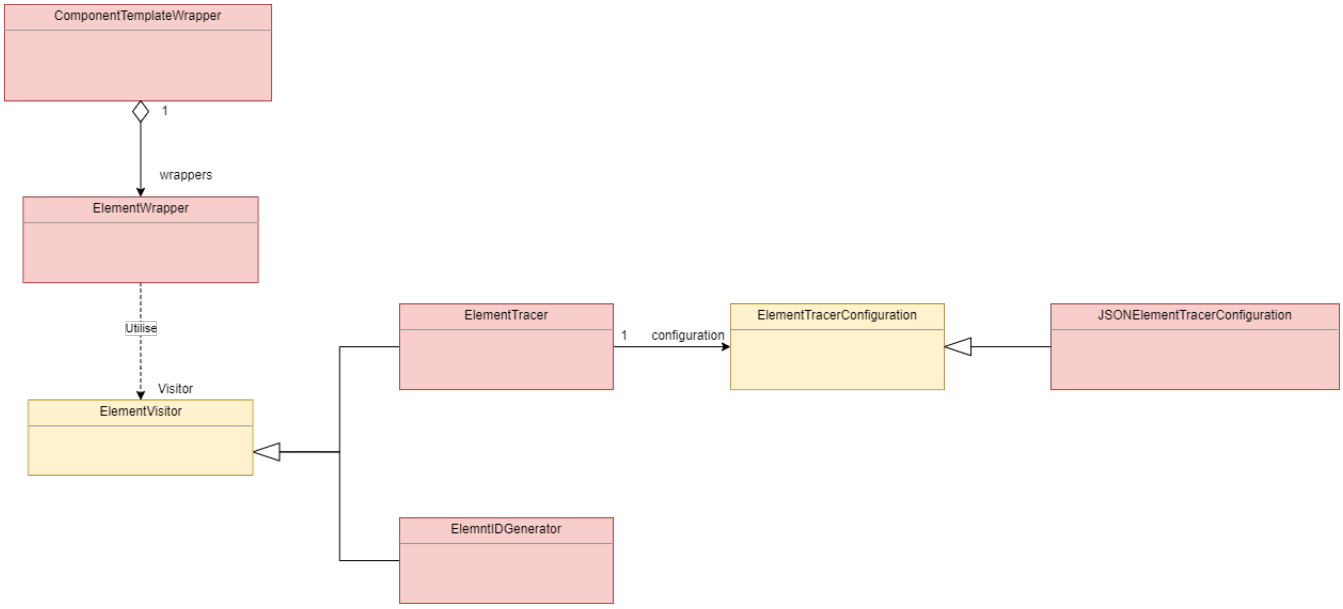

FIGURE 3.3 – Diagramme de classe UML de l'ElementTracer et du ElementIDGenerator

3.3 Du traçage Angular au traçage React

Le design présenté ci-dessus est totalement hérité de celui qui permet le traçage de GUI Angular. La conservation du design découle de la résolution des différents conflits inhérents à la nature d'Angular vs. celle de React.

En Angular, le traçage reposait sur une dichotomie entre logique métier contenue dans des fichiers TypeScript et présentation contenue dans des fichiers HTML qui n'existe pas en React. En effet, en React, tous les fichiers sont écrits en JSX, un langage mêlant le JavaScript à l'HTML et qui permet la liberté de définir la logique métier des composants ainsi que la façon dont ils doivent être affichés au sein d'un même fichier.

```
import React, { useEffect, useContext } from "react";
//[...]

const ApplicationList = (props) => {
    //[...]

  const handleDelete = async (e, id) => {
    //[...]

  };
  const safety = (nbPos) => {
    if (nbPos == 0) return <p className="text-success">SAFE</p>;
    else return <p className="text-danger">MALICIOUS</p>;
  };

  return (
    <div className="list-group">
    //[...]
        <tr onClick={() => handleApplicationSelect(Application.id)}
        key={Application.id}>
    //[...]
    </div>
  );
};

export default ApplicationList;
```

FIGURE 3.4 – Exemple de composant fonctionnel en React, alliant logique métier et présentation

L'architecture de base étant fortement couplée avec les outils de manipulation d'AST et de DOM que sont ts-morph et JSDOM, il fallait trouver un moyen de les réutiliser et d'adapter leur usage à React. Pour cela, ce sont les classes de ComponentBusinessLogicWrapper, ComponentTemplateWrapper et la classe Tracer qui est le fichier qu'on injecte dans les projets qu'on trace et qui implémente la fonction de traçage qu'il a principalement fallu adapter.

