Vol.2021-GN-112 No.15
Vol.2021-CDS-30 No.15
Vol.2021-DCC-27 No.15
2021/1/25

Webアプリケーションの入力フォーム作成を支援する自動実行テストの提案

玉村 庄汰[1,a)]　荻原 剛志[2,b)]

概要：Web アプリケーション開発において，テストの自動化・効率化が重要視されるようになっている．一方，フロントエンド開発では開発サーバによってアプリケーションの動作を確認するスタイルが主流である．本稿では開発サーバを用いたフロントエンド開発において, 動作テストの作業効率向上を目的としたブラウザ操作の自動実行手法を提案する．提案手法を用いることで，開発者はスクリプトを 1 つ実行するだけでフォーム入力作業を自動化でき，従来は手作業で行っていたブラウザ操作の手間を省くことが可能になる．我々はこの手法に基づいた試作システムを作成し，提案手法の効果を確認した．

An automated testing method for input forms of Web applications

Tamamura Shota[1,a)]　Ogihara Takeshi[2,b)]

1. はじめに

Web アプリケーション開発ではソフトウェアのリリースサイクルの短縮が重要視されており，特にユーザの操作に関わるフロントエンド開発におけるテストの効率化が注目されている．現在，フロントエンド開発は**図 1** に示すように，仮想的な開発サーバをローカル環境に立ち上げ，そのサーバに開発中のアプリケーションを自動的に配置する仕組みを利用することが多い．開発者が開発中のアプリケーションのソースコードを変更すると自動的にその変更部分のビルドが開発サーバで行われ，更新された部分を開発者がブラウザでチェックする．

仮想的な開発サーバをローカル環境に立ち上げ，アプリケーションを自動配置するソフトウェアとして Docker[1] などがよく利用されているほか，コードエディタとして現在主流となっている Visual Studio Code にも，簡易的な開発サーバを立ち上げてアプリケーションをビルドする機能が存在している．

[1] 京都産業大学先端情報学研究科
Kyoto Sangyo University, Kyoto, 603–8555, Japan
[2] 京都産業大学情報理工学部
Kyoto Sangyo University, Kyoto, 603–8555, Japan
[a)] i1986197@cc.kyoto-su.ac.jp
[b)] ogihara@cc.kyoto-su.ac.jp

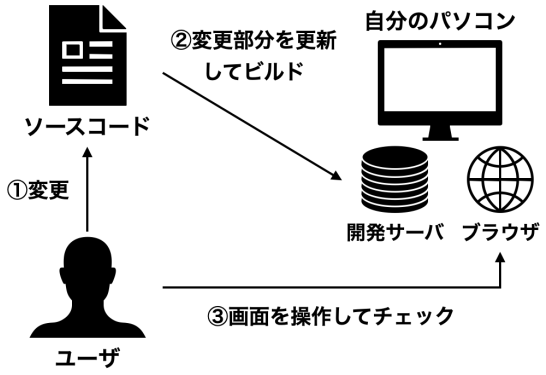

図 1　フロントエンド開発の開発スタイル

このようにフロントエンド開発では，開発サーバを立ち上げてアプリケーションをビルド，配置する機能を備えたフレームワークを用いるのが主流となっている．仮想的な開発サーバをローカル環境に立ち上げることで，ブラウザでアプリケーションを動かしながら開発作業を進めることが可能となり，開発効率の向上が実現できる．

一方，このような開発環境においても，ソースコードの変更のたびにブラウザの画面を手作業で操作して動作をチェックしなければならず，開発作業を低下させる原因となっている．特に入力フォームのチェックでは，カーソルを入力フォームにフォーカスしてから入力する必要がある

