BAB 2
DASAR TEORI DAN TINJAUAN PUSTAKA

Pada bab 2 akan dibahas tentang dasar teori dan tinjauan pustaka yang digunakan dalam pembuatan Proyek Akhir ini.

2.1 Dasar Teori

Dasar teori berisi tentang konsep atau pengetahuan, metode, dan tools yang digunakan untuk mendukung penyelesaian Proyek Akhir. salah satu metode yang digunakan adalah automation testing menggunakan *framework Cypress*, selain itu juga terdapat beberapa *tools* yang digunakan untuk menjalan Automation testing.

2.1.1. Quality Assurance

Quality Assurance, atau sering disebut *QA*, adalah individu yang bertanggung jawab atas perencanaan jaminan kualitas, identifikasi kesalahan, penyimpanan catatan, analisis, dan pelaporan. Jaminan kualitas perangkat lunak merupakan kegiatan yang melindungi seluruh proses pengembangan perangkat lunak. Tugas utama seorang *QA Tester* meliputi pengujian perangkat atau emulator, pembuatan alur pengujian, pembuatan laporan hasil pengujian, dan terkadang juga mencakup pembuatan program pengujian otomatis. Selain itu, *QA Tester* memberikan masukan terhadap aplikasi yang diuji dan berkomunikasi dengan berbagai pihak yang berkepentingan, seperti pengembang *UI/UX, Back-End*, atau Manager (PM) (Zakiyah, 2018).

Dalam pengembangan aplikasi manajemen testing menggunakan 7 principles of testing atau 7 hal prinsip dasar tentang testing yang tujuanya untuk mengoptimalkan pada saat melakukan (Mirza, 2020). berikut merupakan 7 prinsip tersebut.

1. ***Testing shows presence of defects*** atau pengujian menunjukan cacat dalam melakukan pengujian perangkat lunak proses yang dilakukan sebenarnya adalah untuk mencari cacat pada sebuah perangkat lunak yang

dikembangkan, yang dimana tester harus membuat sebuah perangkat lunak itu menjadi gagal.

2. ***Early testing*** atau pengujian dilakukan lebih awal pengujian yang dimulai sedingin mungkin dalam pengembangan perangkat lunak, sehingga cacat bisa ditemukan oleh tester dalam testing yang nantinya tester bisa merencanakan testing yang akan dibuat itu seperti apa dan ketika menemukan sebuah cacat/bug nantinya tester bisa memberi saran kepada developer
3. ***Exhaustive testing is not possible*** atau mustahil melakukan pengujian secara menyeluruh melakukan pengujian perangkat lunak secara keseluruhan itu sangat tidak mungkin, tidak perlu melakukan testing secara menyeluruh melainkan bisa juga tester set ekspektasi dan berasumsi membuat skenario batas atas dan batas bawah sebuah perangkat lunak dan tidak perlu mengetes semuanya melainkan menggunakan asumsi bahwa.
4. ***Testing is context dependent*** atau pengujian bergantung pada konteks pengujian bergantung pada konteks yang pada dasarnya berarti bahwa cara menguji sebuah situs akan berbeda dari cara menguji aplikasi komersial setiap test case yang dibuat tergantung konteksnya seperti contoh semisal test case A tentang login sedangkan test case B tentang logout itu hal yang berbeda dan ekspektasinya juga berbeda baru bisa melakukan testing.
5. ***Defect clustering*** atau pengelompokan cacat/bug pengelompokan cacat sebuah perangkat lunak pada proses pengujian lebih diutamakan melakukan pengujian pada modul atau fungsional program yang kecil, biasanya semakin kecil modulnya maka cacat/bug yang akan ditemukan akan semakin banyak
6. ***Pesticide paradox*** atau paradox pestisida proses pengujian yang dilakukan secara berulang pada akhirnya tidak akan menemukan cacat pada perangkat lunak. tester tidak bisa begitu saja bergantung pada teknik pengujian yang ada maka harus terus menerus memperbaiki metode yang ada untuk membuat pengujian lebih efektif seperti melakukan kesalahan membiarkan test case tidak di-update dan menganggap remeh bahwa test case tersebut

selalu hijau dan tidak perlu melakukan update test case itu hal yang salah dan tidak boleh dilakukan.