3.3.1 Adaptation du ComponentBusinessLogicWrapper

En Angular

En Angular, le ComponentBusinessLogicWrapper contenait l'AST correspondant au code source de la logique métier d'un composant. La fonction d'instrumentation pour la logique métier consistait en 3 étapes :

1. **Ajouter la déclaration d'importation de classe du Tracer :** ts-morph permet l'ajout d'une telle déclaration en mentionnant la classe à importer ainsi que le fichier où elle est définie.

```
import Tracer from "../bl-gui-tracer/tracer";
```

2. **Ajouter l'instruction d'instanciation d'un objet Tracer :** Pour chaque classe définie dans le fichier, on instancie un objet de type Tracer.

```
export class DateTimeFormComponent implements OnInit {
//[...]
    tracer = new Tracer();
//[...]
}
```

3. **Ajouter la fonction de traçage à la classe** Pour chaque classe, définir la fonction qui fait appel à la fonction de traçage du Tracer.

```
export class DateTimeFormComponent implements OnInit {
//[...]
    trace(event: Event) {
            this.tracer.trace(event);
    }
//[...]
}
```

En React

En React, c'est plus simple de ce côté-ci car il est suffisant d'injecter la déclaration d'importation non pas de la classe de traçage mais de la fonction de traçage (car nous sommes en JSX).

```
import {trace} from "../bl-gui-tracer/tracer";
```

3.3.2 Adaptation du ComponentTemplateWrapper

En Angular

En Angular, le ComponentTemplateWrapper contient l'entièreté du fichier HTML définissant la présentation du composant. Cette classe contient une méthode

d'instrumentation qui par l'intermédiaire de l'ElementWrapper et des différents tracer ainsi que de l'ElementIDGenerator, permet d'injecter les instructions nécessaires à l'exécution de la méthode de traçage au bon endroit.

```
<input type="datetime-local" class="form-control" formControlName="dateTime">
```

FIGURE 3.5 – Element du template avant traçage

```
<input type="dateTime-local" class="form-control" formControlName="dateTime"
    (focusout)="trace($event);" data-eventful-widget="true"
    id="INPUT__dateTime__f3475536-6cdc-4bf0-9943-e169a5a5107f">
```

FIGURE 3.6 – Element du template après traçage

En React

En React, il n'est pas possible de faire les choses de la même façon car c'est du JSX et non du HTML que nous manipulons. De là 2 solutions s'offrent à nous :

1. On pourrait changer de parser et utiliser un API de parser JSX, mais cela forcerait à repenser l'architecture logicielle de toutes les classes qui ont permis de tracer du Angular.
2. On pourrait, par une fonction de preprocess sur le code source, adapter les parties du fichier JSX qui s'apparentent à du HTML de façon à ce qu'elles soient reconnues par le parser comme étant vraiment du HTML, exécuter le process d'instrumentation comme on le fait pour Angular, et inverser les changements du preprocess lors d'une phase de postprocess.

L'avantage de la première solution est d'avoir un parser JSX qui nous permettrait de facilement accéder aux noeuds de l'AST et de les modifier sans avoir à effectuer de procédures de reformattage sur le code source.

C'est cependant la seconde solution que nous avons retenu car c'est celle qui permet de préserver la structure de traçage et de répliquer exactement les comportements du traçage qu'on avait pour Angular tout en réutilisant l'infrastructure logicielle fournie. L'inconvénient est que le parsing d'un fichier JSX par un parser HTML rend complexe l'ajout de fonctionnalités spécifiques au JSX qui devront toutes êtres ajoutées "à la main"(dans le sens où les différences syntaxiques devront être codées en dur).

Illustration des différents process

```
import React from "react";
import ReactDOM from "react-dom/client";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <div>
    <input
      type="button"
      onClick={(event) => {
        console.log(event);
      }}
    />
  </div>
);
```

FIGURE 3.7 – Composant React à tracer

```
import React from "react";
import ReactDOM from "react-dom/client";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <div>
    <input type="button" (click)="{(event)" =""
    (click)="trace(event);" data-eventful-widget="true"
    id="INPUT__{_console.log(event);_}}_/>__c6afe31f-3c1a-4f9c-9d95-67b1862ec070"> {
        console.log(event);
      }}
    />
  </div>
);
```

FIGURE 3.8 – Composant React après traçage sans adaptation du ComponentTemplateWrapper