Vol.2021-GN-112 No.15
Vol.2021-CDS-30 No.15
Vol.2021-DCC-27 No.15
2021/1/25

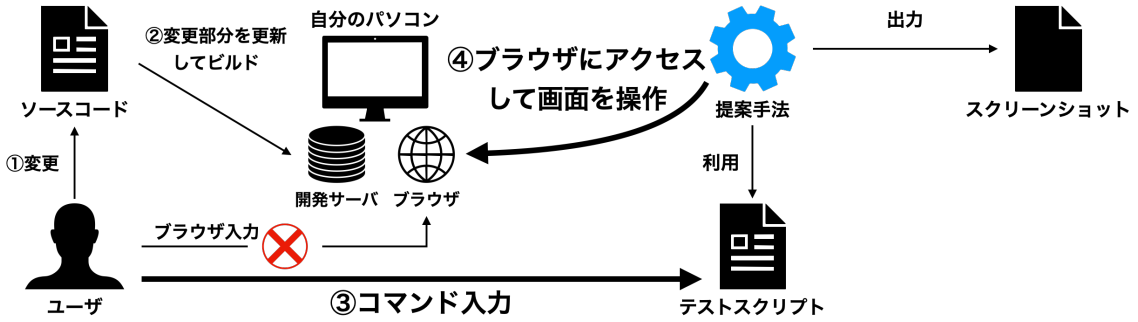

図 2 提案手法を用いた開発とテストの概念

ため，余計に時間がかかってしまう．

そこで本稿では，ブラウザ操作を自動化するライブラリを利用した入力フォームに特化した自動入力ツールを実装することで，入力フォームにおける動作チェックの自動化を行う手法を提案する．この手法を用いることで，開発段階で動作チェックを行う際の作業効率を向上させることができる．

以下，本稿ではこの手法の概要を説明し，試作システムの動作についても述べる．

2. 研究の背景

2.1 フロントエンド開発のフレームワーク

ここで，Web アプリケーションのフロントエンド開発で広く利用されているフレームワークと，テスト環境について述べる．

2.1.1 React

React[2] はシングルページアプリケーション (SPA) 開発のベースとなるユーザインタフェース構築の Javascript ライブラリである．アプリケーションの状態管理のフレームワークである Redux[3] と組み合わせて開発することが多い．React を用いたアプリケーションでは，webpack や parcel といったモジュールバンドラを用いることで開発サーバをローカル環境に立ち上げて，アプリケーションをビルド，配置することができる．

React アプリケーションには jest や enzyme といった単体テストを行うテスティングフレームワークが充実している．しかしこれらのフレームワークは，利用するまでの学習コストが膨大であり，テストコード作成に時間がかかってしまう．

2.1.2 Ruby on Rails

Ruby on Rails[4] は Ruby 言語で書かれた Web アプリケーションフレームワークであり，開発サーバをローカル環境に立ち上げ, アプリケーションをビルド，配置する機能を内包している．Ruby on Rails には RSpec といったテスティングフレームワークが存在するが，こちらもまた単体テストが中心である．利用にあたっては学習コストが膨大であり，テストコード作成に多くの時間がかかる．

2.2 関連研究

Wu ら [5] は，Web システムのテストにおける文字入力の手間を削減することを主な目的としたシステムを提案している．この手法では，与えられた Web システムのページを調査し，テキスト入力が必要なページを発見した場合に自動的にデータを入力してテスト実行を行う．この際，テキスト入力フォームの多くが個人情報の入力を求めるものであることに注目し，用意されたデータバンクから個人情報の例を入力するようになっている点が特徴である．

ただし，入力インタフェースに限定したテストではなく，Web システムの動作の確認を目的としているため，開発する Web システムで利用するデータをあらかじめ用意した上で，入力に対応した動作を検証するためのスクリプト（アクションルール）をあらかじめ記述しておかなければならない．

3. 提案手法

3.1 提案手法の概要

本手法は，Web アプリケーションにおけるテキスト入力フォームの動作チェックを自動的に行うことを目的とする．本手法を用いることによる新たな開発スタイルの仕組みを**図 2** に示す．

