# **Blueprint Programmer Andal: Dari Perencanaan Strategis hingga Penguasaan Tools Modern untuk Pengembangan Website**

## **Pendahuluan**

Dalam dunia pengembangan perangkat lunak, tantangan umum seperti siklus *debugging* yang memakan waktu seringkali bukan sekadar masalah teknis, melainkan gejala dari proses kerja yang kurang terstruktur. Pengalaman menghabiskan berjam-jam untuk mencari kesalahan yang sebenarnya bisa dicegah oleh sebuah *linter* adalah cerminan dari pendekatan reaktif, di mana pengodean dilakukan sebelum perencanaan dan persiapan yang matang. Laporan ini dirancang sebagai panduan definitif untuk mengubah pola pikir tersebut, beralih dari sekadar "menulis kode" menjadi "merekayasa sistem" yang tangguh, dapat dipelihara, dan berkualitas tinggi.

Visi dari laporan ini adalah untuk menyajikan sebuah *blueprint* holistik yang mengintegrasikan metodologi kerja profesional. Ini bukan sekadar daftar *tools*, melainkan sebuah kerangka kerja yang dimulai dari fondasi strategis, dilanjutkan dengan perancangan cetak biru arsitektural, penguasaan perangkat kerja modern, hingga pengelolaan alur kerja yang efisien. Dengan mengikuti pendekatan ini, seorang programmer dapat secara signifikan mengurangi waktu *debugging*, meningkatkan kualitas kode, dan membangun aplikasi web yang sukses secara konsisten.

Laporan ini terstruktur menjadi empat bagian utama. Bagian pertama, **Fondasi Strategis**, akan membahas pentingnya perencanaan sebelum satu baris kode pun ditulis. Bagian kedua, **Cetak Biru Arsitektural**, akan memandu dalam merancang kerangka teknis sistem. Bagian ketiga, **Perangkat Kerja Developer Modern**, akan secara langsung menjawab kebutuhan akan penguasaan *tools* esensial yang menjadi inti dari produktivitas dan kualitas. Terakhir, Bagian keempat, **Mengelola Alur Kerja**, akan menyatukan semua elemen menjadi sebuah proses eksekusi yang terkelola dengan baik.

## **Bagian 1: Fondasi Strategis - Perencanaan Sebelum Kode**

Fase perencanaan adalah tahapan yang paling krusial namun seringkali diremehkan. Keberhasilan atau kegagalan sebuah proyek website seringkali ditentukan di sini, jauh sebelum proses pengembangan teknis dimulai. Bagian ini menguraikan langkah-langkah strategis untuk membangun fondasi yang kokoh, memastikan setiap upaya pengembangan memiliki tujuan yang jelas dan terarah.

### **1.1 Mendefinisikan Visi dan Tujuan Proyek**

Persiapan yang matang adalah langkah pertama dan paling vital dalam siklus hidup pengembangan website.1 Tanpa visi yang jelas, proyek akan berjalan tanpa arah, rentan terhadap penambahan fitur yang tidak perlu (

*scope creep*), dan pada akhirnya, berisiko gagal memenuhi ekspektasi bisnis maupun pengguna.

Langkah awal adalah menetapkan tujuan utama atau "The Why" dari pembuatan website. Apakah tujuannya untuk menjadi wadah memperkenalkan produk, berfungsi sebagai media pemasaran digital, menjadi representasi resmi individu atau perusahaan, atau sebagai alat untuk mengenali kebutuhan pasar melalui interaksi pengguna?.1 Tujuan ini harus diidentifikasi sejak awal untuk membangun fondasi yang kuat bagi website.4

Untuk mengubah tujuan yang bersifat umum menjadi target yang konkret, disarankan untuk menggunakan kerangka kerja SMART (*Specific, Measurable, Achievable, Realistic, Time-bound*).5 Sebagai contoh, tujuan "meningkatkan penjualan" dapat diubah menjadi "meningkatkan penjualan produk sepatu lari model Z sebesar 20% melalui platform website dalam 6 bulan pertama setelah peluncuran." Tujuan yang SMART memberikan metrik yang jelas untuk mengukur keberhasilan.

Selanjutnya, identifikasi target audiens menjadi sangat penting. Terdapat perbedaan krusial antara *target market* dan *target audiens*. *Target market* adalah kelompok konsumen yang lebih luas yang menjadi sasaran penjualan produk secara keseluruhan, sementara *target audiens* adalah segmen yang lebih spesifik di dalam *target market* yang menjadi sasaran langsung dari pesan atau kampanye pemasaran, dalam hal ini, website itu sendiri.6 Mendefinisikan audiens secara detail—berdasarkan demografi, minat, perilaku, dan

*pain points* (masalah yang mereka hadapi)—akan sangat memengaruhi pilihan bahasa, gaya desain, dan fitur yang akan dikembangkan.8

Analisis kompetitor juga merupakan bagian tak terpisahkan dari fase persiapan. Dengan menganalisis website pesaing, tim dapat mengidentifikasi fitur-fitur standar industri, menemukan celah untuk diferensiasi, dan belajar dari kesalahan yang mungkin telah mereka buat.4

Semua informasi ini harus didokumentasikan dalam sebuah ringkasan kreatif atau *project brief*. Dokumen ini berfungsi sebagai panduan atau acuan utama yang berisi tujuan, sasaran pengunjung, dan garis besar konten yang akan ditampilkan, memastikan semua pemangku kepentingan memiliki pemahaman yang sama.10 Kegagalan dalam mendefinisikan tujuan dan audiens secara spesifik di awal adalah akar dari

*scope creep*. Jika tidak jelas siapa yang akan menggunakan website dan untuk tujuan apa, setiap permintaan fitur baru menjadi sulit untuk divalidasi. Hal ini membuka pintu bagi penambahan fitur yang tidak esensial, yang pada akhirnya menyebabkan proyek membengkak, jadwal terlewat, dan anggaran terlampaui. Investasi waktu dalam perencanaan strategis di awal secara langsung mengurangi risiko teknis dan finansial di kemudian hari.

### **1.2 Menerjemahkan Visi menjadi Fitur dengan User Stories**

Setelah visi dan tujuan proyek ditetapkan, langkah selanjutnya adalah menerjemahkannya menjadi daftar fitur yang konkret dan dapat dikerjakan. Salah satu metode paling efektif untuk menjembatani antara tujuan bisnis dan tugas teknis adalah dengan menggunakan *User Stories* dalam kerangka kerja *Agile*. *User story* adalah penjelasan informal dan umum tentang sebuah fitur perangkat lunak yang ditulis dari perspektif pengguna akhir.11 Ini adalah unit kerja terkecil yang menempatkan pengguna sebagai pusat dari setiap percakapan pengembangan.11

Struktur umum dari sebuah *user story* adalah: **"Sebagai [persona], saya ingin [melakukan sesuatu], sehingga [mendapatkan nilai/manfaat]"**.11 Format ini memaksa tim untuk selalu menjawab tiga pertanyaan fundamental:

*Siapa* yang kita layani? *Apa* yang mereka butuhkan? dan *Mengapa* mereka membutuhkannya?