Bien que la logique métier (tout ce qui n'est pas un élément type HTML) n'a pas été touchée et que le code lié à la présentation lui a été modifié (ce qui est une bonne chose), le résultat de l'instrumentation de ne produit pas du code syntaxiquement correct, et cela est dû à toutes les petites variations qu'il peut y avoir entre une balise HTML en JSX et une balise en HTML pur.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <div>
    <input
      type='button'
      onClick="{(event) __ARROWSYMBOL__ {
        console.log(event);
      }}"
    />
  </div>
);
```

FIGURE 3.9 – Composant React après phase de preprocess

La partie présentation du code obtenu respecte parfaitement la syntaxe HTML et est prêt à être parsé lors de la phase de process qui est la même pour Angular.

```
<html><head></head><body>import React from 'react';
import ReactDOM from 'react-dom/client';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <div>
    <input type="button" onClick="{(event) __ARROWSYMBOL__ {
        console.log(event);
      }}" onClick="trace();" data-eventful-widget="true"
      id="INPUT_adca0837-894a-4d3c-9136-282c61017a7e">
  </div>
);
</body></html>
```

FIGURE 3.10 – Composant React après phase de process

Après la phase de process, la génération d'ID a été faite ainsi que l'injection de l'appel à la fonction de traçage. On voit cependant que le code produit n'est pas syntaxiquement correcte pour du JSX car un comportement assez farfelu du parser HTML de jsdom est qu'il essaie constamment de convertir en HTML du texte qu'il ne comprend pas.