前述のように（図 1），通常の開発ではソースコードの変更後，開発サーバにビルドされたアプリケーションをブラウザで操作して動作のチェックを行う必要がある．提案手法では，入力フォームにおける「フォーム入力→ submit ボタン押下」という操作をスクリプトで自動実行し，結果をスクリーンショットによって確認することができるツールを作成する．これによって，開発者はブラウザを手作業で操作する必要がなくなり，スクリプトを 1 つ実行するだけで，入力フォームの動作確認が可能となる．

提案手法では，学習コストとテストコード作成コストの問題を解決するために，コマンド入力のみでフォーム入力作業を自動化することに焦点を置いた．図 1 のような開発スタイルの場合，開発の段階から頻繁に行うブラウザ画面の操作を自動化し，作業の時間を大幅に短縮する効果が期待される．

Vol.2021-GN-112 No.15
Vol.2021-CDS-30 No.15
Vol.2021-DCC-27 No.15
2021/1/25

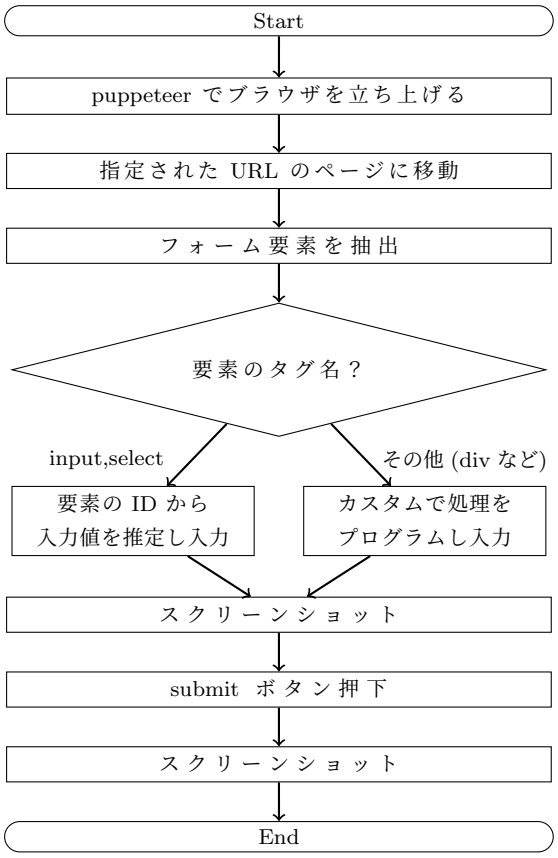

図 3 テストスクリプトの動作全体のフロー

なお本手法では，最終的にブラウザに HTML 形式のデータによって表示される Web アプリケーションを対象としている．さらに，テキスト入力のフォームは input 要素であると仮定している．これら以外のモバイルアプリケーションなどは適用範囲外であり，HTML で表示されない特殊なアプリケーションや，Web アプリケーションであっても入力フォームが input 要素でないものには適用できない．

3.2 提案手法の詳細

以下では，提案手法がツールを用いてどのようにブラウザにアクセスして画面を操作しているかを説明する．

3.2.1 テストスクリプトの詳細と全体のフロー

テストスクリプトは Javascript によって，Node.js[6] を用いて実装している．ソースファイルを変更し（図 2 の①），ビルド（②）を行ってから，コマンドを入力（③）してこのスクリプトを動作させるだけで，入力フォームのテストが実行できる．テストスクリプトがブラウザの自動操作を行う動作全体のフローを**図 3** に示す．入力フォームの入力後と submit ボタン押下後にスクリーンショットを保存するようになっており，これらを確認することで動作のチェックができる．