Manfaat utama penggunaan *user stories* meliputi:

* **Menjaga Fokus pada Pengguna:** Tim tetap fokus untuk memecahkan masalah nyata bagi pengguna, bukan sekadar menyelesaikan daftar tugas.11
* **Mendorong Kolaborasi:** Dengan tujuan akhir yang jelas, seluruh tim (developer, desainer, manajer produk) dapat berkolaborasi untuk menemukan cara terbaik melayani pengguna.11
* **Memicu Solusi Kreatif:** *User stories* mendorong tim untuk berpikir kritis dan kreatif tentang bagaimana cara terbaik mencapai tujuan akhir, bukan terpaku pada satu implementasi teknis.11
* **Menciptakan Momentum:** Setiap *user story* yang berhasil diselesaikan memberikan kemenangan kecil bagi tim, yang dapat mendorong momentum dan motivasi.11

Dalam menulis *user stories* yang baik, penting untuk fokus pada niat atau tujuan pengguna, bukan pada detail implementasi UI.11 Cerita harus dibuat singkat, sederhana, dan idealnya dapat diselesaikan dalam satu siklus pengembangan atau

*sprint*.11 Sebagai contoh, jika tujuan bisnisnya adalah "meningkatkan interaksi pengguna," ini dapat diterjemahkan menjadi

*user story* yang spesifik: "Sebagai *pengguna terdaftar*, saya ingin *mengundang teman-teman saya melalui email*, sehingga *kami dapat menikmati layanan ini bersama-sama*".11

Kumpulan dari *user stories* ini kemudian akan membentuk *product backlog*, yang merupakan daftar prioritas dari semua pekerjaan yang perlu dilakukan. *Backlog* ini menjadi sumber utama untuk perencanaan *sprint* dan pengembangan iteratif, mengubah dokumen perencanaan strategis menjadi serangkaian tugas yang jelas, terukur, dan siap untuk dieksekusi oleh tim pengembangan.13

## **Bagian 2: Cetak Biru Arsitektural - Merancang Kerangka Sistem**

Setelah fondasi strategis diletakkan, langkah selanjutnya adalah membangun cetak biru teknis. Fase ini berfokus pada desain tingkat tinggi yang akan menjadi tulang punggung aplikasi. Keputusan yang dibuat di sini akan secara fundamental menentukan bagaimana sistem dibangun, seberapa mudah sistem tersebut untuk dikembangkan, dipelihara, dan diskalakan di masa depan. Mengabaikan fase ini sama dengan membangun gedung tanpa rancangan arsitektur, yang pasti akan menimbulkan masalah struktural di kemudian hari.

### **2.1 Merancang Arsitektur Sistem**

Desain arsitektur adalah proses mendefinisikan komponen-komponen utama perangkat lunak, bagaimana komponen tersebut dialokasikan ke perangkat keras, dan bagaimana mereka saling berinteraksi.14 Tujuan utamanya adalah untuk memfasilitasi kolaborasi tim, mengurangi risiko proyek, memastikan efisiensi, dan yang terpenting, merancang sistem yang dapat tumbuh atau skalabel seiring waktu.15

Dalam pengembangan web modern, terdapat beberapa pola arsitektur yang umum digunakan:

* **Arsitektur Monolitik:** Seluruh aplikasi—mulai dari antarmuka pengguna (*frontend*), logika bisnis (*backend*), hingga lapisan data—dibangun dan di-*deploy* sebagai satu unit tunggal. Arsitektur ini cenderung lebih sederhana untuk dikembangkan dan diterapkan pada tahap awal, menjadikannya pilihan yang cocok untuk proyek berskala kecil, prototipe, atau *Minimum Viable Product* (MVP) yang tidak memerlukan skalabilitas tinggi di awal.16
* **Arsitektur *Microservices*:** Aplikasi dipecah menjadi kumpulan layanan-layanan kecil yang independen. Setiap layanan bertanggung jawab atas satu fungsi bisnis spesifik, memiliki basis datanya sendiri, dan berkomunikasi dengan layanan lain melalui API. Pendekatan ini sangat cocok untuk aplikasi yang kompleks dan berskala besar yang membutuhkan fleksibilitas pengembangan, di mana tim yang berbeda dapat mengerjakan layanan yang berbeda secara paralel.15
* **Arsitektur *Serverless*:** Dalam model ini, pengembang tidak perlu mengelola infrastruktur server secara langsung. Mereka hanya menulis dan men-*deploy* fungsi (*functions*) yang akan dijalankan oleh penyedia layanan *cloud* (seperti AWS Lambda) sebagai respons terhadap suatu peristiwa (*event*). Ini sangat ideal untuk aplikasi yang *event-driven* atau memiliki beban kerja yang tidak dapat diprediksi, karena biaya didasarkan pada penggunaan aktual.15

Penting untuk memvisualisasikan arsitektur yang dipilih melalui **diagram arsitektur**. Diagram ini menggunakan representasi visual seperti kotak untuk komponen (misalnya, basis data, aplikasi), panah untuk interaksi, dan label untuk memberikan konteks. Diagram ini menjadi bahasa universal yang memfasilitasi komunikasi dan pemahaman bersama di antara anggota tim, baik teknis maupun non-teknis.15

| Kriteria | Arsitektur Monolitik | Arsitektur Microservices | Arsitektur Serverless |
| --- | --- | --- | --- |
| **Skalabilitas** | Terbatas, menskalakan seluruh aplikasi sekaligus | Tinggi, layanan individu dapat diskalakan secara independen | Sangat tinggi, penskalaan otomatis dikelola oleh penyedia cloud |
| **Kecepatan Pengembangan Awal** | Cepat, karena kesederhanaan struktur | Lambat, membutuhkan penyiapan awal yang lebih kompleks (API, deployment) | Cepat, fokus hanya pada kode fungsi |
| **Kompleksitas** | Rendah, semua kode dalam satu basis | Tinggi, mengelola banyak layanan, jaringan, dan data terdistribusi | Sedang, kompleksitas bergeser ke manajemen fungsi dan orkestrasi |
| **Biaya** | Rendah di awal, bisa menjadi mahal untuk diskalakan | Tinggi di awal (infrastruktur), lebih efisien saat skala besar | Berbasis pemakaian (*pay-per-use*), bisa sangat hemat biaya untuk lalu lintas rendah |
| **Kasus Penggunaan Ideal** | Proyek kecil, MVP, aplikasi dengan logika bisnis sederhana | Aplikasi besar dan kompleks, platform e-commerce, sistem enterprise | Aplikasi berbasis peristiwa, tugas latar belakang, API backend, IoT |
| **Tabel 2.1: Perbandingan Arsitektur Aplikasi Web** (Berdasarkan data dari 16) |  |  |  |

### **2.2 Memilih Technology Stack yang Tepat**

*Technology stack* (tumpukan teknologi) adalah kombinasi dari bahasa pemrograman, *framework*, basis data, dan *tools* lain yang digunakan untuk membangun dan menjalankan sebuah aplikasi web.17 Pemilihan

*tech stack* adalah keputusan krusial yang dipengaruhi oleh pilihan arsitektur sebelumnya dan akan berdampak pada seluruh siklus hidup proyek.