```
1  import React from "react";
2  import ReactDOM from "react-dom/client";
3  import { trace } from "../bl-gui-tracer/tracer";
4
5  const root = ReactDOM.createRoot(document.getElementById("root"));
6  root.render(
7    <div>
8      <input
9        type="button"
10       onClick={(event) => {
11         trace(event);
12         console.log(event);
13       }}
14       data-eventful-widget="true"
15       id="INPUT_adca0837-894a-4d3c-9136-282c61017a7e"
16     />
17   </div>
18 );
```

FIGURE 3.11 – Composant React après phase de postprocess

Après la phase de postprocess, le code est syntaxiquement correct, le déclenchement de l'event associé au widget déclenchera ainsi l'exécution de la fonction de traçage. On note par ailleurs ligne 3 que le ComponentBusinessLogicWrapper vient injecter la déclaration d'importation après que le ComponentTemplateWrapper a fini son travail, l'instrumentation par l'un et par l'autre des wrappers étant synchrone pour éviter tout problème de concurrence lors de l'écriture du fichier.

3.3.3 Adaptation de la classe Tracer

En Angular

En Angular, la classe Tracer est la classe qui est injectée dans la structure du projet à tracer(on copie le fichier dans le dossier du projet à tracer et le ComponentBusinessLogicWrapper se charge d'écrire les déclarations d'importation dans les fichiers qui en ont besoin). Elle implémente toutes les méthodes qui servent à construire une trace.

