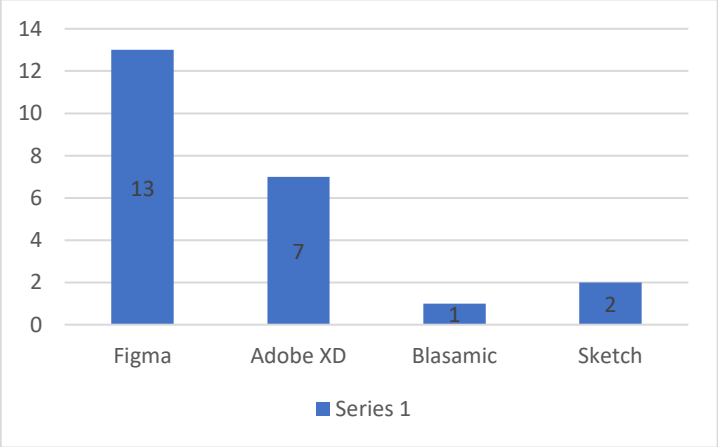
BAB II
LANDASAN TEORI

2.1 Figma

Figma merupakan alat desain berbasis web yang telah mendapatkan perhatian luas dalam pengembangan antarmuka pengguna (UI) dan pengalaman pengguna (UX) (Herniyanti et al., 2022). Salah satu keunggulan utama Figma mendukung kolaborasi *real-time*, yang memungkinkan desainer dan pengembang untuk bekerja secara bersamaan dalam satu proyek tanpa batasan geografis. Hal ini sangat penting dalam konteks pengembangan perangkat lunak, di mana komunikasi yang efektif antara tim desain dan pengembangan (Asnal et al., 2021). Penelitian sebelumnya menunjukkan bahwa penggunaan Figma dalam pelatihan desain UI /UX di institusi pendidikan menghasilkan pemahaman yang lebih baik tentang proses desain dan meningkatkan keterampilan peserta dalam menciptakan prototipe yang efektif (Pramudita et al., 2021).



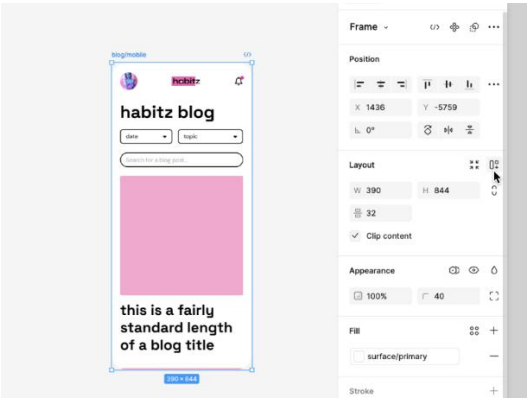
Gambar 2. 1 Popularitas Figma (G. M. De Oliveira et al., 2023)

Pada Gambar 2.1 menunjukan bahwa menurut (G. M. De Oliveira et al., 2023) setelah melakukan penelitian terkait voting di komunitas Ui/Ux. Figma aplikasi

yang sering kali di gunakan pada kalangan desainer Ui /Ux dibandingkan adobe XD yang tidak memiliki fitur mendukung kolaborasi dalam waktu nyata. Selain itu juga Figma sebagai platform yang mendukung pengembangan produk digital yang lebih baik, yang pada akhirnya berkontribusi pada peningkatan kualitas kode yang dihasilkan dalam proyek pengembangan perangkat lunak (Pramudita et al., 2021).

2.2 Auto Layout Figma

Auto Layout pada Figma merupakan Fungsi utama untuk mengatur elemen-elemen desain secara otomatis berdasarkan aturan yang ditetapkan, seperti jarak antar elemen, ukuran, dan orientasi (Helpfigma, 2020). Dengan menggunakan *Auto Layout*, desainer dapat dengan mudah membuat komponen yang terstuktur, yang secara otomatis menyesuaikan untuk memastikan konsistensi desain dan mempermudah proses generate desain menjadi kode, karena struktur desain yang dihasilkan menjadi lebih rapi dan sesuai standar pengembangan perangkat lunak (Jin et al., 2018). *Auto Layout* juga memungkinkan pengelompokan elemen yang logis, sehingga memudahkan dalam pengelolaan dan pemeliharaan desain (Wang et al., 2019).



Gambar 2. 2 Ilustrasi Auto Layout Figma (Helpfigma, 2020)

Gambar 2.2 mengambarkan cara auto layout bekerja pada sebuat desain Figma. Dalam proses generate desain menjadi kode, penggunaan *Auto Layout* sangat krusial karena struktur desain yang dihasilkan menjadi lebih rapi dan sesuai dengan standar pengembangan perangkat lunak (Zhang et al., 2021). Oleh karena itu, *Auto Layout* tidak hanya meningkatkan efisiensi dalam proses desain, tetapi juga berkontribusi pada pengembangan kode yang lebih berkualitas dan mudah dipelihara (Gu et al., 2019).