Faktor-faktor kunci yang perlu dipertimbangkan adalah:

* **Ukuran dan Kompleksitas Proyek:** Proyek yang besar dan kompleks memerlukan teknologi yang lebih kuat dan skalabel. Misalnya, sebuah blog sederhana mungkin cukup dengan WordPress (PHP), sementara platform media sosial membutuhkan *stack* yang lebih kompleks seperti Python (Instagram) atau JavaScript/Node.js (LinkedIn).17
* **Biaya Pengembangan:** Anggaran proyek harus mempertimbangkan tidak hanya biaya pengembangan awal tetapi juga biaya pemeliharaan jangka panjang. Tarif developer untuk teknologi populer dan banyak diminati (seperti React) bisa lebih tinggi. Biaya juga mencakup dukungan teknis dan pembaruan pasca-peluncuran.17
* ***Time to Market* (Waktu Peluncuran):** Jika kecepatan adalah prioritas, menggunakan teknologi populer dengan komunitas besar dan banyak sumber daya dapat mempercepat pengembangan. Membangun MVP dengan fitur inti juga merupakan strategi cerdas untuk meluncurkan produk lebih cepat dan menguji pasar.17
* **Skalabilitas:** Sejak awal, penting untuk memikirkan bagaimana aplikasi akan menangani peningkatan beban pengguna atau penambahan fitur di masa depan. Arsitektur dan teknologi yang dipilih harus mendukung penskalaan, baik secara vertikal (menambah kekuatan server) maupun horizontal (menambah jumlah server).17
* **Ketersediaan Talenta:** Memilih *stack* teknologi yang populer, seperti React untuk *frontend* 19 dan Node.js untuk  
  *backend* 20, akan memudahkan dalam mencari dan merekrut developer yang kompeten.

### **2.3 Membangun Fondasi Data: Desain Skema Database**

Data adalah jantung dari hampir semua aplikasi web. Merancang skema *database* yang baik adalah fondasi untuk memastikan data disimpan secara efisien, konsisten, dan mudah diakses. Proses desain ini melibatkan langkah-langkah sistematis 21:

1. **Menentukan Tujuan:** Pahami informasi apa yang perlu disimpan dan pertanyaan apa yang ingin dijawab oleh *database*.
2. **Mengorganisir Informasi:** Kumpulkan semua tipe data yang dibutuhkan.
3. **Membagi ke dalam Tabel:** Kelompokkan informasi ke dalam entitas utama (misalnya, Produk, Pesanan, Pengguna), di mana setiap entitas menjadi sebuah tabel.
4. **Menentukan Kolom:** Setiap item informasi dalam sebuah tabel menjadi kolom (misalnya, tabel Pengguna memiliki kolom nama\_depan, nama\_belakang, email).
5. **Menentukan *Primary Key*:** Pilih satu kolom (atau kombinasi kolom) yang secara unik mengidentifikasi setiap baris dalam tabel, seperti id\_pengguna.
6. **Menetapkan Relasi:** Tentukan bagaimana data antar tabel saling berhubungan menggunakan *foreign key*.

Beberapa prinsip penting dalam desain *database* yang baik antara lain **normalisasi** untuk menghindari duplikasi data, **integritas data** untuk menjaga konsistensi, dan **keterbacaan** melalui penamaan yang jelas.23 Penting juga untuk memecah informasi ke dalam bagian logika terkecil (misalnya, pisahkan nama depan dan belakang) dan menghindari penyimpanan data yang dapat dihitung (misalnya, simpan harga dan kuantitas, bukan total harga).21

Terdapat dua model *database* utama: **Relasional (SQL)** seperti MySQL dan PostgreSQL yang terstruktur dalam tabel 24, dan

**Non-Relasional (NoSQL)** seperti MongoDB yang lebih fleksibel dan sering digunakan untuk data bervolume tinggi.17

### **2.4 Merancang Antarmuka dan Pengalaman Pengguna (UI/UX)**

Desain antarmuka (UI) dan pengalaman pengguna (UX) harus dibuat sebelum tahap pengodean dimulai. Proses ini berfungsi untuk memvisualisasikan rancangan, memastikan alur pengguna logis, dan menyelaraskan tampilan dengan identitas merek.4

Proses desain biasanya melalui dua tahap utama:

1. ***Wireframe*:** Ini adalah kerangka dasar atau cetak biru dari sebuah halaman web, yang fokus pada struktur, tata letak, dan alur informasi tanpa elemen desain visual seperti warna atau logo. Tujuannya adalah untuk memvalidasi fungsionalitas dan alur pengguna.10
2. ***Mockup* (Desain Visual):** Ini adalah evolusi dari *wireframe* yang mencakup elemen visual seperti palet warna, tipografi, ikon, dan gambar. *Mockup* memberikan gambaran yang akurat tentang bagaimana tampilan akhir website nantinya.4

Untuk membuat desain ini, developer dan desainer menggunakan berbagai *tools*. Berikut perbandingan tiga *tools* paling populer:

| Fitur | Figma | Adobe XD | Sketch |
| --- | --- | --- | --- |
| **Kolaborasi Real-time** | Unggul, berbasis web memungkinkan kolaborasi tim yang mulus | Baik, dengan fitur Coediting, namun tidak se-instan Figma | Terbatas, membutuhkan langganan dan aplikasi Mac yang kompatibel |
| **Platform** | Web, Windows, macOS (Sangat Fleksibel) | Windows, macOS (Desktop) | Khusus macOS |
| **Prototyping** | Kuat dan terintegrasi | Sangat kuat, dengan fitur auto-animate untuk transisi dinamis | Fungsional, namun lebih dasar dibandingkan kompetitor |
| **Ekosistem Plugin** | Berkembang pesat dan aktif | Cukup baik, terintegrasi dengan ekosistem Adobe | Sangat matang dan luas |
| **Keunggulan Utama** | Kolaborasi tim, aksesibilitas lintas platform, ramah pemula | Integrasi dengan Adobe Creative Cloud, prototyping canggih | Spesialis UI/UX untuk ekosistem Apple, kontrol vektor yang presisi |
| **Tabel 2.4: Perbandingan Alat Desain UI/UX** (Berdasarkan data dari 25) |  |  |  |

### **2.5 Mendefinisikan Protokol Komunikasi: Desain API**

Dalam arsitektur modern, terutama *microservices* atau aplikasi dengan pemisahan *frontend* dan *backend* (*headless*), *Application Programming Interface* (API) berfungsi sebagai jembatan komunikasi. Merancang API yang baik sangat penting untuk memastikan komponen-komponen sistem dapat berinteraksi secara efisien dan andal.