En React

En React, la classe Tracer ne peut pas être conservée car on ne peut pas injecter une classe en TypeScript pour l'utiliser dans un projet en JSX(sans restructurer le projet à tracer et importer le module Typescript du moins). La classe a donc été convertie en une suite de fonctions qui permettent de tracer exactement de la même manière qu'on le faisait en Angular. Les méthodes ont été converties en fonctions et les attributs de classe ont été convertis en variables globales.

3.3.4 Et pour le reste ?

C'est là que le choix de continuer à utiliser les parser qui ont été utilisés pour tracer des applicatons Angular prend tout son sens. Hormis des changements mineurs (par exemple (click) en Angular qui devient onClick en React) dans certaines classes, le workflow et les fonctionnalités qui ont été implémentées pour Angular fonctionnent sans problème en React que ce soit au niveau de la configuration du traçage, de la diversité des widgets tracés, des events détectés ou même au niveau du résultat de la fonction de traçage.

Partie 4

Expérimentation

4.1 Questions

➢ La solution proposée fonctionne-t-elle pour des projets de toute taille (en nombre de fichier ou/et en nombre de caractères par fichier) ?

➢ La solution proposée fonctionne-t-elle avec tous les modules qu'il est possible de combiner au React ?

➢ La solution proposée fonctionne-t-elle indépendemment de l'usage de composants fonctionnels ou de classe ?

4.2 Protocole expérimental

Pour le traçage d'Angular, une application réalisée sous Angular et comportant une multitude de widgets a été réalisé puis le module de traçage a été exécuté dessus. L'application était considérée assez large et assez diversifiée pour représenter un projet Angular lambda.
Nous avons repris cette méthode d'évaluation et avons ainsi réécrit cette application en React en de façon à pouvoir répondre aux questions définies ainsi qu'en y ajoutant spécifiquement des éléments dont nous pressentissions qu'ils allaient produire des comportements indésirables.

Application de posts Angular

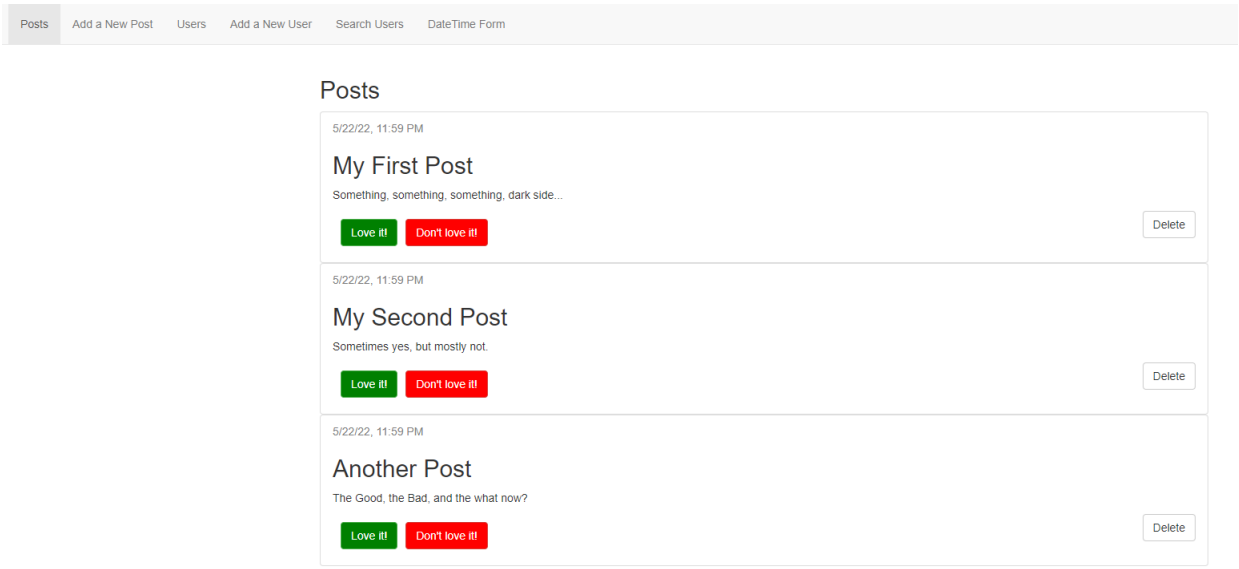

FIGURE 4.1 – Composant de liste de posts

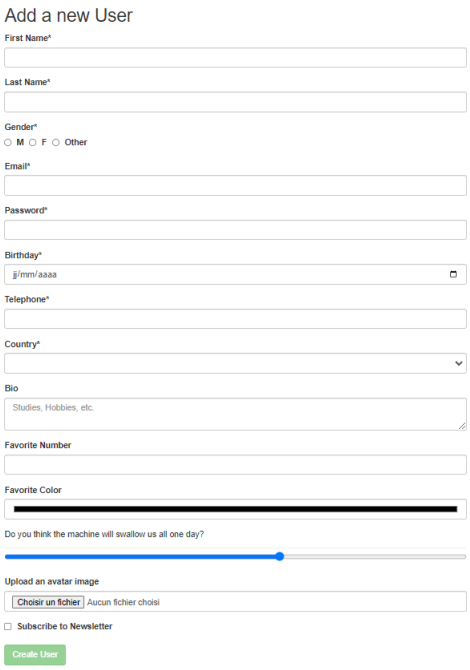

FIGURE 4.2 – Composant de formulaire d'inscription

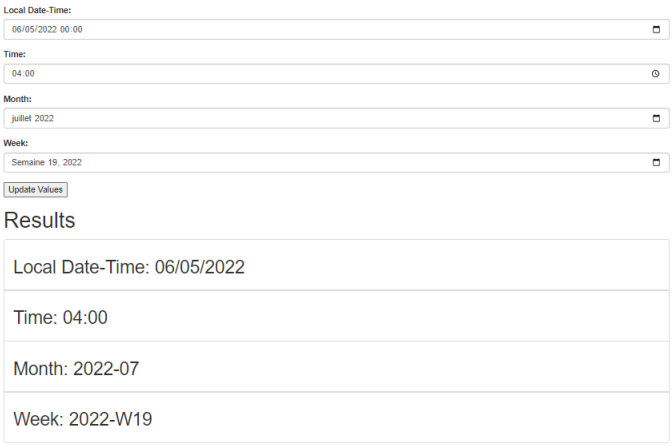

FIGURE 4.3 – Composant de manipulation de dates

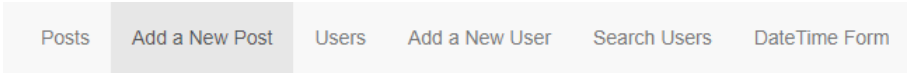

FIGURE 4.4 – Composant de Navbar

Application de posts React

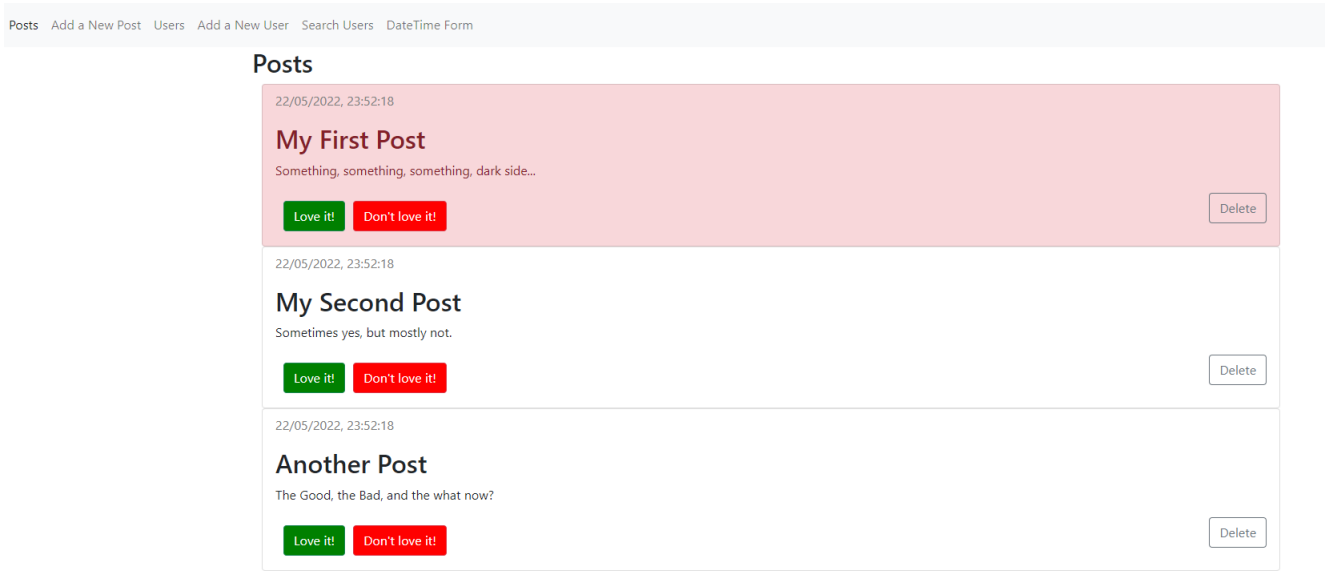

FIGURE 4.5 – Composant de liste de posts

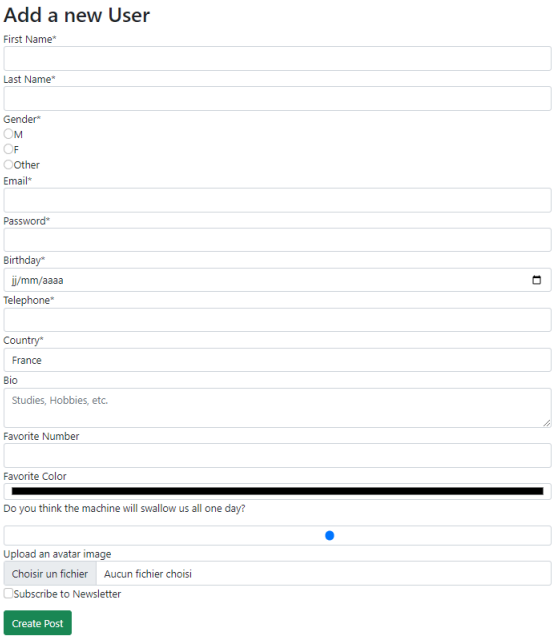

FIGURE 4.6 – Composant de formulaire d'inscription

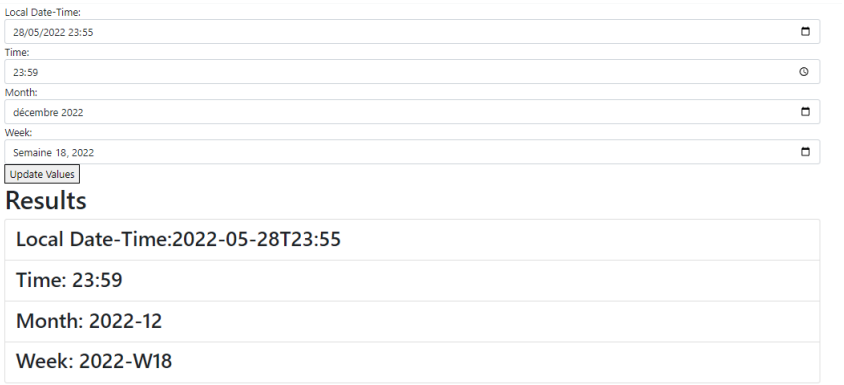

FIGURE 4.7 – Composant de manipulation de dates

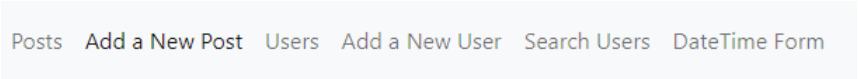