7. ***Absence of error fallacy*** atau tidak ada bug merupakan kesalahan dalam pengujian tidak ditemukan adanya cacat/bug maka hal itu perlu dicurigai, bisa saja perangkat lunak yang 99% bebas dari cacat masih tidak dapat digunakan, hal ini bisa terjadi jika sistem diuji secara menyeluruh untuk persyaratan yang salah maka pengujian perangkat lunak tidak hanya menemukan cacat tetapi juga untuk memeriksa bahwa perangkat lunak memenuhi kebutuhan bisnis.

2.1.2 Pengujian *Otomatis*

Pengujian *otomatis* adalah sebuah pengujian yang dilakukan secara otomatis dengan menggunakan kode program tertentu (Arcuri, 2019). Kode tersebut dibuat kemudian dijalankan yang berguna untuk melakukan pengujian *end-to-end* dengan waktu yang singkat. Menurut pengujian yang telah dilakukan oleh Mahomed dan koleganya 29 pengujian yang telah mereka lakukan hasilnya memenuhi kriteria (Mahomed dkk., 2021). Dengan adanya hal ini, pengujian manual dapat tergantikan dengan pengujian secara otomatis dengan hasil yang sama dan waktu yang lebih singkat.

2.1.3 Pengujian *End-to-End*

Pengujian end-to-end yaitu urutan langkah/tindakan yang dilakukan pada aplikasi dari awal hingga akhir. Hal ini didukung dari penjelasan Taky tentang end-to-end testing, yaitu sebuah pengujian yang meng ecek aplikasi di semua sistem agar sesuai dengan yang diharapkan (Taky, 2021). Pada saat proses pembuatan *IndiCar* berlangsung, penjelasan dari Taky terkait pengujian *end-to-end* sangat diterapkan agar tidak membuat para *stakeholder* kecewa dengan aplikasi yang telah dibuat.

Menurut Badal dan Grünler, pengujian *end-to-end* dapat menguji aplikasi pada tingkat sistem yang bersesuaian dengan skenario penggunaan. Akan tetapi, mereka juga menyebutkan bahwa pengujian *end-to-end* ini lambat dan tidak dapat diandalkan. Tidak hanya itu, mereka juga mengatakan bahwa apabila terjadi

kegagalan dalam pengujian, pengujian *end-to-end* tidak dapat mengidentifikasi tersebut (Badal & Grünler, 2021)..

2.1.4 *Cypress*

Cypress adalah sebuah *framework end-to-end* testing *open source* yang berguna untuk melakukan pengujian secara *otomatis.* Dengan menggunakan *Cypress*, para penggunanya sangat diuntungkan dari segi waktu dan tenaga. Hal ini juga di disebutkan dalam skripsi Nguyen yang menyebutkan bahwa *Cypress* adalah sebuah *framework* terbaik untuk *developers* dalam *end-to-end* testing (Nguyen, 2022). Tidak hanya itu, penjelasan tentang Cypress ini juga diperkuat oleh Taky yang menjelaskan bahwa *Cypress* adalah sebuah perkakas *open source grati*s tanpa harus bayar sepeserpun yang berguna untuk menguji tampilan secara gratis yang memiliki visual dalam proses pengujiannya (Taky, 2021)

2.1.5 *Test Case*

Test case merupakan rancangan skenario tugas yang dilakukan oleh user untuk memverifikasi fitur dari perangkat lunak. Pengujian perangkat lunak dapat membantu pengembang aplikasi untuk menemukan dan menghapus kesalahan *(error)* maupun cacat *(defect)* sebelum perangkat lunak tersebut dirilis. Pengujian yang dilakukan pada perangkat lunak yang sudah dirilis bertujuan untuk menjaga mutu dari aplikasi tersebut. Hasil dari pengujian menggunakan *test case* akan dibandingkan dengan hasil yang telah ditetapkan pengembang aplikasi (Hasibuan & Dirgahayu, 2021). Jika terdapat perbedaan dari hasil pengujian dan hasil yang telah ditetapkan pengembang aplikasi, maka akan dilakukan perbaikan pada kode program aplikasi tersebut. Dalam pembuatan *test case* terdapat beberapa atribut yang dibutuhkan beserta keterangannya, yaitu:

1. Fitur: Merupakan nama fitur atau *fungsionalitas* yang akan diuji.
2. *Test Case ID*: Merupakan identitas dari fitur atau *fungsionalitas* yang akan diuji.
3. *Test case description:* Merupakan identitas dari fitur atau *fungsionalitas* yang akan diuji.
4. *Precondition:* Merupakan kondisi sebelum sistem dilakukan pengujian.