Selain itu, Auto Layout juga berperan penting dalam memastikan hasil generate dapat tampil secara responsif saat dibuka di browser. Struktur dan pengaturan layout yang tepat di Figma akan memengaruhi kemampuan elemen-elemen antarmuka untuk menyesuaikan tampilan pada berbagai ukuran layar. Oleh karena itu, penggunaan Auto Layout menjadi salah satu penentu utama dalam evaluasi visual dan fungsionalitas hasil generate yang dilakukan dalam penelitian ini.

2.3 Plugin Figma

Plugin Figma dalam konteks penelitian ini alat yang memungkinkan transisi desain visual ke kode untuk pengembangan perangkat lunak secara efisien. *Plugin* ini membantu mengonversi elemen desain menjadi kode yang siap digunakan, sehingga mengurangi proses pengkodean manual yang memakan waktu (Kashif et al., 2023). Dalam konteks penelitian ini, plugin yang digunakan difokuskan untuk menghasilkan kode yang terstruktur sesuai standar pengembangan, mendukung efisiensi kolaborasi antara desainer dan pengembang (Alao et al., 2022).

2.3.1. Dualite

Plugin Dualite merupakan alat yang dirancang untuk mengonversi desain dari Figma menjadi kode React JS yang terstruktur (Dualite.dev, 2023). *Dualite* berfokus pada pembuatan struktur file yang mengikuti praktik terbaik pengembangan perangkat lunak, seperti direktori *src* dan *components*. Dengan kemampuannya menghasilkan kode langsung dari elemen desain, Dualite mempermudah transisi dari desain ke implementasi, menghemat waktu, dan meminimlisir waktu pengembangan (Santoso, 2024).

Selain itu, struktur file yang dihasilkan oleh Dualite mempermudah developer dalam memahami alur proyek, terutama dalam tim yang bekerja secara kolaboratif. Hal ini sangat membantu dalam proses revisi atau pengembangan lanjutan (Booch et al., 2019; Perlak, 2019).

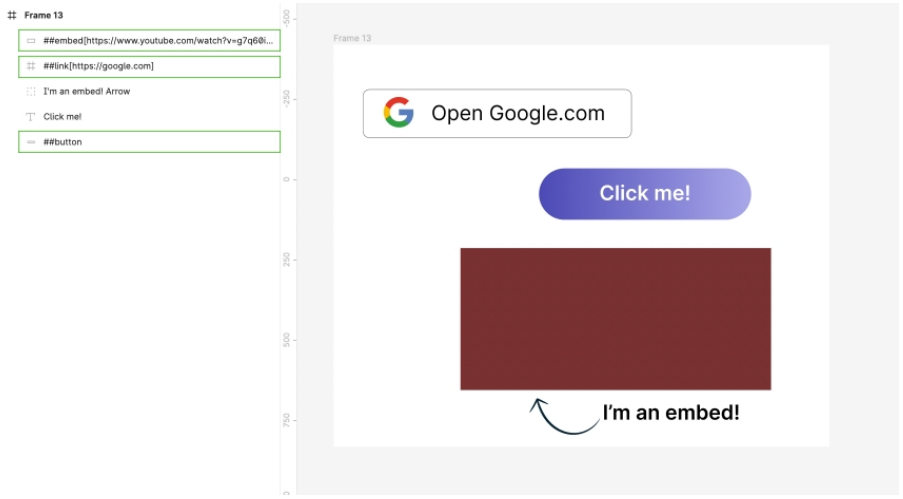
Gambar 2. 3 Interface Dualite (Dualite.dev, 2023)

Pada gambar 2.3 menunjukan Bagaimana penampilan *plugin Dualite* di muncul kan di Figma*, Plugin* ini sangat berguna untuk membuat antarmuka *web*

seperti *landing page*, meskipun memiliki batasan seperti tidak mendukung *animas* (Laine et al., 2021). Oleh karena itu, *Dualite* cocok untuk kebutuhan sederhana dengan tetap memperhatikan batasannya dalam generate code.

1. *Tagging*

Dalam proses konversi desain ke kode menggunakan plugin Dualite, beberapa fitur yang disediakan adalah tagging, yaitu enlemen-elemen tertentu pada desain figma secara otomatis di-konversi sebagai komponen-komponen dengan fungsi unik pada kode hasil generate (Dualite.dev, 2023). Tagging sendiri dilakukan dengan menambahkan tanda khusus pada nama elemen yang ada didalam figma sebelum proses generate kode dilakukan. Adapun dua tag yang digunakan dalam penelitian ini yaitu; *#link#* dan *#button#*.