3.2.2 入力フォームに対する ID の指定

提案手法を使用するにあたって注意すべき部分がある．提案手法では入力フォームに対して指定された ID を手がかりに入力値の自動推定を行う．このため，開発者は HTML の入力フォームに該当する要素 (input など) に，あらかじめ決められた ID を指定しておく必要がある．テストスクリプトは，入力フォームの ID が既知の ID と一致した場合に，その ID に従った適切な入力値を推定して入力する．

ただし，どのような入力の種類を想定し，どのような ID を用いるのか，さらにテストにおいてどのような入力を適用するのかは，プロジェクトごとに自由に設定可能である．試作システムにおいては，既知の ID とその ID によって推測される入力のタイプを**表 1** のように定義して用いた．

```
import puppeteer from "puppeteer"

(async () => {
  const browser = await puppeteer.launch()
  const page = await browser.newPage()
})()
```

図 4 puppeteer でブラウザを立ち上げるサンプルコード

```
// url の Web ページに遷移
page.goto("https://sample.com", {
  waitUntil: "networkidle2",
  timeout: 5000
})

// Web ページのスクリーンショットを保存
page.screenshot({
  path: "path/to/sample.png",
  fullPage: true
})

// id="sample"の要素をクリック
page.click("#sample")
```

図 5 puppeteer による Web ページ操作のサンプルコード

3.2.3 ブラウザ操作ライブラリ

今回ブラウザ操作のツールとして用いたのは Puppeteer[7] という Javascript ライブラリである．ブラウザ操作ができるツールとしては，Selenium Webdriver[8] がよく知られており，テストの自動化に使われる例も多いが，Puppeteer はヘッドレスブラウザを使うことができるため，Selenium Webdriver より高速に動作させることができる．一方，Puppeteer には Chrome のブラウザしか操作できないという制約があり，注意が必要である．

3.3 自動化スクリプトの例

図 3 のフローにおける各部分は，Puppeteer で提供されている様々なクラスのメソッドを用いることで実現が容易に可能である．以下にいくつかの例を示す．

図 4 は puppeteer でブラウザを立ち上げる例である．ブラウザを立ち上げて得られる Page クラスのインスタンスが，ブラウザに表示される Web ページに対する操作を実

Vol.2021-GN-112 No.15
Vol.2021-CDS-30 No.15
Vol.2021-DCC-27 No.15
2021/1/25

表 1 ID の対応表

ID	入力タイプ	推定される入力 (形式)
lastName	苗字	佐藤，鈴木など
firstName	名前	大翔，蓮など
kanaLastName	フリガナ (苗字)	サトウ，スズキなど
kanaFirstName	フリガナ (名前)	タイショウ，レンなど
year	年 (生年月日)	1900〜2020 までの数値
month	月 (生年月日)	1〜12 までの数値
date	日 (生年月日)	1〜31 までの数値
email	メールアドレス	ローカル部@ドメイン (例:foo@example.com)
tel	電話番号	0???-??-????，0?-????-????などの形式 (?は 0〜9 の数値)
region	都道府県 (住所)	北海道，大阪府など
locality	市区町村以下 (住所)	漢字，ひらがな，カタカナと数字，ハイフン (-) は使用可能
password	パスワード	半角英数記号 8 文字以上
rePassword	パスワード確認	半角英数記号 8 文字以上

現させるメソッドを持っており，そのメソッドを使うことにより入力フォームの自動入力を可能にしている．

図 5 では，Web ページに対する操作を実現させるメソッドのいくつかの例を示した．これらのメソッドを使うことで，図 3 のフローに従った入力フォームの自動入力プログラムを 50 行ほどのコードで実現することができた．

Web アプリケーション開発のインタフェース部分では，見栄えの良いデザインにするための UI ライブラリが用いられていることも多い．そこで試作システムは，UI ライブラリの様々な入力フォーム，特にセレクトボックスの実装に対応できるように拡張可能な仕組みとした．図 3 のフローの中では，「カスタムで処理をプログラムし入力」という部分に相当する．