REST (*Representational State Transfer*) adalah gaya arsitektur yang paling populer untuk merancang API web. Beberapa prinsip utamanya adalah 27:

* **Arsitektur Client-Server:** Adanya pemisahan yang jelas antara klien (yang meminta data) dan server (yang menyediakan data). Ini memungkinkan keduanya untuk dikembangkan secara independen.
* ***Stateless* (Tanpa Status):** Setiap permintaan (*request*) dari klien ke server harus berisi semua informasi yang diperlukan server untuk memahami dan memproses permintaan tersebut. Server tidak menyimpan konteks atau status sesi klien di antara permintaan.
* ***Uniform Interface* (Antarmuka Seragam):** Ini adalah batasan fundamental dari REST, yang mencakup:
  + **Identifikasi Sumber Daya:** Setiap sumber daya (misalnya, pengguna, produk) diidentifikasi secara unik melalui URI (contoh: /users/123).
  + **Manipulasi Melalui Representasi:** Klien memanipulasi sumber daya melalui representasinya (biasanya dalam format JSON), bukan dengan mengakses sumber daya secara langsung.
  + **Pesan yang Deskriptif:** Setiap respons dari server harus berisi informasi yang cukup bagi klien untuk memprosesnya, termasuk metadata seperti tipe media.

Pilihan arsitektur di awal memiliki efek domino yang merambat ke seluruh aspek teknis dan operasional proyek. Misalnya, memilih arsitektur *Microservices* untuk skalabilitas akan secara langsung menciptakan kebutuhan turunan: setiap layanan perlu berkomunikasi, yang mengharuskan adanya desain API yang solid seperti REST. Mengelola puluhan layanan secara manual menjadi tidak praktis, yang mendorong adopsi *tools* seperti Docker dan Kubernetes. Melacak pekerjaan yang terdistribusi di antara tim-tim yang berbeda juga menjadi lebih kompleks, membuat alat manajemen proyek terstruktur seperti Jira lebih sesuai daripada Trello yang lebih sederhana. Keputusan arsitektural tunggal ini pada akhirnya mendikte *tech stack*, *toolchain*, dan bahkan struktur tim yang dibutuhkan.

## **Bagian 3: Perangkat Kerja Developer Modern - Menguasai Lingkungan Anda**

Bagian ini merupakan inti dari laporan yang secara langsung menjawab pertanyaan tentang *tools* yang harus dipahami sebelum melakukan pengodean. Penguasaan perangkat kerja modern bukan lagi sebuah pilihan, melainkan sebuah keharusan. *Tools* ini bukan sekadar "tambahan", melainkan fondasi dari *Developer Experience* (DevEx) yang efisien, yang memungkinkan produksi kode berkualitas tinggi secara konsisten dan mengurangi waktu yang terbuang untuk tugas-tugas manual dan *debugging* yang tidak perlu.

### **3.1 Batu Penjuru: Kontrol Versi dengan Git dan GitHub**

*Version Control System* (VCS) adalah sistem yang wajib dikuasai oleh setiap programmer andal. VCS berfungsi untuk melacak setiap modifikasi yang dilakukan pada kode sumber, memungkinkan banyak developer bekerja pada proyek yang sama secara bersamaan tanpa menimbulkan konflik, dan memfasilitasi perbaikan *bug* dengan memungkinkan tim untuk kembali ke versi kode sebelumnya yang stabil.29 Tanpa VCS, kolaborasi dalam tim pengembangan modern hampir mustahil untuk dilakukan secara efektif.

Terdapat dua jenis utama VCS: *Centralized* (CVCS) dan *Distributed* (DVCS). Git adalah contoh DVCS yang paling populer saat ini. Dalam DVCS, setiap developer memiliki salinan lengkap dari seluruh riwayat proyek di mesin lokal mereka, yang memungkinkan mereka untuk bekerja secara *offline* dan memberikan fleksibilitas yang lebih besar.30

Konsep inti dalam Git yang harus dipahami meliputi:

* **Repository (Repo):** Folder proyek yang dilacak oleh Git.
* **Commit:** "Snapshot" atau rekaman dari perubahan yang telah disimpan.
* **Branch:** Versi independen dari *repository*. Praktik terbaik adalah mengembangkan fitur baru atau memperbaiki *bug* di *branch* terpisah untuk menjaga stabilitas *branch* utama (biasanya main atau master).32
* **Merge:** Proses menggabungkan perubahan dari satu *branch* ke *branch* lainnya.

GitHub, di sisi lain, bukanlah Git itu sendiri. GitHub adalah platform hosting berbasis web untuk repositori Git yang menyediakan lapisan fitur kolaborasi di atas Git, seperti *Pull Requests* (untuk mengajukan dan meninjau perubahan), *Issue Tracking*, dan *Code Review*.32

### **3.2 Pusat Komando Anda: Mengoptimalkan Visual Studio Code**

Visual Studio Code (VS Code) telah menjadi editor kode de facto bagi banyak web developer. VS Code bersifat gratis, ringan, berjalan di berbagai platform (Windows, macOS, Linux), dan yang terpenting, sangat dapat dikustomisasi melalui ekosistem ekstensinya yang luas.34

Setelah menginstal VS Code, langkah selanjutnya adalah mengoptimalkannya dengan ekstensi esensial yang dapat meningkatkan produktivitas secara drastis:

* **Live Server:** Meluncurkan server pengembangan lokal dengan fitur *live reload*. Setiap kali file disimpan, halaman web di browser akan diperbarui secara otomatis, mempercepat siklus *feedback* visual.36
* **Path Intellisense:** Secara otomatis melengkapi nama file dan path, sangat membantu dalam proyek dengan struktur folder yang kompleks dan mengurangi kesalahan pengetikan.36
* **GitLens:** Meningkatkan kemampuan Git bawaan VS Code. Ekstensi ini memungkinkan developer untuk melihat siapa yang mengubah setiap baris kode (*blame*), menavigasi riwayat *commit*, dan membandingkan *branch* langsung dari dalam editor.36
* **REST Client:** Memungkinkan pengiriman permintaan HTTP dan melihat respons API langsung di dalam VS Code. Ini sangat berguna untuk menguji *endpoint backend* tanpa perlu beralih ke aplikasi lain seperti Postman.38
* **JavaScript (ES6) Code Snippets:** Menyediakan *shortcut* untuk sintaks JavaScript modern yang umum digunakan, seperti import, export, dan *arrow functions*, mempercepat penulisan kode boilerplate.36

### **3.3 Otomatisasi Kualitas: Linter dan Formatter (ESLint & Prettier)**

Inilah inti dari masalah yang sering dihadapi programmer: *debugging* yang lama akibat kesalahan sepele. *Linter* dan *formatter* adalah *tools* yang secara proaktif mencegah masalah ini terjadi.