Gambar 2. 4 Tagging (Dualite.dev, 2023)

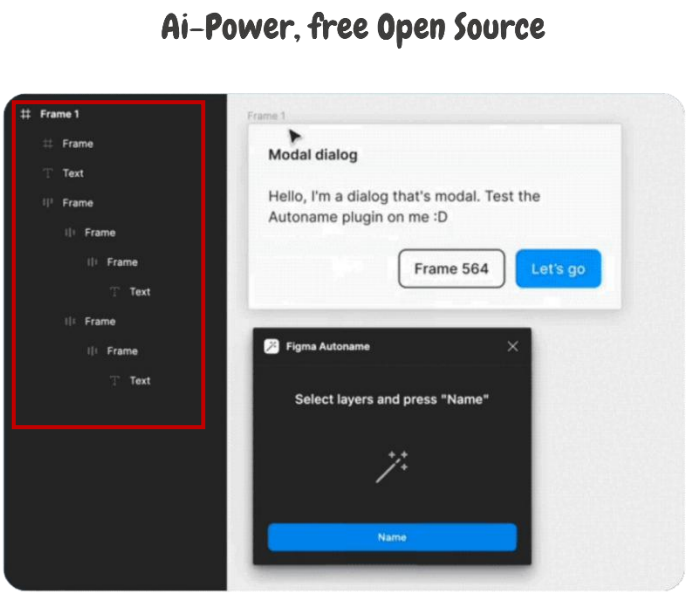
Pada gambar 2.4 Tagging menampilkan poroses tag dilakukan pada frame elemen figma. Tag *#link#* digunakan untuk menandai elemen sebagai hyperlink, sehingga saat kode dihasilkan, elemen tersebut secara otomatis dikonversi menjadi elemen <*a*> dengan atribut *href* yang sesuai. Hal ini mempermudah

pengembang dalam mengintegrasikan navigasi tanpa perlu melakukan perubahan manual pada kode hasil generate. Sementara itu, *tag #button#* digunakan untuk menandai elemen sebagai tombol interaktif *(<button>)*, sehingga komponen tersebut langsung berfungsi sebagai tombol tanpa perlu pengaturan tambahan.

Sebagai bagian dari proses konversi desain ke kode, hasil generate dari plugin Dualite juga perlu dievaluasi secara visual dan fungsional. Evaluasi ini bertujuan untuk memastikan bahwa seluruh elemen hasil konversi, seperti tombol, warna, layout, dan navigasi, telah bekerja dan tampil sesuai dengan desain awal di Figma. Oleh karena itu, dalam tahap akhir implementasi, dilakukan pengamatan manual terhadap hasil generate di browser untuk mengidentifikasi kesesuaian tampilan dan fungsi antarmuka.

2.3.2. Figma Auto Name

Plugin Figma *Auto Name* merupakan alat yang dirancang untuk meningkatkan efisiensi dalam pengelolaan nama frame dalam desain yang dibuat di Figma (Duprez, 2020). Dengan menggunakan *plugin* ini, nama-nama *frame* dapat diatur secara otomatis dan terstruktur, yang sangat membantu *developer* dalam memahami elemen-elemen desain saat mereka beralih ke pengkodean di editor seperti Visual Studio Code (VS Code). Penggunaan plugin ini memungkinkan desainer untuk fokus pada aspek kreatif dari desain, sementara developer dapat dengan mudah mengidentifikasi dan mengelola elemen-elemen tersebut berdasarkan nama yang telah diatur dengan baik.



Gambar 2. 5 Import from Figma (Hugo, 2024)

Pada gambar 2.4 menujukan setiap *frame* yang di *generate* dapat menjadikan penamaan otomatis pada suatu *frame* yang di pilih. Salah satu manfaat utama dari *plugin Auto Name* adalah kemampuannya untuk mengurangi kebingungan yang sering terjadi akibat penamaan elemen yang tidak konsisten. Ini sejalan menurut (Kashif et al., 2023). Dalam konteks kolaborasi tim, konsistensi dalam penamaan sangat penting untuk memastikan bahwa semua anggota tim dapat dengan mudah memahami dan mengakses elemen desain yang diperlukan.

2.4 Java Script

JavaScript dipilih dalam penelitian ini karena perannya yang krusial dalam pengembangan aplikasi *web*, terutama dalam mengonversi desain ke kode menggunakan plugin Figma seperti *Dualite*. *JavaScript* memungkinkan interaksi langsung dengan elemen-elemen dan mendukung pengembangan aplikasi *web*,

menjadikannya pilihan ideal untuk menghasilkan kode dari desain Figma (Viana et al., 2017).