5. *Test steps:* Merupakan langkah-langkah yang akan dilakukan saat pengujian.
6. Test data: Merupakan data yang akan dimasukkan pada *test case* ini.
7. *Expected Result*: Merupakan hasil yang diharapkan dari fitur atau fungsionalitas pada program yang akan diuji.
8. *Actual Result*: Merupakan hasil sebenarnya yang terjadi pada fitur atau fungsionalitas pada program yang diuji.
9. Status: Merupakan status dari fitur yang diuji apakah berhasil *(pass)* atau gagal *(fail*). Jika hasil tes tidak sesuai dengan hasil yang diharapkan *(Expected Result)* maka status pada test tersebut akan diisi gagal *(fail),* begitu juga sebaliknya.

Setelah memahami dan menyusun atribut yang dibutuhkan, langkah selanjutnya adalah membuat model pengujian manual dengan *test case* dalam bentuk tabel sesuai dengan atribut yang telah ditetapkan sebelumnya.

2.1.6 *Npm*

NPM merupakan paket manager untuk *Node.js*, yang ditemukan pada tahun 2009 sebagai suatu proyek terbuka untuk membantu pemgembang *Javascript* saling berbagi kode paket modul. *NPM* juga merupakan sebuah kode perintah untuk memungkin pengembang menggunakan dan mempublikasi paket modul. (Rahmad Setya Budi 2017)

2.1.7 *Node js*

Node.js merupakan salah satu platform pengembang yang dapat digunakan untuk membuat aplikasi berbasis *Cloud.* Node.js dikembangkan dari *engine JavaScrip*t yang dibuat oleh *Google* untuk *browser Chrome* ditambah dengan *libuv* serta beberapa pustaka lainnya. *Node.js* menggunakan *JavaScri*pt sebagai bahasa pemrograman dan *event-driven, non-blocking I/O* (*asynchronous*) model yang membuatnya ringan dan efisien. *Node.js* memiliki fitur *built-in HTTP server librar*y yang menjadikannya mampu menjadi sebuah web server tanpa bantuan *software* lainnya seperti *Apache* dan *Nginx* (Rahmad Setya Budi 2017).

Pada dasarnya, *Node.js* adalah sebuah runtime *environment* dan *script library*. Sebuah *runtime environment* adalah sebuah *softwar*e yang berfungsi untuk mengeksekusi, menjalankan dan mengimplementasikan fungsi-fungsi serta cara kerja inti dari suatu bahasa pemrograman. Sedangkan *script library* adalah kumpulan, kompilasi atau bank data berisi skrip/kode-kode pemrograman

2.1.8 *Javascript*

JavaScript merupakan bahasa *scripting* yang sering dipakai sebagai pengembangan website. Saat ini *JavaScript* memiliki kelebihan yaitu tidak memerlukan *compiler* untuk menjalankannya seperti pada bahasa C dan C++. Kode JavaScript biasanya berjalan langsung di *browser* web dan ditulis dengan menggunakan *scripting* yang ringan dan cepat. *Javascript* dapat digunakan sebagai menyempurnakan tampilan pada sisi *client* agar website menjadi *interaktif* dan *user friendly*. Untuk penggunaan *JavaScript* akan disisipkan pada elemen head pada HTML (yatini 2014).

Tidak hanya digunakan pada *client-side* saja, saat ini *JavaScript* banyak digunakan untuk *meng-handle* pada sisi server atau *server-side* dengan

pengembangan aplikasi pada *Node.js* yang digunakan sebagai *run time*. Berbeda dengan *JavaScript* yang berjalan disisi *client, Node.js* dapat dijalankan dengan dukungan *V8 Engine* buatan *Google* dan *dependency* atau modul bawaan yang terintegrasi dengan *HTTP, security, filesystem*, dan modul penting lainnya (R. FAJRIN 2017)