Terdapat perbedaan fundamental antara keduanya:

* **ESLint (sebagai *Linter*):** Adalah alat analisis kode statis. Tugasnya adalah memeriksa kode JavaScript untuk menemukan pola-pola yang bermasalah atau tidak sesuai dengan praktik terbaik (*code-quality rules*). ESLint dapat mendeteksi potensi *bug* (seperti variabel yang dideklarasikan tapi tidak pernah digunakan), penggunaan sintaks yang usang, atau pelanggaran aturan keamanan.39 Singkatnya, ESLint berfokus pada  
  **kebenaran** dan **kualitas** fungsional kode.
* **Prettier (sebagai *Formatter*):** Adalah sebuah *opinionated code formatter*. Tujuannya hanya satu: memastikan gaya penulisan kode (*code style*) konsisten di seluruh proyek (*formatting rules*). Prettier akan secara otomatis mengatur ulang spasi, inden, titik koma, panjang baris, dan elemen gaya lainnya sesuai dengan aturan yang telah ditentukan.42 Prettier berfokus pada  
  **estetika** dan **konsistensi** visual kode.

Mengapa menggunakan keduanya? Karena mereka saling melengkapi dengan sempurna.44 Dengan mengintegrasikan keduanya, developer dapat menciptakan alur kerja yang sangat efisien. Prettier akan secara otomatis merapikan kode setiap kali file disimpan, menghilangkan perdebatan tentang gaya penulisan di dalam tim. Sementara itu, ESLint akan bekerja di latar belakang, memberikan peringatan

*real-time* tentang potensi *bug* atau masalah kualitas. Untuk memastikan keduanya bekerja tanpa konflik, digunakan eslint-config-prettier yang berfungsi untuk menonaktifkan semua aturan gaya dari ESLint, sehingga Prettier menjadi satu-satunya sumber kebenaran untuk pemformatan.46

### **3.4 Mengelola Dependensi: NPM vs. Yarn**

*Package manager* adalah alat yang mengotomatiskan proses instalasi, pembaruan, konfigurasi, dan penghapusan pustaka (*packages*) atau modul pihak ketiga yang menjadi dependensi sebuah proyek.48 Dua

*package manager* paling dominan di ekosistem JavaScript adalah NPM dan Yarn.

| Aspek | NPM (Node Package Manager) | Yarn (Yet Another Resource Negotiator) |
| --- | --- | --- |
| **Kecepatan Instalasi** | Telah meningkat pesat, namun secara historis lebih lambat karena instalasi sekuensial. | Umumnya lebih cepat karena memanfaatkan instalasi paralel dan mekanisme caching yang agresif. |
| **Lock File** | package-lock.json untuk memastikan instalasi yang konsisten. | yarn.lock untuk memastikan instalasi yang deterministik dan konsisten di berbagai lingkungan. |
| **Keamanan** | Memiliki perintah npm audit untuk memindai kerentanan keamanan pada dependensi. | Juga memiliki perintah yarn audit dan melakukan verifikasi integritas paket saat instalasi. |
| **Fitur Unggulan** | Terintegrasi secara default dengan Node.js, komunitas dan registri paket terbesar. | *Workspaces* untuk manajemen *monorepo* yang efisien, *Plug'n'Play* (PnP) untuk instalasi super cepat. |
| **Komunitas** | Sangat besar dan menjadi standar de facto bagi sebagian besar proyek dan tutorial. | Kuat, terutama di kalangan perusahaan besar dan proyek yang memprioritaskan performa. |
| **Tabel 3.4: Perbandingan *Package Manager* (NPM vs. Yarn)** (Berdasarkan data dari 48) |  |  |

### **3.5 Mesin Pembangun: Memahami Build Tools (Vite & Webpack)**

Dalam pengembangan web modern, kode yang ditulis developer bukanlah kode yang dijalankan langsung oleh browser. Diperlukan sebuah proses "membangun" (*build*) yang ditangani oleh *build tools*. Fungsi krusial dari *build tools* antara lain 52:

* ***Bundling*:** Menggabungkan ratusan file JavaScript, CSS, dan aset lainnya menjadi beberapa file yang dioptimalkan.
* ***Transpilation*:** Mengubah kode modern yang tidak didukung semua browser (seperti JSX dari React, TypeScript, atau sintaks ES6+) menjadi JavaScript versi lama yang lebih kompatibel.
* **Optimasi:** Melakukan minifikasi (menghapus karakter yang tidak perlu), *tree-shaking* (menghapus kode yang tidak terpakai), dan optimasi lainnya untuk mengurangi ukuran file.
* ***Hot Module Replacement* (HMR):** Memperbarui modul yang diubah di browser secara instan tanpa perlu memuat ulang seluruh halaman, yang secara dramatis mempercepat proses pengembangan.

Dua *build tools* yang paling menonjol saat ini adalah Vite dan Webpack.

| Kriteria | Vite | Webpack |
| --- | --- | --- |
| **Kecepatan Dev Server** | Sangat cepat, hampir instan. Menggunakan *native ES modules* untuk melayani file sesuai permintaan. | Lebih lambat, perlu membundel seluruh aplikasi sebelum server siap. |
| **Konfigurasi** | Minimalis dan "out-of-the-box". Dirancang untuk kesederhanaan dan kemudahan penggunaan. | Sangat fleksibel dan dapat dikonfigurasi secara mendalam, namun bisa menjadi sangat kompleks. |
| **Hot Module Replacement (HMR)** | Secepat kilat dan sangat efisien, hanya memperbarui modul yang berubah. | Fungsional, tetapi umumnya lebih lambat karena pendekatan berbasis *bundle*. |
| **Ekosistem Plugin** | Berkembang pesat dan kompatibel dengan banyak plugin Rollup. | Sangat matang dan luas, memiliki plugin untuk hampir semua kebutuhan. |
| **Kasus Penggunaan Ideal** | Proyek baru, aplikasi modern (SPA), developer yang memprioritaskan kecepatan dan kesederhanaan. | Proyek *legacy*, aplikasi berskala besar dan kompleks, kebutuhan kustomisasi build yang mendalam. |
| **Tabel 3.5: Perbandingan *Build Tools* (Vite vs. Webpack)** (Berdasarkan data dari 53) |  |  |

Penguasaan *toolchain* ini secara holistik adalah pembeda utama antara seorang *coder* dan seorang *software engineer*. *Tools* ini menciptakan sebuah *pipeline* kualitas otomatis: developer menulis kode di VS Code, ESLint memberikan peringatan *real-time*, Prettier merapikan kode saat disimpan, Git hooks memastikan hanya kode yang lolos pemeriksaan yang bisa di-*commit*, dan *build tool* mengoptimalkan kode bersih tersebut untuk produksi. Kegagalan pada satu tahap, seperti tidak menggunakan *linter*, akan menciptakan efek riak yang menyebabkan masalah di tahap selanjutnya, seperti *bug* di lingkungan produksi dan waktu *debugging* yang lama.

## **Bagian 4: Mengelola Alur Kerja - Eksekusi dan Dokumentasi**

Setelah fondasi strategis diletakkan dan perangkat kerja modern disiapkan, keberhasilan proyek bergantung pada eksekusi yang terkelola dengan baik dan dokumentasi yang solid. Bagian ini menyatukan semua elemen sebelumnya ke dalam sebuah proses kerja yang kohesif, memastikan proyek berjalan sesuai rencana, dapat dilacak, dan mudah dipelihara di masa depan.