Plugin Dualite menggunakan *JavaScript* untuk mengonversi desain menjadi komponen *React*, mempermudah pembuatan kode yang efisien dan mudah dipelihara (Auler et al., n.d.). Penelitian ini berfokus pada analisis kualitas kode yang dihasilkan, tanpa melibatkan pengujian pengguna atau elemen desain kompleks seperti *animasi* (Silva et al., 2017).

2.5 React JS

React JS adalah library *JavaScript* yang digunakan untuk membangun antarmuka pengguna dengan struktur yang efisien dan mudah dikelola. Dalam penelitian ini, React JS diterapkan oleh *plugin* Figma seperti *Dualite* untuk menghasilkan kode dari desain visual di Figma. Salah satu alasan utama pemilihan React JS yang memudahkan pengorganisasian kode dan meningkatkan efisiensi pengembangan (Kurniaji et al., 2023). Hal ini sangat relevan dengan tujuan penelitian ini yang berfokus pada analisis kualitas kode yang dihasilkan, di mana struktur yang jelas dan modular memungkinkan pengembangan yang lebih terorganisir dan mudah dipelihara (Kurniaji et al., 2023).

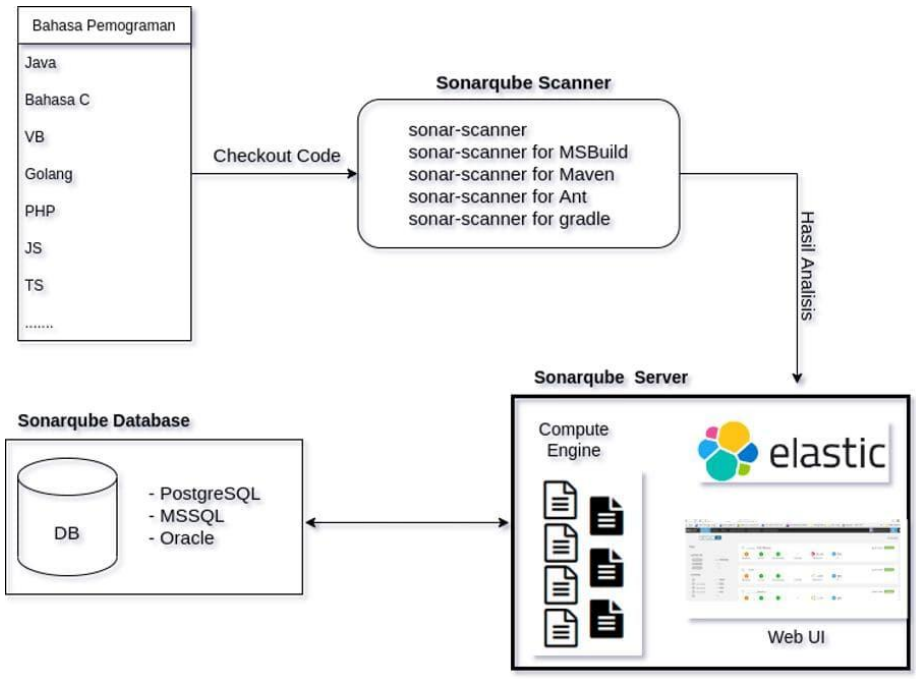
Dengan pendekatan berbasis komponen, React JS memungkinkan setiap bagian antarmuka pengguna dikembangkan secara terpisah dan digabungkan dalam proyek, mempermudah pemeliharaan kode jangka panjang. React JS juga menawarkan fitur-fitur seperti Virtual *DOM*, yang memungkinkan pembaruan antarmuka pengguna dengan lebih cepat dan efisien. Ini mengurangi beban pada browser dan meningkatkan pengalaman pengguna secara keseluruhan. Dengan

demikian, React JS tidak hanya memastikan bahwa kode yang dihasilkan x plugin Figma berfungsi dengan baik, tetapi juga meningkatkan kualitas dan keterkelolaan kode tersebut sesuai dengan fokus penelitian ini.

2.6 Sonarqube

Sonarqube adalah alat analisis kode statis yang berfungsi untuk meningkatkan kualitas kode dengan mendeteksi *bug* dan *code smells* (Trautsch et al., 2023; Xue et al., 2019). Pada konteks penelitian ini Sonarqube digunakan untuk menganalisis kode React JS yang dihasilkan dari proses konversi desain Figma.

Dengan memeriksa kode berdasarkan aturan pengkodean untuk berbagai bahasa, termasuk *JavaScript*, Sonarqube memberikan umpan balik cepat dan memberikan rekomendasi perbaikan.