FIGURE 4.8 – Composant de Navbar

Librairies de composants utilisées : bootstrap (librairie de composants GUI), react-router-dom(librairie de composants de routages)

4.3 Résultats

Comparaison side-by-side

Composant de liste de posts

FIGURE 4.9 – Logs Angular

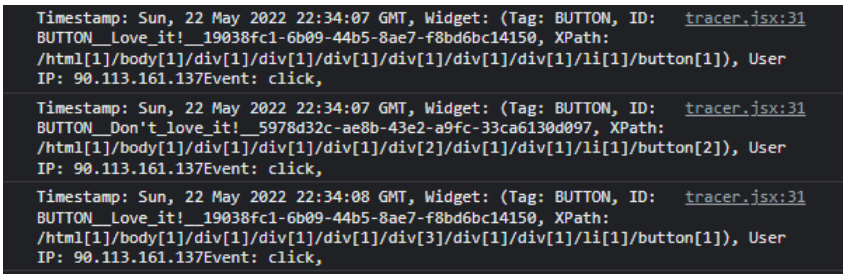

FIGURE 4.10 – Logs React

Composant de formulaire d'inscription

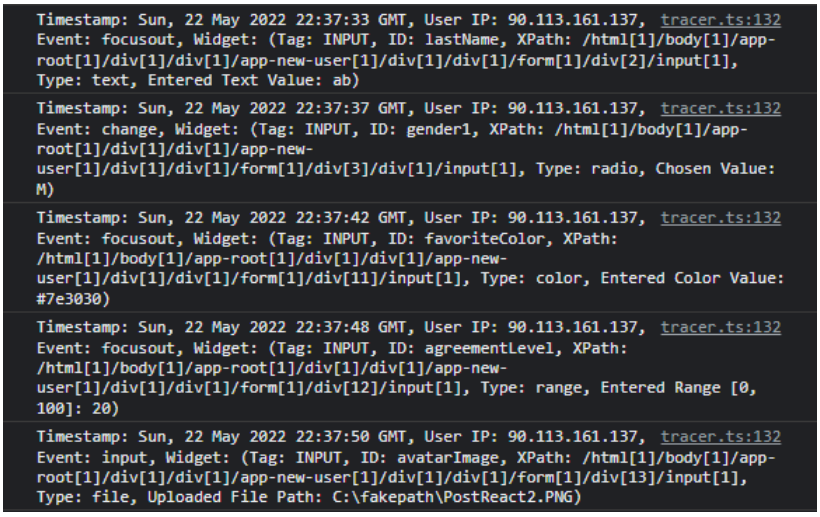

FIGURE 4.11 – Logs Angular

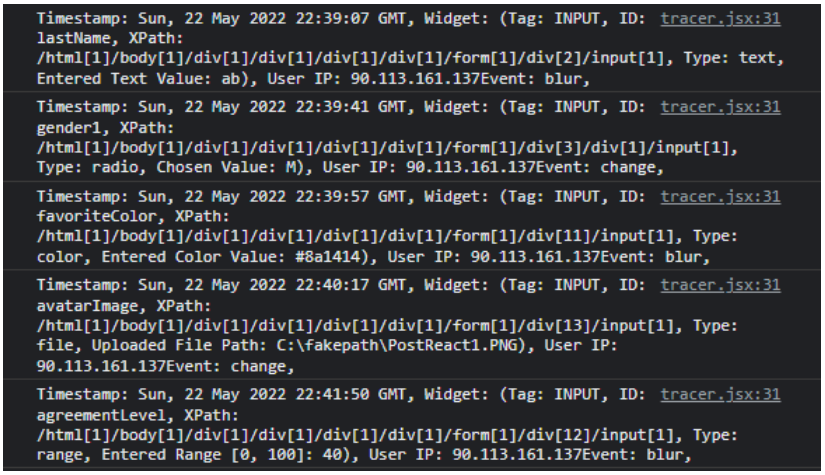

FIGURE 4.12 – Logs React

Composant de manipulation de dates

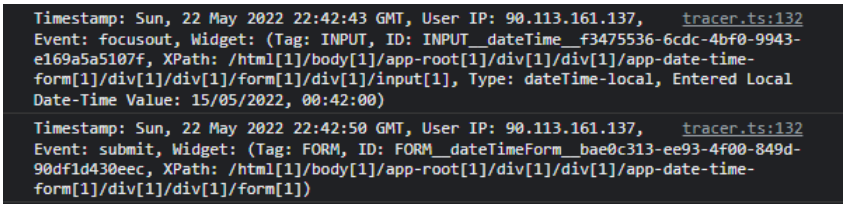

FIGURE 4.13 – Logs Angular

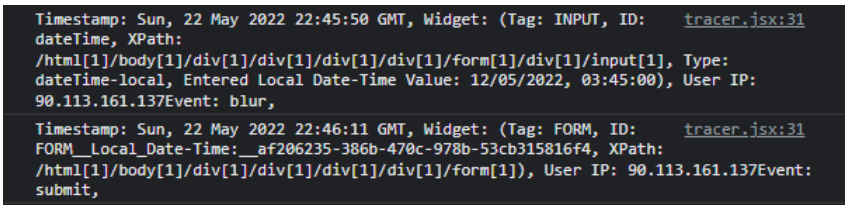

FIGURE 4.14 – Logs React

Composant de Navbar et le problème de la méta-programmation

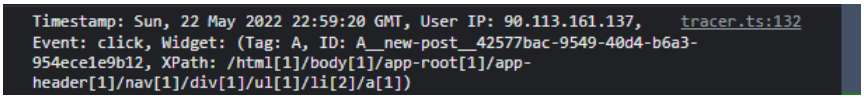

FIGURE 4.15 – Logs Angular

Pas de logs disponibles en React cependant car nous avons utilisé un composant de la libraire bootstrap pour la définir. Ce genre de composants relèvent de la méta-programmmation dans la mesure ou dans le code source, en React, une barre de navigation

bootstrap est identifiée par une balise <Navbar> alors que sur le navigateur, l'élément réellement présent dans le DOM est du type <nav>, qui est natif à l'HTML. De même pour les éléments de cette Navbar qui sont des éléments de type <Nav.Link> facilitant l'écriture d'une vraie <nav> avec des éléments de type <a>.