### **4.1 Mengorkestrasi Proyek: Trello vs. Jira**

Manajemen proyek yang efektif adalah kunci untuk memastikan *timeline* terpenuhi, sumber daya dialokasikan dengan benar, dan semua anggota tim selaras dengan tujuan yang sama.4 Dua alat yang paling dominan dalam manajemen proyek perangkat lunak adalah Trello dan Jira.

* **Trello:** Adalah alat manajemen proyek yang sangat visual dan fleksibel, berbasis papan Kanban. Pengguna dapat membuat papan untuk setiap proyek, daftar untuk setiap tahapan (*To Do, In Progress, Done*), dan kartu untuk setiap tugas. Karena kesederhanaan dan antarmuka *drag-and-drop* yang intuitif, Trello sangat ideal untuk tim kecil, proyek dengan alur kerja yang tidak terlalu rumit, atau untuk manajemen tugas pribadi.56
* **Jira:** Adalah alat manajemen proyek yang jauh lebih kuat dan terstruktur, dirancang secara khusus untuk tim pengembangan perangkat lunak yang menggunakan metodologi *Agile* (seperti Scrum dan Kanban). Jira unggul dalam pelacakan *issue* yang detail, perencanaan *sprint*, manajemen *backlog*, pembuatan laporan yang canggih, dan pengelolaan alur kerja yang kompleks.56

Pemilihan antara keduanya sangat bergantung pada kompleksitas proyek. Sebuah proyek sederhana dengan arsitektur monolitik dan tim kecil dapat dikelola dengan baik menggunakan Trello. Namun, proyek yang menggunakan arsitektur *microservices* dengan banyak dependensi antar tim dan alur kerja yang rumit akan lebih efektif dikelola menggunakan Jira, yang mampu menangani kompleksitas tersebut. Menariknya, kedua alat ini, yang merupakan bagian dari keluarga Atlassian, dapat diintegrasikan. Tim dapat menggunakan Trello untuk perencanaan tingkat tinggi atau manajemen tugas individu, sementara pekerjaan teknis yang detail dilacak dan dikelola di Jira.56

| Kriteria | Trello | Jira |
| --- | --- | --- |
| **Target Pengguna** | Tim umum, non-teknis, tim kecil, manajemen tugas pribadi | Tim pengembangan perangkat lunak, tim Agile, proyek enterprise |
| **Metode Agile** | Mendukung Kanban secara visual dan sederhana | Dirancang untuk Scrum dan Kanban dengan fitur khusus (backlog, sprint, burndown chart) |
| **Kompleksitas** | Rendah, sangat mudah digunakan dan fleksibel | Tinggi, kurva belajar lebih curam namun sangat kuat dan dapat dikustomisasi |
| **Kurva Belajar** | Sangat cepat, intuitif untuk pemula | Membutuhkan waktu untuk dipelajari, terutama untuk konfigurasi tingkat lanjut |
| **Harga** | Menawarkan versi gratis yang sangat fungsional, paket berbayar lebih terjangkau | Lebih mahal, terutama untuk tim besar yang membutuhkan fitur premium |
| **Kasus Penggunaan Terbaik** | Proyek sederhana, perencanaan konten, *onboarding* karyawan, alur kerja visual | Pengembangan perangkat lunak, pelacakan bug, manajemen rilis, proyek berskala besar |
| **Tabel 4.1: Perbandingan Alat Manajemen Proyek (Trello vs. Jira)** (Berdasarkan data dari 56) |  |  |

### **4.2 Pahlawan Tanpa Tanda Jasa: Peran Kritis Dokumentasi Teknis**

Dokumentasi sering dianggap sebagai tugas yang membosankan, namun perannya sangat krusial dan merupakan fondasi untuk pemeliharaan, kolaborasi, dan skalabilitas jangka panjang sebuah proyek perangkat lunak.60 Dokumentasi yang baik mengurangi ketergantungan pada pengetahuan implisit yang hanya dimiliki oleh developer tertentu, memfasilitasi proses

*onboarding* anggota tim baru, dan mempercepat penyelesaian masalah.61

Ada beberapa jenis dokumentasi penting dalam pengembangan perangkat lunak:

* **Dokumentasi Proses:** Menjelaskan bagaimana proyek dikelola dan dijalankan. Ini bisa berupa rencana proyek, definisi alur kerja, dan prosedur standar tim.60
* **Dokumentasi Produk/Teknis:** Ini adalah manual referensi untuk tim pengembangan. Isinya mencakup penjelasan mendalam tentang arsitektur sistem, desain API (misalnya, spesifikasi *endpoint*, format *request/response*), skema *database*, panduan instalasi lingkungan pengembangan, dan keputusan teknis penting yang dibuat selama proyek.60
* **Dokumentasi Pengguna:** Ditujukan untuk pengguna akhir aplikasi. Ini berisi panduan tentang cara menggunakan fitur-fitur perangkat lunak, FAQ, dan panduan pemecahan masalah dasar.61

Untuk membuat dokumentasi yang efektif, beberapa praktik terbaik perlu diikuti: kenali audiens yang dituju (developer vs. pengguna akhir), gunakan bahasa yang jelas dan sederhana, manfaatkan visual seperti diagram dan grafik untuk menjelaskan konsep kompleks, dan yang terpenting, jaga agar dokumentasi selalu diperbarui seiring dengan perubahan pada kode.60

Ada hubungan simbiosis yang kuat antara kompleksitas arsitektur, pilihan alat manajemen proyek, dan kebutuhan akan dokumentasi. Semakin kompleks arsitektur sistem (misalnya, *microservices*), semakin formal dan terstruktur alat manajemen (seperti Jira) dan dokumentasi (seperti dokumentasi API yang detail) yang dibutuhkan. Menggunakan alat yang tidak sesuai atau mengabaikan dokumentasi pada proyek yang kompleks akan menyebabkan miskomunikasi, kekacauan, dan pada akhirnya, perlambatan atau bahkan kegagalan proyek.

## **Kesimpulan**

Perjalanan untuk menjadi seorang "programmer andal" melampaui kemampuan teknis menulis kode. Seperti yang telah diuraikan dalam laporan ini, prosesnya dimulai jauh sebelum editor kode dibuka dan berlanjut hingga setelah aplikasi diluncurkan. Ini adalah sebuah pergeseran fundamental dari pola pikir seorang *coder* menjadi seorang *software engineer*—dari fokus pada bagaimana membuat kode "berfungsi" menjadi bagaimana membangun sistem yang tangguh, dapat dipelihara, dan berkualitas tinggi.