Gambar 2. 6 Arsitektur Sonarqube (Febriana Rahmawati et al., 2023)

Pada gambar 2.5 memperlihatkan bagaimana cara Sonarqube menganalisis sebuah code yang ada penelitian ini menggunakan bahasa *Java Script* dan

Mengunakan data base dari *PostgreSQL*. SonarQube memiliki berbagai metrik yang digunakan untuk mengevaluasi kualitas kode, termasuk *deteksi bug, dan identifikasi code smells*, yang dijelaskan lebih lanjut dalam subbab berikut.

2.6.1. Bug

Kesalahan kode yang dapat menyebabkan aplikasi tidak berjalan sebagaimana mestinya disebut bug. Dalam konteks penelitian ini Bug SonarQube ditemukan dalam kode React JS yang dibuat oleh plugin Dualite dari desain Figma (Sitorus et al., 2025a) .

Table 2. 1 Bug Variabel Undefined

```
    function getMessage() {
        return message; // BUG: Variabel 'message' tidak
dideklarasikan sebelumnya
    }
    console.log(getMessage());   //   Akan   menghasilkan
error: ReferenceError: message is not defined
```

Pada Tabel 2.1 menunjukkan contoh bug *Variabel Undefined*, yaitu kesalahan dalam penggunaan variabel yang belum dideklarasikan sebelum digunakan. Bug ini sering terjadi dalam pengembangan perangkat lunak, terutama ketika kode dihasilkan secara otomatis oleh alat seperti plugin Dualite pada Figma.

2.6.2. Code Smells

Code smales adalah tanda bahwa ada masalah kode yang mungkin terjadi. Ini tidak selalu menyebabkan kesalahan tetapi dapat mempengaruhi keterbacaan, pemeliharaan, dan efisiensi (Fadhli et al., 2024). Dalam penelitian ini, SonarQube digunakan untuk mengidentifikasi Code Smells pada React JS yang dibuat oleh

plugin Dualite. Tujuannya adalah untuk menemukan pola yang tidak efektif atau tidak sesuai dengan standar pengembangan perangkat lunak yang ideal.

Table 2. 2 Duplicated Code

```
    import React from "react";

    const ButtonPrimary = () => {
      return   <button   className="btn   btn-primary">Click
Me</button>;
    };

    const ButtonSecondary = () => {
      return   <button   className="btn   btn-secondary">Click
Me</button>;
    };

    const App = () => {
      return (
        <div>
          <ButtonPrimary />
          <ButtonSecondary />
        </div>
      );
    };

export default App;
```

Pada Tabel 2.2 menunjukkan contoh Code Smells: *Duplicated Code*, yaitu kondisi di mana terdapat duplikasi kode yang tidak perlu dalam suatu program. Salah satu contoh *duplicated code* ketika terdapat dua komponen tombol yang memiliki struktur kode hampir sama, seperti *ButtonPrimary* dan *ButtonSecondary*. Keduanya memiliki fungsi yang mirip tetapi ditulis secara terpisah, sehingga mengakibatkan kode berulang dan tidak fleksibel.

Dalam konteks penelitian ini, code smells seperti duplicated code bisa muncul pada kode yang dihasilkan secara otomatis oleh plugin Dualite dalam Figma. Jika tidak diperbaiki, dapat mempengaruhi keterbacaan, pemeliharaan, dan efisiensi (Fadhli et al., 2024).

2.6.3. Vulnerability

Vulnerability merupakan celah atau kelemahan dalam kode program yang dapat dimanfaatkan oleh pihak tidak bertanggung jawab untuk mengakses, memodifikasi, atau merusak sistem tanpa izin (Anjum et al., 2020). Dalam konteks pengembangan perangkat lunak, keberadaan *Vulnerability* menjadi salah satu ancaman serius yang dapat berdampak pada keamanan, integritas, dan ketersediaan sistem (Sitorus et al., 2025).

Dalam penelitian ini, analisis *Vulnerability* dilakukan menggunakan SonarQube sebagai alat bantu pemeriksaan statis terhadap kualitas kode. SonarQube memiliki kemampuan untuk mendeteksi berbagai potensi *Vulnerability* berdasarkan aturan-aturan keamanan standar industri (Filus et al., 2021). eskipun dalam hasil analisis pada penelitian ini tidak ditemukan adanya kategori *Vulnerability*, namun proses pemeriksaan tetap dilakukan untuk memastikan bahwa kode yang dihasilkan memenuhi standar keamanan yang ditetapkan.