この部分のカスタマイズを実装して処理を行うためには，Puppeteer の学習コストと，プログラムの作成コストが生じるが，Puppeteer の学習コストは 2 章で述べたテスティングフレームワークの学習コストに比べるとかなり低い．カスタムで処理をプログラムした実際の例は 4 章で説明するが，このようなケースで作成するカスタムプログラムは 15 行程度であった．

4. 手法の適用例

提案手法の効果を確認するため，2 章で紹介した React を用いた簡単な Web アプリケーションを作成し，実験を行った．この Web アプリケーションの入力フォームを**図 6** に示す．今回の実験で用いる入力フォームには Material-UI[9] という UI ライブラリを用いた．作成したツールは様々な UI ライブラリを用いても自動入力を可能にできることを確認するために，ツールの拡張が可能になるように設計した．実験では，Material-UI を用いて実装したセレクトボックスに値を入力する部分をカスタムでプログラムした．その部分を**図 7** に示す．

また，作成した入力フォームは，表 1 の対応表に従って入力フォームに ID を指定している．試作 Web アプリケーションにおいて，入力フォーム部分の HTML の記述の簡易版を**図 8** に示す．

テスト方法は至って単純で，プロジェクトディレクトリ上で次のようなコマンドを入力するだけで良い．

```
node テストスクリプトのパス URL
```

するとスクリプトが図 3 に従って自動でフォームへの入力を行う．試作 Web アプリケーションにおいてテストスクリプトを実行して出力されたスクリーンショットを紹介する．入力が終わった後のフォームの様子をスクリーンショットしたものが**図 9** である．また，「登録」ボタンを押した際の様子をスクリーンショットしたものが**図 10** と**図 11** である．図 10 の場合は，バリデーションに失敗し，エラーが出ている場合であり，図 11 の場合は，バリデーションに成功し，次の画面に遷移している場合であるため，入力フォームは正しく動作していることが確認できる．

5. 議論

5.1 提案手法の評価

試作 Web アプリケーションによる評価実験を行った結果，コマンド 1 つで Web アプリケーションの入力フォームの自動入力とスクリーンショットによる結果の確認を自動で実行することに成功した．この実験では，仮想環境上で手動入力を行った場合に比べ，1 回のテストにかかる時間を 50%以上削減できた．この結果は，仮想環境での開発において頻繁に行われるブラウザ画面の操作を自動化し，作業時間の短縮に役立つものであると言えるだろう．

提案システムでは，典型的によく利用される input 要素に対しては自動で対応できるが，その他の多様な入力フォームには必ずしも対応ができない．したがって，その部分に関してはツールのカスタマイズが可能な設計としている．結果として，コスト自体は大きいものではないが，

Vol.2021-GN-112 No.15
Vol.2021-CDS-30 No.15
Vol.2021-DCC-27 No.15
2021/1/25

図 6 実験に用いる入力フォームのサンプル

```
if (tagName === "DIV" && id !== "root") {
  await this.page.click(`#${id}`);
  const item = await this.page.$("div[role='presentation']");
  if (item !== undefined) {
    const count = await this.page.$$eval(
      "div[role='presentation'] li",
      (elements) => elements.length
    );
    await this.page.click(
      `div[role='presentation'] li:nth-child(${
        Math.floor(Math.random() * count) + 1
      })`
    );
  }
}
```

図 7 カスタムでプログラムした部分

```
<div class="MuiCardContent-root">
  <div style="display: flex;">
    <p class="MuiTypography-root MuiTypography-body1">名前</p>
    <input id="lastName" type="text">
    <input id="firstName" type="text">
  </div>
  <div style="display: flex;">
    <p class="MuiTypography-root MuiTypography-body1">フリガナ</p>
    <input id="kanaLastName" type="text">
    <input id="kanaFirstName" type="text">
  </div>
  <div style="display: flex;">
    <p class="MuiTypography-root MuiTypography-body1">生年月日</p>
    <div role="button" aria-haspopup="listbox" id="year">
    <p class="MuiTypography-root MuiTypography-body1">年</p>
    <input id="month" type="text">
    <p class="MuiTypography-root MuiTypography-body1">月</p>
    <input id="date" type="text">
    <p class="MuiTypography-root MuiTypography-body1">日</p>
  </div>
  <!-- 以後省略 -->
</div>
```