Laporan ini telah memetakan empat pilar utama yang menopang pendekatan rekayasa ini:

1. **Fondasi Strategis:** Menegaskan bahwa perencanaan yang cermat—mendefinisikan tujuan yang jelas, memahami audiens, dan menerjemahkannya ke dalam *user stories* yang terfokus—adalah langkah preventif paling efektif untuk menghindari *scope creep* dan memastikan proyek tetap pada jalurnya.
2. **Cetak Biru Arsitektural:** Menunjukkan bahwa keputusan arsitektur di awal memiliki efek domino yang menentukan pilihan *tech stack*, desain *database*, dan bahkan alur kerja tim. Merancang cetak biru yang solid adalah investasi untuk skalabilitas dan kemudahan pemeliharaan di masa depan.
3. **Perangkat Kerja Modern:** Mengatasi secara langsung tantangan *debugging* yang tidak efisien dengan memperkenalkan *toolchain* modern. Penguasaan *tools* seperti Git untuk kontrol versi, VS Code dengan ekstensi yang tepat, ESLint dan Prettier untuk kualitas dan konsistensi kode otomatis, serta *build tools* seperti Vite atau Webpack, adalah fondasi dari produktivitas dan kualitas kode yang konsisten.
4. **Manajemen Alur Kerja:** Menyatukan semua elemen melalui manajemen proyek yang terstruktur menggunakan alat seperti Trello atau Jira, dan didukung oleh dokumentasi teknis yang komprehensif. Ini memastikan eksekusi yang lancar dan keberlanjutan proyek jangka panjang.

Menjadi programmer andal berarti mengadopsi sebuah metodologi. Ini adalah tentang memahami "mengapa" di balik setiap fitur, merancang sebelum membangun, dan memanfaatkan *tools* untuk mengotomatiskan kualitas, bukan hanya untuk menulis kode. Investasi waktu untuk mempelajari dan menerapkan prinsip-prinsip serta perangkat kerja yang diuraikan dalam laporan ini akan memberikan hasil berlipat ganda dalam bentuk efisiensi, kualitas kode yang lebih tinggi, lebih sedikit waktu yang terbuang untuk *debugging*, dan pada akhirnya, kepuasan kerja yang lebih besar dalam menciptakan produk digital yang luar biasa.