Penting untuk dicatat bahwa tidak ditemukannya *Vulnerability* dalam hasil analisis bukan berarti meniadakan risiko sepenuhnya, melainkan menunjukkan bahwa berdasarkan standar pemeriksaan SonarQube yang digunakan, tidak ada celah keamanan yang terdeteksi pada saat pengujian. Oleh karena itu, pemeriksaan terhadap *Vulnerability* tetap menjadi bagian integral dalam proses evaluasi kualitas kode, khususnya untuk menjaga keberlanjutan dan kepercayaan terhadap sistem perangkat lunak yang dikembangkan.

2.6.4. Kriteria Kode Layak

Evaluasi kelayakan kode dalam penelitian ini mengacu pada kriteria yang ditetapkan oleh SonarQube untuk menilai kualitas kode berdasarkan persentase masalah yang ditemukan dalam hasil analisis (Sonarqube, 2024a). Kriteria kelayakan kode yang digunakan dalam SonarQube dibagi menjadi empat kategori berdasarkan persentase jumlah masalah yang ditemukan, yaitu:

- Kategori A: Persentase masalah 0% - 5% → Kode dianggap layak.
- Kategori B: Persentase masalah 6% - 10% → Kode dianggap cukup layak.
- Kategori C: Persentase masalah 11% - 20% → Kode dianggap kurang layak.
- Kategori D: Persentase masalah 21% - 50% → Kode dianggap tidak layak.

Untuk menghitung persentase masalah, digunakan rumus berikut merujuk dari dokumentasi sonarqube (Sonarqube, 2024):

$$Rasio\ Masalah = \left(\frac{Jumlah\ masalah}{Jumlah\ baris\ kode}\right) x\ 100$$

Keterangan:

- Jumlah Masalah: Total masalah yang ditemukan (bugs, code smells, vulnerabilities).
- Jumlah Baris Kode: Total lines of code (LOC) yang dianalisis dalam proyek.

Kriteria-kriteria ini digunakan untuk menentukan sejauh mana kode memenuhi standar kualitas yang diinginkan dan untuk memberikan gambaran tentang keandalan dan kemudahan pemeliharaan kode.

2.7 Penelitian terkait

Table 2. 3 Penelitain Terkait

Penulis	Judul	Plugin	Fokus Penelitian
(F. Oliveira et al., 2020)	Prototypes of Very High Fidelity, Accessibility, and Reusability with Laum	LAUM (Figma plugin)	Fokus penelitian ini membahas LAUM, pustaka micro frontend yang meningkatkan kegunaan dan aksesibilitas. Melalui plugin Figma, LAUM memfasilitasi kolaborasi desainer-pengembang, menghasilkan kode HTML berkualitas tinggi, dan memastikan aksesibilitas bagi pengguna disabilitas.
(Kashif et al., 2023)	Automated Accessibility Evaluation of Mobile Applications on	Builder.io (Figma Plugin)	mengembangkan plugin Builder.io untuk evaluasi aksesibilitas otomatis pada desain aplikasi seluler. Plugin

Penulis	Judul	Plugin	Fokus Penelitian
	Initial Stages of Design and Development		ini menghasilkan laporan berdasarkan kriteria WCAG 2.1, membantu memastikan aksesibilitas sejak tahap awal pengembangan.
(F. Oliveira et al., 2020)	Development and evaluation of the plugin for Figma for Accessibility Documentation for Interfaces - DAI	DAI (Figma plugin)	Fokus Penelitian ini mengembangkan dan mengevaluasi plugin DAI untuk Figma, yang membantu mendokumentasikan aksesibilitas UI berdasarkan pedoman WCAG. Uji kegunaan melibatkan desainer dan pengembang untuk menilai efektivitas DAI dalam memfasilitasi komunikasi dan menciptakan UI yang lebih inklusif.

Penulis	Judul	Plugin	Fokus Penelitian
(Santoso, 2024)	Implementation Of UI /UX Concepts And Techniques In Web Layout Design With Figma	Weblify (Figma Plugin)	Membahas mengevaluasi penerapan prinsip UI /UX dalam desain web menggunakan Figma. Fokusnya pada proses desain hingga penggunaan plugin untuk menciptakan antarmuka yang konsisten, andal, dan harmonis, dengan peningkatan pada harmoni warna, font, dan ukuran yang sesuai sistem desain.
(Malnati, 2024)	Automatic generation of User Interface (UI) with the help of AI technologies	Auto HTML (Figma Plugin)	mengusulkan otomatisasi pembuatan UI dengan AI, menggunakan plugin Auto HTML Figma yang terhubung. Plugin

Penulis	Judul	Plugin	Fokus Penelitian
			ini memproses deskripsi proyek pengguna untuk menghasilkan desain UI relevan, mempercepat prototipe, dan mendukung kreativitas desainer.
(Ericsson et al., 2024)	Examining the Accessibility of AI-Generated Code by Generative Code Tools in Transition from Prototype to Development	Locofy (Figma Plugin)	Pada jurnal ini lebh membahas pada kemampuan alat generative code AI dalam menghasilkan kode sesuai standar aksesibilitas web. Studi ini juga mengusulkan kerangka kerja untuk mengintegrasikan alat tersebut ke proses pengembangan, meningkatkan efisiensi dari prototipe ke kode siap-pengembangan.