```
<Navbar expand="sm" bg="light" variant="light">
     <container>
    <Nav className="me-auto">
         <Nav.Link as={NavLink} to="posts">
        Posts
         </Nav.Link>
         <Nav.Link as={NavLink} to="new-post">
           Add a New Post
         </Nav.Link>
         <Nav.Link as={NavLink} to="users">
        Users
         </Nav.Link>
         <Nav.Link as={NavLink} to="new-user">
        Add a New User
         </Nav.Link>
         <Nav.Link as={NavLink} to="user-search">
        Search Users
         </Nav.Link>
         <Nav.Link as={NavLink} to="date-time-form">
        DateTime Form
         </Nav.Link>
    </Nav>
     </container>
</Navbar>
```

FIGURE 4.16 – Extrait de code d'une Navbar bootstrap en React

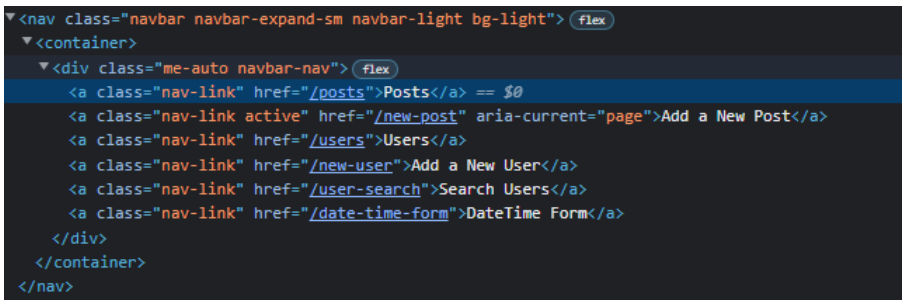

FIGURE 4.17 – La même Navbar telle que générée dans le DOM

Cet exemple est révélateur d'une très grosse limitation de notre approche. En effet, dans sa philosophie d'utilisation, React se veut une librairie qui s'utilise avec d'autres

librairies qui restent au choix de l'utilisateur, notamment des librairies fournissant des outils de génération de composants prêts à l'emploi. L'origine de cette limitation découle directement du fait qu'on analyse l'AST à la recherche d'évènements d'intérêts ou de widgets en HTML pur qui doivent donc être explicitement présents dans le code source. Si on se limite à cette stratégie d'analyse statique, le support de nouvelles librairies doit être codé à la main dans le module de traçage, ce qui pourrait être une solution satisfaisante d'un point de vue fonctionnel mais absolument pas envisageable si on recherche une bonne extensibilité au vu du nombre de librairies qu'il existe pour React.
Une piste de recherche qui pourrait faire l'objet d'un projet en soi serait l'injection de code d'une application en cours d'exécution à partir du DOM virtuel que génère React ainsi que du DOM réel affiché par le navigateur.

Partie 5

Conclusion

5.1 Bilan

La solution que nous avons proposé pour le traçage de GUI React est convenable dans la mesure où elle est une vraie adaptation de l'architecture proposée pour le traçage de GUI Angular. Cela nous permet de conserver à l'identique le workflow et les fonctionnalités qui avaient déjà été définies. Cette avancée est un signe prometteur quant à la faisabilité d'un tel projet, ainsi qu'à la possibilité de l'étendre à d'autres frameworks/librairies tels que Vue.js, Svelte ou d'autres. Bien que le futur du support des outils basés sur le JSX comme React devra probablement passer par la réalisation d'une API de parsing du JSX faite sur mesure pour l'application de traçage, le choix d'essayer de passer du HTML au JSX par diverses transformations syntaxiques a permis l'identification des similarités entre les deux langages qui devrait faciliter la réalisation d'une telle API. Cependant, plus que le module de traçage, c'est l'identification du challenge lié au support des librairies fournissant des outils de méta-programmation – entraînant ainsi la necessité d'une analyse dynamique et non pas statique – qui représente, à notre sens, le plus grand apport que l'on ait pu apporter à ce projet.

5.2 Difficultés rencontrées et apports

Bien que les fichiers sources du module de traçage pour Angular à notre disposition étaient lisibles et commentés, le travail de reverse engineering d'une telle infrastructure dans un langage qui nous était étranger jusqu'alors (le Typescript) se révéla être une tâche absolument chronophage. La réalisation du projet est ensuite passé par la maîtrise du framework Angular ainsi que de la librairie React afin de créer les connexions qui nous ont servi à adapter l'infrastructure. Ce qui représentait une difficulté lors des premières phases de ce projet s'avère finalement être un grand apport à nos compétences d'étudiants en génie logiciel dans la mesure où les qualités de reegineering, d'apprentissage des différents outils et d'analyse dont nous avons fait preuve sont des requis pour progresser en tant qu'ingénieurs logiciel.

5.3 Perspectives

Le travail que nous avons effectué représente certes un progrès dans la réalisation du module de traçage, mais il reste encore à répondre aux défis de l'extensibilité à d'autres librairies ainsi qu'à développer les outils qui permettraient de connecter ce micro-service à tous les autres qui participent au fonctionnement de SoftScanner afin de créer un véritable écosystème logiciel. Nous souhaitons bonne chance à M.Rima ainsi qu'à M.Seriai dans cette entreprise et nous espérons pourvoir continuer notre apport à leur côté s'ils le permettent.