#### Works cited

1. 6 Tahapan Membuat Website Lengkap untuk Pemula, Tanpa Ribet!, accessed August 9, 2025, <https://www.jagoanhosting.com/blog/tahapan-membuat-website/>
2. Urutan Proses Pembuatan Website yang Efektif untuk Pemula - Whello Indonesia, accessed August 9, 2025, <https://whello.id/tips-digital-marketing/proses-pembuatan-website/>
3. Arsitektur dan Teknik Desain Informasi pada Web - Neliti, accessed August 9, 2025, <https://media.neliti.com/media/publications/243738-arsitektur-dan-teknik-desain-informasi-p-e9db73bd.pdf>
4. 9 Tahapan Membuat Website dari Planning hingga Maintenance, accessed August 9, 2025, <https://www.dewaweb.com/blog/tahapan-membuat-website/>
5. Pengertian & 7 Langkah Rancangan Proyek beserta Kiat Pakar [2025] - Asana, accessed August 9, 2025, <https://asana.com/id/resources/project-design>
6. 5 Cara Menentukan Target Audiens - Telkomsel, accessed August 9, 2025, <https://www.telkomsel.com/enterprise/insight/blog/5-cara-menentukan-target-audiens>
7. Pentingnya Target Audience bagi Bisnis, Ini Cara Menentukannya - www.latifaba.com, accessed August 9, 2025, <https://www.latifaba.com/news-events/pentingnya-target-audience-bagi-bisnis-ini-cara-menentukannya>
8. Target Audience: Definisi, Tipe dan Cara Mengidentifikasinya - Sasana Digital, accessed August 9, 2025, <https://sasanadigital.com/target-audience-adalah/>
9. Apa itu Target Audience? Cara Menentukan & Manfaatnya - Ngalup Artikel, accessed August 9, 2025, <https://ngalup.co/artikel/target-audience/>
10. TAHAPAN MEMBUAT WEBSITE | BINUS UNIVERSITY MALANG, accessed August 9, 2025, <https://binus.ac.id/malang/2020/06/tahapan-membuat-website/>
11. User Stories | Examples and Template | Atlassian, accessed August 9, 2025, <https://www.atlassian.com/agile/project-management/user-stories>
12. How to Write User Stories in Agile Software Development, accessed August 9, 2025, <https://www.easyagile.com/blog/how-to-write-good-user-stories-in-agile-software-development>
13. How to write a user story for Website content... explain to a noob. - Reddit, accessed August 9, 2025, <https://www.reddit.com/r/ProductManagement/comments/s9x3o0/how_to_write_a_user_story_for_website_content/>
14. Desain Arsitektur Sistem, accessed August 9, 2025, <https://repository.dinus.ac.id/docs/ajar/pertemuan-10_11_Desain-Arsitektur-dan-Antarmuka_YUS.pdf>
15. Apa itu Diagram Arsitektur? - Penjelasan Diagram Arsitektur Sistem & Perangkat Lunak - AWS, accessed August 9, 2025, <https://aws.amazon.com/id/what-is/architecture-diagramming/>
16. Arsitektur Aplikasi Web Adalah: Praktik Terbaik Tahun 2025, accessed August 9, 2025, <https://idwebhost.com/blog/arsitektur-aplikasi-web-adalah/>
17. Technology Stack for Web Development: How to Choose the Right ..., accessed August 9, 2025, <https://www.halo-lab.com/blog/technology-stack-for-web-development>
18. 10 Tips to Choose the Best Tech Stack for Web App Development in 2025 | SECL Group, accessed August 9, 2025, <https://seclgroup.com/tips-to-choose-tech-stack-for-web-app-development/>
19. Choosing the best tech stack for web development 2025 - Imaginary Cloud, accessed August 9, 2025, <https://www.imaginarycloud.com/blog/choosing-the-best-tech-stack-for-web-development-2022>
20. What are the current tech stacks that most people use in web dev nowadays? - Reddit, accessed August 9, 2025, <https://www.reddit.com/r/webdev/comments/xg9vvt/what_are_the_current_tech_stacks_that_most_people/>
21. Dasar-dasar desain database - Dukungan Microsoft, accessed August 9, 2025, <https://support.microsoft.com/id-id/topic/dasar-dasar-desain-database-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5>
22. accessed January 1, 1970, [https.support.microsoft.com/id-id/topic/dasar-dasar-desain-database-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5](http://docs.google.com/https.support.microsoft.com/id-id/topic/dasar-dasar-desain-database-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5)
23. Skema Database: Pengertian, Syarat beserta Contohnya - Rumahweb Blog, accessed August 9, 2025, <https://blog.rumahweb.com/skema-database/>
24. Dasar-dasar Desain Basis Data | AppMaster, accessed August 9, 2025, <https://appmaster.io/id/blog/dasar-dasar-desain-basis-data>
25. Figma vs. Sketch vs. Adobe XD: Which Design Tool Is Right for You ..., accessed August 9, 2025, <https://www.coursera.org/articles/figma-vs-sketch-vs-adobe-xd>
26. Best UI/UX Design Tool in 2025: Figma vs Adobe XD vs Sketch - Airbow, accessed August 9, 2025, <https://airbow.io/best-ui-ux-design-tool-in-2025-figma-vs-adobe-xd-vs-sketch/>
27. REST API Tutorial: What is REST?, accessed August 9, 2025, <https://restfulapi.net/>
28. Design Principles RESTful API - Medium, accessed August 9, 2025, <https://medium.com/@alfisalim.12/design-principal-restful-api-e327ebbf2850>
29. Version Control System, Ketahui Jenis dan Fungsinya di Sini ... - Glints, accessed August 9, 2025, <https://glints.com/id/lowongan/version-control-system/>
30. Kenapa Teknologi Version Control System dibutuhkan dalam pengembangan perangkat lunak? - OPTIBIS, accessed August 9, 2025, <https://optibis.id/baca-blog/kenapa-teknologi-version-control-system-dibutuhkan-dalam-pengembangan-perangkat-lunak>
31. Version Control Sistem - PT Teltics Media, accessed August 9, 2025, <https://teltics.com/blog/version-control-sistem>
32. Tutorial Belajar Git dan GitHub untuk Pemula - AMD Academy Indonesia, accessed August 9, 2025, <https://edu.amdacademy.id/home/course/tutorial-belajar-git-dan-github-untuk-pemula/70>
33. Complete Git and GitHub Tutorial - YouTube, accessed August 9, 2025, <https://m.youtube.com/watch?v=apGV9Kg7ics&pp=ygUJI2dpdGluZGlh>
34. Visual Studio Code - Code Editing. Redefined, accessed August 9, 2025, <https://code.visualstudio.com/>
35. Setting up Visual Studio Code, accessed August 9, 2025, <https://code.visualstudio.com/docs/setup/setup-overview>
36. 20 Best Visual Studio Code Extensions for Web Developers, accessed August 9, 2025, <https://www.syncfusion.com/blogs/post/best-vs-code-extensions-web>
37. 10 Extension VSCode Terbaik yang Harus Dimiliki oleh Web Developer - Koding Akademi, accessed August 9, 2025, <https://www.kodingakademi.id/10-extension-vscode-terbaik-yang-harus-dimiliki-oleh-web-developer/>
38. 10 VS Code Extensions for Web Developers in 2024 - GitKraken, accessed August 9, 2025, <https://www.gitkraken.com/blog/10-best-vs-code-extensions-web-development>
39. Find and fix problems in your JavaScript code - ESLint - Pluggable JavaScript Linter, accessed August 9, 2025, <https://eslint.org/>
40. Prettier vs. Linters, accessed August 9, 2025, <https://prettier.io/docs/comparison>
41. Getting Started with ESLint - ESLint - Pluggable JavaScript Linter, accessed August 9, 2025, <https://eslint.org/docs/latest/use/getting-started>
42. Cara Setup Prettier di VS Code - Medium, accessed August 9, 2025, <https://medium.com/@sarascahya/cara-setup-prettier-di-vs-code-812ba908a060>
43. Prettier - Code formatter - Visual Studio Marketplace, accessed August 9, 2025, <https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode>
44. ESLint vs Prettier: What's the Difference and Why ESLint with Prettier is an Ideal Combination? - Aglowid IT Solutions, accessed August 9, 2025, <https://aglowiditsolutions.com/blog/eslint-vs-prettier/>
45. ESLint dan Prettier adalah Kombinasi Terbaik | Gattigaga Hayyuta Dewa, accessed August 9, 2025, <https://gattigaga.com/id/articles/eslint-dan-prettier-adalah-kombinasi-terbaik>
46. prettier/eslint-config-prettier: Turns off all rules that are unnecessary or might conflict with Prettier. - GitHub, accessed August 9, 2025, <https://github.com/prettier/eslint-config-prettier>
47. Integrating with Linters - Prettier, accessed August 9, 2025, <https://prettier.io/docs/integrating-with-linters>
48. Yarn vs NPM: Which package manager should I use? - Imaginary Cloud, accessed August 9, 2025, <https://www.imaginarycloud.com/blog/npm-vs-yarn-which-is-better>
49. Yarn vs. NPM: Which Package Manager You Should Choose, and Why? - Aqua Security, accessed August 9, 2025, <https://www.aquasec.com/cloud-native-academy/supply-chain-security/yarn-vs-npm/>
50. A Developer's Guide to Comparing Yarn vs. NPM - Spectral, accessed August 9, 2025, <https://spectralops.io/blog/a-developers-guide-to-comparing-yarn-vs-npm/>
51. Yarn vs. npm: Choosing the Best Package Manager - Flatirons Development, accessed August 9, 2025, <https://flatirons.com/blog/yarn-vs-npm/>
52. Why Build Tools Are Essential for Modern Front-End Development ..., accessed August 9, 2025, <https://medium.com/@emre.deniz/why-build-tools-are-essential-for-modern-front-end-development-8298616687ed>
53. Vite vs. Webpack: Which JavaScript Bundler Should You Use ..., accessed August 9, 2025, <https://betterstack.com/community/guides/scaling-nodejs/vite-vs-webpack/>
54. Vite vs. Webpack: A Head-to-Head Comparison - Kinsta®, accessed August 9, 2025, <https://kinsta.com/blog/vite-vs-webpack/>
55. Webpack vs Vite: A Detailed Comparison for Web Development - CodeParrot, accessed August 9, 2025, <https://codeparrot.ai/blogs/webpack-vs-vite-a-detailed-comparison-for-modern-web-development>
56. Trello vs. Jira: which to choose (and how to use them together) - Work Life by Atlassian, accessed August 9, 2025, <https://www.atlassian.com/blog/project-management/trello-vs-jira>
57. Trello vs Jira: Which Project Management Tool Wins in 2025? - Planyway, accessed August 9, 2025, <https://planyway.com/blog/trello-vs-jira-comparison-guide>
58. 7 Project Management Tools Terbaik untuk Anda - MTARGET, accessed August 9, 2025, <https://mtarget.co/blog/project-management-tools-terbaik-untuk-anda/>
59. Trello vs. Jira: Which project management tool is right for you? [2025] - Zapier, accessed August 9, 2025, <https://zapier.com/blog/trello-vs-jira/>
60. Apa Itu Dokumen Teknis Perangkat Lunak atau Aplikasi? - Wordsmith Group, accessed August 9, 2025, <https://wordsmithgroup.com/id/dokumen-teknis-perangkat-lunak/>
61. Mengapa Dokumentasi Perangkat Lunak Sangat Penting dalam Rekayasa Perangkat Lunak (RPL)? - UPI Cibiru, accessed August 9, 2025, <https://kd-cibiru.upi.edu/index.php/component/content/article/mengapa-dokumentasi-perangkat-lunak-sangat-penting-dalam-rekayasa-perangkat-lunak-rpl?catid=29&Itemid=101>
62. Dokumentasi yang dibutuhkan dalam sebuah software - VSID, accessed August 9, 2025, <https://www.vsid.info/2024/07/dokumentasi-yang-dibutuhkan-dalam.html>