図 8 入力フォーム部分の HTML の記述 (簡易版)

カスタマイズ方法に関する学習コスト，テストスクリプトの作成コストが必要となってしまっている．

5.2 入力値のバリエーションへの対応

提案手法の改良点として，入力値の組み合わせのバリエーションを組み込むことがある．入力値のバリエーションというのは，具体例として「ふりがなのひらがなとカタカナ」「メールアドレスと電話番号は片方で良いのか両方必須か」「パスワードとパスワード確認の一致と不一致」といったものがあり，こういったものの組み合わせの様々なパターンについて検証できるように，さらには必ずバリデーションに成功するようなパターンを指定することができるように改良が必要である．

5.3 先行研究との比較

すでに述べたように，Wu らの研究 [5] は，Web アプリケーションの入力フォームからデータの形式を推定し，あらかじめ用意した個人情報のデータバンクから適切なデータを選択して入力することでテストを自動的に行おうとするものである．入力フォームに対して自動的に文字列を入力してテストを実行する点で，我々の提案手法と共通する部分が多い．

ただし，Wu らのシステムは Web アプリケーション全体の動作のチェックを目的としており，そのために入力フォームに対する自動入力が適切に機能したかどうかを調べるアクションルールを記述する必要がある．このルールは，Web アプリケーションのロジックを反映したものになるため，実質的には，動作を確認するためのテストスクリ

Vol.2021-GN-112 No.15
Vol.2021-CDS-30 No.15
Vol.2021-DCC-27 No.15
2021/1/25

図 9 入力が終わった後のフォームのスクリーンショット

図 10 「登録」ボタンを押した際のスクリーンショット (バリデーション失敗)

図 11 「登録」ボタンを押した際のスクリーンショット (バリデーション成功)

プトを入力フォームに関連づけて記述するのと同じことになると考えられる．テストを実行するためのコストは低いとは言えないであろう．

手法は似ているものの，開発作業中に，低いコストで容易にテストを行うことを目的としている我々の提案とは異なっている．

5.4 E2E テストにおける提案手法の利用と関連研究との比較

Web アプリケーションの End-to-End(E2E) テストでの提案手法の利用の可能性について検討する．

E2E テストの自動化を行う関連研究を紹介する．青井ら [10] は，Web アプリケーションの頻繁な仕様変更に対応可能なページオブジェクトデザインパターンを利用した保守性の高い内部 DSL および，内部 DSL に基づくテストコードを自動生成するテスティングフレームワーク DePoT を開発した．DePoT を用いることで，可読性の高いテストコードのテンプレートが自動生成でき，テストの理解・作成・修正に必要な時間を削減できる．切貫ら [11] は End-to-End テストのログを用いて，ページオブジェクトパターンを採用した有用なテストスクリプトを自動生成した．切貫らの提案手法では，手作業によるテストを一度行うだけで，以降のテストはほとんどの機能が自動生成されたスクリプトによって実行が可能となった．

しかしこれらの手法では，E2E テストを容易に自動化できるわけではない．DePoT では，具体的なテストケースの内容を生成されたテンプレートに追記するために，詳細なシナリオの作成を行う必要がある．また，破壊的な変更が起きた際は，もう一度シナリオの設計を行う必要があるだろう．また，DePoT は React などを用いたシングルページアプリケーション (SPA) に関しては適用対象外である．切貫らの手法では，手作業によるテストを一度行えばよいとされているが，そのテストはすべての機能を網羅できるように時間をかけて探索的に行ったり，あるいは仕様を把握してテストケースを作成した上で実行しており，労力と時間がかかってしまっている．