2.1.9 *Visual Studio Code*

Visual Studio Code (VS Code) adalah sebuah editor kode sumber yang populer yang dikembangkan oleh *Microsof*t. Dikenal karena antarmuka yang bersih dan ringan, *VS Code* memberikan pengalaman pengembangan yang kuat dan *responsi*f. Dengan fitur-fitur seperti *linting, IntelliSense, integrasi G*it, dan *debugging* yang kuat, VS Code mendukung berbagai bahasa pemrograman dan kerangka kerja. Dengan ekstensi yang dapat diinstal oleh pengguna, VSCode dapat diadaptasi untuk memenuhi kebutuhan pengembangan yang spesifik. Sebagai perangkat lunak terbuka sumber, VS Code menawarkan kemungkinan berkontribusi dan mengembangkan ekstensi atau fitur baru. *Platform* ini dapat dijalankan di berbagai sistem operasi, termasuk *Windows*, *macO*S, dan *Linux.* Lebih lanjut, tutorial dan dokumentasi yang komprehensif mendukung pengguna dalam memanfaatkan sepenuhnya potensi dari *VSCode*, menjadikannya pilihan favorit bagi pengembang perangkat lunak di seluruh dunia. Kunjungi situs web resminya di https://code.visualstudio.com/ untuk memulai pengalaman pengembangan yang ditingkatkan.

2.1.10 *MVP*

MVP (*Minimum Viable Produc*t) adalah sebuah produk dengan fitur-fitur dasar yang dapat memberikan solusi kepada pengguna. Sebuah *MVP* mencakup fitur-fitur sederhana yang esensial untuk mengatasi Pain Point pengguna dan memiliki alur penggunaan paling dasar dan penting.

2.1.11 *User Flow*

*User Flo*w merupakan rangkaian langkah yang diikuti oleh pengguna, mulai dari saat pertama kali mereka menggunakan sistem hingga mencapai langkah terakhir dalam penggunaan sistem tersebut. Biasanya, *User Flow* ditampilkan dalam bentuk diagram alur (*flowchar*t) untuk mempermudah pemahaman setiap proses yang dijalani oleh pengguna selama menggunakan sistem (Sutanto, 2022)

2.2 Tinjauan Pustaka

Adapun beberapa penelitian sebelumnya yang dijadikan referensi untuk pembuatan Proyek Akhir ini antara lain :

Tabel 2.1 Tinjauan Pustaka

Peneliti	Judul Penelitian	Hasil
Bima Prakoso	Pemanfaatan *cypress* untuk untuk pengujian *end-to-end* (studi kasus pengembangan aplikasi *indicar*)	Hasil dari penelitian adalah metode *end-to-end* testing dapat dipermudah menggunakan *Cypress* karena dapat mempercepat proses pengujian. Adapun tahapan yang dilakukan selama pembuatan pengujian *end-to-end* menggunakan *Cypress* seperti pembuatan skenario pengujian, pemilihan perkakas pengujian, validasi skenario pengujian,
Yoga kosasih	Automation testing tool dalam pengujian aplikasi *the point of sale*	Berdasarkan eksperimen yang dilakukan dapat diambil kesimpulan bahwa kesimpulan tergantung dari kebutuhan pengujian yang dilakukan jika membutuhkan pengujian yang *repetitif* atau harus menguji banyak platform data yang besar sebaiknya itu dibuat *automation script* dan jika sifatnya cuma 1 kali testing atau dibutuhkan perasaan atau *eksperienc*e langsung biasanya sifatnya *emergency* sebaiknya manual juga bisa dipertimbangkan.
Guido Wira Raditya Baskoro Putro	Perbandingan *performa selenium* dan *cypress* dalam pengujian website bimbingan mahasiswa universitas atma jaya yogyakarta	Hasil dari penelitian ini adalah bahwa *Cypress* lebih cepat dalam menguji Situs Bimbingan Mahasiswa Universitas Atma Jaya. Perbedaan rata-rata waktu pengujian antara *Cypress* dan *Selenium* adalah sebesar 3.48 detik

Hartono rusli	Analisa perbandingan *black-box automated* testing dan manual testing pada aplikasi ACCMART	Hasil dari penelitian ini adalah Perbandingan yang didapatkan setelah melakukan kedua pengujian secara otomatis dan manual adalah bahwa pengujian otomatis cocok digunakan untuk pengujian yang sudah memiliki *spesifikasi* yang jelas, sementara pengujian manual cocok untuk digunakan pada aplikasi yang sebelumnya belum pernah diuji
(Neumeyer, 2022)	Penggunaan test case template sederhana untuk pengujian perangkat lunak menggunakan aplikasi *Microsoft Excel*	Atribut-atribut yang digunakan dalam test case template serta contoh test case menggunakan template menggunakan aplikasi *Microsoft Excel*