ATAC-test

Strumenti e versioni

> Test eseguiti sul browser CEFSharp, basato su Chrome v73.0.3683.75

- Puppeteer v1.17.0
- Jest v29.7.0
- Jest-html-reporter v3.10.2
- Node v20.12.2
- npm v10.5.2

Contenuto della directory

- `Tests/`: contiene le *suites* dei test Puppeteer. Una suite è un insieme di test raggruppati in un file js.
  - `general.test.js`: test generali della UI (data e ora, apertura popup, cambio lingua, ecc...).
  - `ticket-flow.test.js`: test del flow di acquisto di un biglietto.
  - `request.test.js`: esempio di test con richiesta POST a un server (server.js).
- `Resources/`:
  - `utility.js`: contiene funzioni di utilità, utilizzate nei test. Le funzioni sono documentate all'interno del file stesso.
  - `uploadTestReport.js`: script che esegue l'upload del report dei test sul server. Utilizza l'omonima funzione `uploadTestReport(filePath)` presente in utility.js. Il report viene salvato sul server con il nome `report_hhmm.html`, dove *hhmm* è l'orario di esecuzione dei test.
- `Report/`: contiene il report HTML dei test, generato con la libreria *jest_html_reporter*.
- `jest.config.js`: file di configurazione della libreria Jest.
- `server.js`: esempio di server con API REST.
- `data.json`: JSON di esempio utilizzato per le richieste POST di *request.test.js*.

Configurazioni

- Path del report HTML: può essere configurato nel file `jest.config.js`:

```
"outputPath": "Report/test-report.html"
```

Server NodeJs

Il server NodeJs viene utilizzato per simulare il cambio dello stato di un dispositivo (es. stampante). Il server è in ascolto su `http://localhost:15001` e sulla rotta `/` accetta richieste POST con dati JSON. Nel file `data.json` sono presenti alcuni esempi.

Avviare con il comando:

```
$ node server.js 15001
```

> Avviare prima dell'esecuzione dei test.

La suites `request.test.js` invia un esempio di richiesta al server.

Esecuzione dei test

> Jest di default individua ed esegue le suites il cui file contiene nel nome ".test", come "general.test.js".

Esecuzione di tutte le suites + upload del report sul server:

```
$ npm run test-upload
```

Esecuzione di tutte le suites:

```
$ npm run test
```

I due comandi citati sono comandi custom, configurati nel `package.json` di npm. In particolare, il comando `npm run test-upload` è configurato per eseguire:

```
$ npx jest --runInBand & node Resources\\uploadTestReport.js
D:\\Sigma\\Testing\\ATAC-test\\Report\\test-report.html
```

In alternativa, è possibile utilizzare i comandi base di Jest. Esecuzione di tutte le suites:

```
$ npx jest --runInBand
```

> Utilizzare il parametro --runInBand per eseguire le suites in modo sequenziale. Di defualt vengono eseguite in parallelo, e ciò può causare problemi.

Esecuzione di un singolo test:

```
$ npm jest test_name.test.js
```

Descrizione della soluzione adottata

Per eseguire i test E2E di applicazioni web installate in apparati self-service, e in esecuzione sul browser CEFSharp, è possibile utilizzare una funzionalità presente nei browser Chrome-based, tra cui CEFSharp stesso.

La funzionalità è il *Chrome DevTools Protocol* (CDP): un protocollo che consente alle applicazioni di controllare e interagire con l'ambiente di navigazione di Chrome o Chromium, consentendo di automatizzare il browser e di eseguire una serie di operazioni come la navigazione web, l'interazione con gli elementi della pagina, il

debug del codice JavaScript e molto altro ancora. Questa funzionalità viene utilizzata in combinazione con il *remote debugging* di Chrome, che consente di connettersi a un browser remoto tramite il Chrome DevTools Protocol e ispezionare il DOM, monitorare le richieste di rete, eseguire il debug del codice JavaScript e analizzare le prestazioni dell'applicazione web.

Solitamente, i framework di test avviano una nuova istanza di un browser (come Chrome, Firefox, Edge, ecc.) che viene utilizzata per collegarsi all'URL dell'applicazione web su cui poi verranno eseguiti i test. Tuttavia, abbiamo riscontrato un problema particolare con questo metodo. L'istanza dell'applicazione aperta dal framework nel browser di test entrava in conflitto con l'istanza della stessa applicazione in esecuzione sul browser CEFSharp, ovvero quella riprodotta nel display esterno della TVM. Questo conflitto rendeva impossibile eseguire i test e rendeva anche inutilizzabile l'applicazione sul display esterno.

La soluzione adottata è stata quella di effettuare i test direttamente sull'applicazione in esecuzione su CEFSharp, senza quindi dover utilizzare un altro browser e un'altra istanza dell'applicazione stessa. Alcuni framework infatti, in particolare: Puppeteer, Playwright e Robot Framework, offrono la possibilità di collegarsi a una determinata istanza di un browser già in esecuzione ed eseguire i test su di essa, sfruttando proprio il remote debugging dei browser Chrome-based.

Come indicato precedentemente, la versione di Chrome utilizzata da CEFSharp è la 73.0.3683.75. Si tratta di una versione piuttosto datata, risalente al 2019. Ciò ha causato alcuni problemi di compatibilità con i framework sopra citati. La soluzione è stata quella di utilizzare una versione anch'essa datata del framework Puppeteer (v1.17.0), compatibile con la versione di Chrome utilizzata. Al link Supported browsers è possibile consultare le versioni di Puppeteer e le relative versioni di Chrome compatibili.