一方，我々の提案手法においても，詳細なシナリオ (テストケース) の作成と破壊的な変更が起きた際のシナリオの再設計が必要になることには変わりない．しかし提案手法は設計したシナリオに従って操作する点においては有効となりうる．今回は入力フォームの入力操作の自動化を行ったが，単純なクリック操作などを組み込むことでシナリオ通りの操作を行うことが可能であろう．そのためには，提案手法の改良と操作の順番などを手作業でテストスクリプトに記述する必要がある．そして提案手法ではスクリプトで実際にブラウザ操作を行うため，次節に挙げる問題を解決する可能性がありうる．

5.5 ユーザの想定外の操作のテストの問題

Web アプリケーションの End-to-End 自動テストの問題として，ユーザの想定外の操作のテストの問題がある．例えば，ボタンを連打した場合にバグが発生するアプリケーションがあったとする．このとき自動化したテストをパスしただけではこのバグは見逃されてしまう．

ブラウザ操作をプログラミングで行う場合，そもそもブ

ラウザ操作をプログラミングで行えるようにするツールが，ありきたりな操作のみを実装し提供しているため，なかなかユーザの想定外の操作を実現することが難しくなる．

ユーザの想定外の操作のテストのため，手動でテストを行ってしまっては，自動化の意味が薄れてしまうと考える．提案手法でも，関連研究でもこの問題については解決されていないため，Web アプリケーションの自動テストにおいて今後この問題に向き合っていく必要があるだろう．

6. まとめ

本稿では，ブラウザ操作を自動化するライブラリを利用し，コマンド 1 つで入力フォームの入力操作を行う手法を提案した．また試作システムを作成し提案手法を用いて自動入力の動作確認を行ったところ，入力フォームの自動入力に成功し，手動入力によるテストよりも短時間で動作の確認を行うことができた．今後は，様々な入力フォームへの対応方法，E2E テストへの本手法の応用の可能性について検討を進めてゆく必要がある．

参考文献

[1] Docker Inc.: Empowering App Development for Developers — Docker，入手先 ⟨https://www.docker.io⟩, 2020.
[2] Facebook Inc.: React ーユーザインタフェース構築のための JavaScript ライブラリ, 入手先 ⟨https://ja.reactjs.org⟩, 2020.
[3] D. Abramov et al.: Redux ー A Predictable State Container for JS Apps，入手先 ⟨https://redux.js.org⟩, 2020.
[4] D. H. Hansson et al.: Ruby on Rails — A web-application framework that includes everything needed to create database-backed web applications according to the Model-View-Controller (MVC) pattern.，入手先 ⟨https://rubyonrails.org⟩, 2020.
[5] C. Wu, F. Wang, M. Weng and J. Lin: Automated testing of web applications with text input, 2015 IEEE Inter'l Conf. Progress in Informatics and Computing (PIC), pp. 343-347, 2015.
[6] R. Dahl et al.: Node.js，入手先 ⟨https://nodejs.org/ja⟩.
[7] Puppeteer v5.5.0，入手先 ⟨https://pptr.dev⟩, 2020.
[8] J. Huggins et al.: SeleniumHQ Browser Automation，入手先 ⟨http://www.seleniumhq.org⟩.
[9] Material-UI: Material-UI: A popular React UI framework，入手先 ⟨https://material-ui.com⟩.
[10] 青井 翔平，坂本 一憲，鷲崎 弘宜，深澤 良彰: DePoT: Web アプリケーションテストにおけるテストコード自動生成テスティングフレームワーク, 情処論, 56, 3, 1–12, 2015.
[11] 切貫 弘之，丹野 治門: 手動テストのログを用いた有用な End-to-End テストスクリプトの自動生成, ソフトウェアエンジニアリングシンポジウム, 2020.