Penulis	Judul	Plugin	Fokus Penelitian
(Lappeenranta Lahti, 2024)	Harnessing Ai For Front-End Development: An Evaluation Of Design-To-Code Tools	Teleport HQ (Figma Plugin)	Penelitiaan ini membahas pada evaluasi kemampuan, keterbatasan, dan efektivitas alat AI design-to-code dalam menerjemahkan desain ke kode. Studi ini bertujuan memahami kegunaan alat tersebut dan dampaknya pada pengembangan website di masa depan.

Berbagai penelitian telah membahas penggunaan plugin Figma dalam proses generate kode. Misalnya, penelitian oleh Oliveira et al. (2020) menggunakan plugin LAUM untuk menghasilkan kode HTML yang mendukung aksesibilitas dan kolaborasi desain. Sementara itu, Kashif et al. (2023) dan De Oliveira et al. (2023) fokus pada plugin yang mengevaluasi aksesibilitas desain berbasis pedoman WCAG.

Penelitian Santoso (2024) mengevaluasi plugin Anima dalam meningkatkan konsistensi desain antarmuka, sedangkan Malnati (2024) dan Ericsson et al. (2024) mengangkat teknologi AI dalam generate UI secara otomatis melalui plugin seperti

Auto HTML. Terakhir, Lappeenranta Lahti (2024) menganalisis kemampuan alat design-to-code seperti Teleport HQ dalam menerjemahkan desain ke kode siap pakai.

Meskipun banyak penelitian membahas plugin Figma dan otomatisasi kode, belum ada studi yang secara khusus mengevaluasi kualitas kode hasil generate menggunakan plugin Figma melalui alat analisis seperti SonarQube. Penelitian ini bertujuan untuk mengisi celah tersebut dengan menganalisis kualitas kode hasil generate plugin Dualite berbasis React JS, serta membandingkannya melalui pendekatan eksperimen dua kondisi desain. Ringkasan perbandingan dapat dilihat pada Tabel 2.2.

Table 2. 4 Matrik Penelitian

Penulis	Plugin	Fitur				
		Bebas Aksess	Hasil Kode Terstruktur	Support React.Js	Support Auto Layout	Manajemen File Terstruktur
(Lappeenranta Lahti, 2024)	Teleport HQ	-	-	-	√	√
(Ericsson et al., 2024)	Locofy	-	-	√	√	-
(Kashif et al., 2023)	Builder.io	-	√	√	√	-

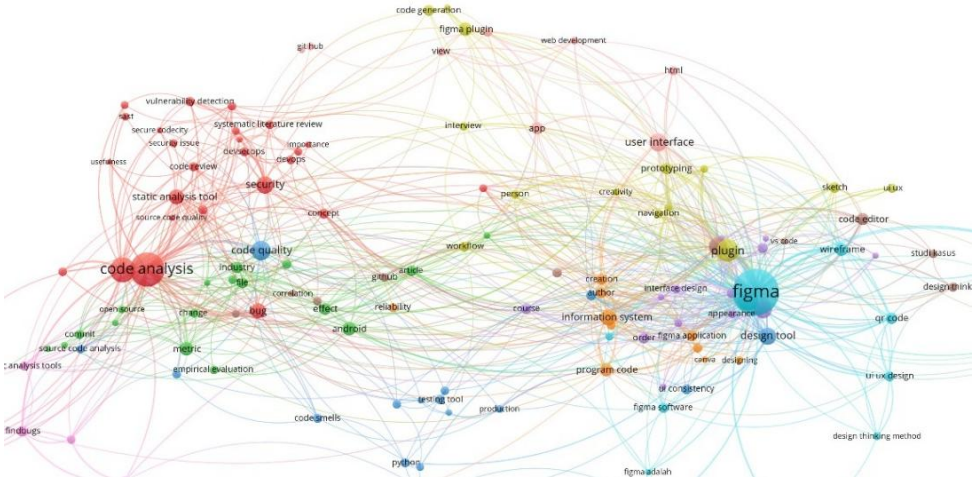
Penulis	Plugin	Fitur				
		Bebas Aksess	Hasil Kode Terstruktur	Support React.Js	Support Auto Layout	Manajemen File Terstruktur
(Malnati, 2024)	Auto HTML	-	√	√	-	√
(F. Oliveira et al., 2020)	LAUM	-	√	-	-	√
(G. M. De Oliveira et al., 2023)	DAI	-	√	-	-	√
(Santoso, 2024)	Weblify	√	-	√	√	-
(Penelitian ini)	Dualite	√	√	√	√	√

Tabel 2.4 menyajikan perbandingan fitur dari beberapa plugin Figma yang digunakan dalam penelitian sebelumnya. Setiap plugin memiliki keunggulan dan keterbatasan tersendiri, seperti Teleport HQ yang hanya mendukung Auto Layout dan manajemen file, atau Locofy yang belum mendukung akses bebas dan React JS.