Procedura base: installazioni e configurazioni

Requisiti

- NodeJs
- comando *npm init* eseguito all'interno di una directory dedicata ai test

Configurazione della TVM

Nella TVM oggetto di test è necessario abilitare il remote debubugging del browser CEFSharp in esecuzione.

Nel Registry Editor, al percorso: `Computer\HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\SIGMA\Session Manager\Environment\Sigma\SelfService\CefSharp\Settings`

impostare l'opzione `RemoteDebuggingPort` inserendo una porta, ad esempio la 9222.

A questo punto, collegandosi all'indirizzo:

- `http://localhost:9222`: possiamo accedere alla lista di pagine aperte nel browser e aprire il Chrome DevTools su di esse.
- `http://localhost:9222/json`: possiamo visualizzare in formato JSON la lista di pagine aperte nel browser e le loro informazioni
- `http://localhost:9222/json/version`: possiamo visualizzare in formato JSON le informazioni del browser

In particolare, all'indirizzo `http://localhost:9222/json/version` vedremo un JSON come questo:

```
{
   "Browser": "Chrome/73.0.3683.75",
   "Protocol-Version": "1.3",
   "User-Agent": "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/73.0.3683.75 Safari/537.36",
   "V8-Version": "7.3.492.22",
   "WebKit-Version": "537.36 (@909ee014fcea6828f9a610e6716145bc0b3ebf4a)",
   "webSocketDebuggerUrl": "ws://localhost:9222/devtools/browser/f6d4343f-d5f2-
4765-aed5-13ff6bbaf826"
}
```

Per permettere ai framework di connettersi al browser è necessario l'indirizzo `webSocketDebuggerUrl` presente nel JSON. Questo indirizzo cambia ogni volta che il browser viene riavviato, perciò è possibile automatizzare il suo recupero con una richiesta GET all'indirizzo `http://localhost:9222/json/version`, come è stato fatto nell'esempio dei test ATAC.

> Quando si eseguono i test chiudere tutte le eventuali connessioni a http://localhost:9222, altrimenti si avranno conflitti.

Installazione di Jest

Jest in combinazione con Puppeteer offre un framework per strutturare i test utilizzando le funzioni *describe* e *it*. La funzione describe è utilizzata per raggruppare insieme i test correlati, fornendo una descrizione significativa del contesto in cui vengono eseguiti i test. All'interno di ciascun blocco describe, è possibile utilizzare la funzione it per definire singoli test, specificando le azioni da eseguire e le aspettative da verificare.

La struttura sarà la seguente:

```
describe('Google', () => {
  beforeAll(async () => {
    await page.goto('https://google.com');
  });

  it('should be titled "Google"', async () => {
    await expect(page.title()).resolves.toMatch('Google');
  });
});
```

Jest può essere installato singolarmente con il comando:

```
$ npm install jest --save-dev
```

Guida completa di Jest

Oppure può essere installato direttamente il pacchetto *jest-puppeteer*:

```
$ npm install jest-puppeteer --save-dev
```

Questa pacchetto contiene già le dipendenze di Jest e Puppeteer, perciò non sarà necessario installare quest'ultimo a parte.

Guida completa di jest-puppeteer

Configurazione di Jest

Guida completa alla configurazione di Jest

E' necessario creare il file `jest.config.js` contenente:

```
const config = {
  verbose: true,
};

module.exports = config;
```

Installazione di Puppeteer

Guida completa

Eseguire il comando:

```
$ npm install puppeteer --save-dev
```

Se si intende eseguire i test su versioni datate di Chromium, installare una versione di Puppeteer compatibile con la versione del browser utilizzata, come detto in precedenza.

```
$ npm install puppeteer@1.17.0 --save-dev
```

Installazione di jest-html-reporter

Guida completa

Eseguire il comando:

```
$ npm install jest-html-reporter --save-dev
```

Configurare Jest per elaborare i risultati del test aggiungendo la seguente voce al file di configurazione (jest.config.js):

```
"reporters": [
    "default",
    ["./node_modules/jest-html-reporter", {
        "pageTitle": "Test Report"
    }]
]
```

Altre configurazioni, come il path dove salvare il report, sono disponibili al link della guida completa.

Connessione al browser

Il framework Puppeteer dispone del metodo *Puppeteer.connect()* (documentazione), utilizzabile in questo modo:

```
browser = await puppeteer.connect({
  browserWSEndpoint: ...,
  defaultViewport: { width: 1024, height: 1280 }
});
```

Al parametro `browserWSEndpoint` va impostato l'indirizzo websocket del browser, recuperato come detto in precedenza. Il parametro `defaultViewport` serve a impostare la risoluzione del browser (di default è 800x600), in questo caso è necessario impostare 1024x1280, ovvero la risoluzione del display esterno della TVM.

Una volta avvenuta la connessione possiamo recuperare la lista di pagine aperte nel browser e specificare quella su cui vogliamo eseguire i test. In questo caso dobbiamo specificare la pagina 0, essendoci una sola pagina aperta nel browser, quella dell'applicazione.

```
const pages = await browser.pages();
page = pages[0];
```

A questo punto possiamo sfruttare i metodi di Puppeteer per interagire con l'applicazione:

```
await page.click('#app > div:nth-child(3) > div.sidebar > button');
```

Esecuzione dei test

> N.B.: ogni test per essere considerato tale, e quindi eseguito, deve contenere nel nome l'estensione `.test`, oltre a `.js`. Ad esempio: `home.test.js` o `file-upload.test.js`.

Esecuzione di tutte le suites:

```
$ npx jest --runInBand
```

> Utilizzare il parametro --runInBand per eseguire le suites in modo sequenziale. Di defualt vengono eseguite in parallelo, e ciò può causare problemi.

Esecuzione di un singolo test:

```
$ npm jest test_name.test.js
```