Plugin lain seperti Builder.io dan Auto HTML telah mendukung struktur kode dan React JS, namun masih memiliki keterbatasan pada fitur lainnya. Dalam penelitian ini, plugin Dualite menonjol karena mendukung fitur lengkap: akses gratis, struktur kode, dukungan React JS, Auto Layout, dan manajemen file terstruktur, menjadikannya lebih unggul dalam proses generate code dari desain Figma.

Untuk memperkuat identifikasi ruang penelitian (research gap), peneliti juga melakukan pemetaan menggunakan alat bantu VOSviewer. Data bibliografis diperoleh melalui aplikasi *Publish or Perish (PoP)* dengan memasukkan beberapa kata kunci utama, antara lain: *"Figma plugin" AND "code generation"*, *"Figma to code" AND "React JS"*, *"Static code analysis" AND "SonarQube"*, dan variasi lainnya.

Hasil pemetaan dari lebih dari 600 publikasi menunjukkan bahwa topik penelitian yang membahas plugin Figma masih berfokus pada aspek visual, desain antarmuka, dan aksesibilitas. Sementara itu, topik mengenai SonarQube dan analisis kode lebih banyak digunakan dalam konteks pengembangan kode manual dan keamanan perangkat lunak. Belum terlihat adanya keterhubungan langsung antara *Figma to code* dengan *analisis kualitas kode* secara menyeluruh. Hal ini dapat dilihat dari jarangnya node yang mengaitkan kata kunci seperti *code generation*, *static analysis*, dan *UI consistency* dalam satu jalur kuat.



Gambar 2. 7 Pemetaan VOS Viwer

Hasil ini memperkuat argumen bahwa penelitian terkait evaluasi kualitas kode hasil generate dari plugin desain seperti Dualite masih sangat terbatas. Oleh karena itu, penelitian ini berkontribusi untuk mengisi kekosongan tersebut dengan melakukan evaluasi menyeluruh terhadap aspek visual-fungsional serta aspek teknis kode menggunakan SonarQube.

2.8 Metode Eksperimen

Metode eksperimen merupakan pendekatan penelitian yang digunakan untuk menguji pengaruh suatu perlakuan (*treatment*) terhadap objek yang diteliti dalam kondisi yang terkendali (Sugiyono, 2020; Wulan, 2020). Menurut (Sugiyono, 2020), yang diacu dalam studi pemanfaatan metode eksperimen, penelitian ini dirancang untuk mencari pengaruh suatu perlakuan bagi objek penelitian dalam situasi yang terkontrol Dalam metode ini, peneliti secara sengaja memberikan perlakuan tertentu dan membandingkannya dengan yang tidak diberi perlakuan. Tujuan utamanya adalah untuk mengetahui adanya perbedaan hasil akibat dari perlakuan yang diberikan (Sheraz et al., 2022).

Pada penelitian ini, metode eksperimen digunakan untuk membandingkan hasil generate kode dari plugin Dualite pada dua kondisi desain yang berbeda. Kondisi A merupakan desain yang telah diberi perlakuan berupa pengaturan *Auto Layout*, penamaan frame otomatis menggunakan plugin Autoname, tagging elemen khusus seperti *#link#* dan *#button#,* serta konversi aset dari format SVG ke PNG. Sementara itu, Kondisi B merupakan desain tanpa perlakuan tersebut.

Kedua kondisi diuji dengan proses generate kode menggunakan plugin Dualite, kemudian dievaluasi dari dua aspek utama, yaitu tampilan visual dan fungsionalitas antarmuka web, serta kualitas teknis kode menggunakan alat bantu analisis statis *SonarQube*. Perbandingan hasil antara kedua kondisi dilakukan berdasarkan data kuantitatif berupa jumlah baris kode, jumlah masalah (bugs, code smells, dan vulnerabilities), serta klasifikasi tingkat keparahan *(Severity)*. Hasil ini dianalisis menggunakan rumus rasio dan persentase sebagai dasar interpretasi terhadap dampak perlakuan pada kualitas hasil generate kode (Dualite.dev, 2023).