BAB 2
TINJAUAN PUSTAKA

2.1. Tinjauan Tempat Penelitian

Tinjauan umum perusahaan merupakan pembahasan mengenai profil, struktur organisasi, serta deskripsi jabatan pada perusahaan yang akan menjadi tempat penelitian.

2.1.1. Visi dan Misi

Berikut adalah Visi Misi dari PT. Kinest Kreatif Ideata:

1. Visi

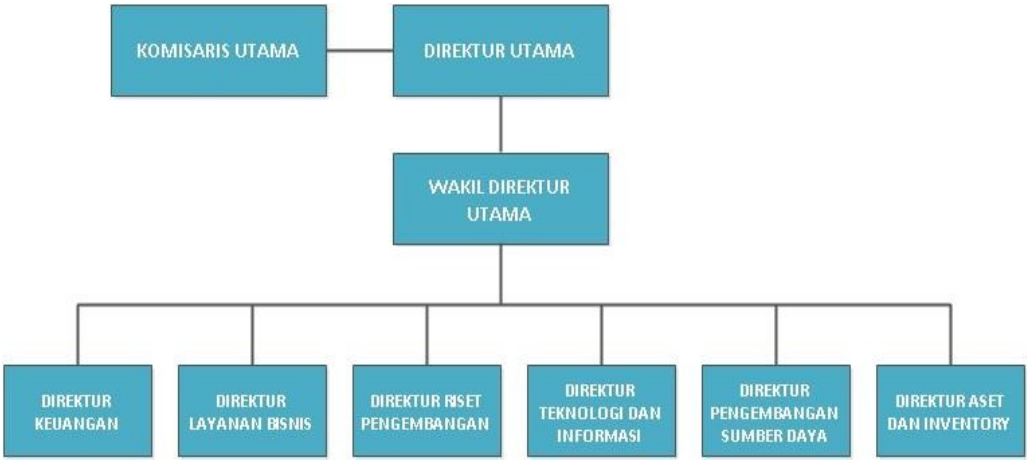
   To be an Indonesian company leader at digital industry.

1. Misi

   a. *Build a strong and coherence teamwork.*
   b. *Generate multilevel partnership network market.*
   c. *Best management quality system.*
   d. *Smooth and efficient management execution.*
   e. *Make strong financial management.*

2.1.2. Struktur Organisasi

Struktur organisasi pada PT. Kinest Kreatif Ideata dapat dilihat pada gambar berikut:



Gambar 2.1: Struktur Organisasi

2.1.3. Deskripsi Kerja Jabatan

Berikut adalah pemaparan deskripsi kerja untuk tiap-tiap jabatan di PT. Kinest Kreatif Ideata:

1. Direktorat Keuangan

   Fokus pada pengelolaan keuangan Perusahaan serta mengendalikan operasi keuangan secara terpusat.

2. Direktorat Layanan Bisnis

   Fokus pada pengembangan bisnis perusahaan secara terpusat melalui produk Kakatu dan Hyjabs.

3. Direktorat Riset dan Pengembangan

   Fokus pada pengelolaan riset dan teknologi yang menyangkut kedua produk dari sisi teknlogi dan user experience produk.

4. Direktorat Teknologi Informasi

   Fokus pada pengelolaan infrastructure strategy & governance, IT Strategy & Governance, and Solution serta pengelolaan pendayagunaan IT dan service operation & management.

5. Direktorat Pengembangan dan Sumber Daya

   Fokus pada manajemen SDM Perusahaan serta penyelenggaraan operasional SDM.

6. Direktorat Aset dan *Inventory*

   Fokus pada pengelolaan asset perusahaan dan inventory penggunaan.

2.2. *Continuous Integration* (CI)

CI adalah suatu metode dalam pengembangan perangkat lunak, dimana tiap-tiap *developer* mengintegrasikan hasil kerja mereka. Biasanya dilakukan tiap hari oleh masing-masing *developer*.

Secara umum, tahapan-tahapan dalam sebuah sistem CI adalah sebagai berikut(dipicu dengan perintah *push* atau *commit*):

1. *Integrated*

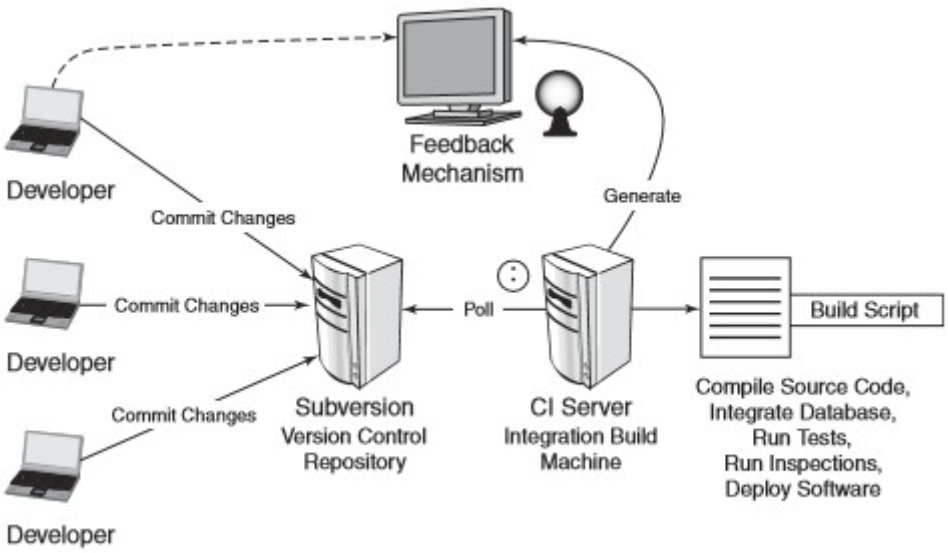
   Merupakan tahapan dimana tiap-tiap *developer* mengintegrasikan kode-kode atau hasil kerja mereka dalam satu tempat
2. *Built*

   Kode di*compile* kedalam *package* atau *executable*
3. *Tested*

   Tes otomatis dijalankan
4. *Archived*

   Mengarsipkan hasil *build* dan *tes*, termasuk *file* log selama proses tersebut berlangsung.
5. *Deployed*

   Memasang atau membagikan hasil *build*.



Gambar 2.2:Arsitektur sebuah sistem CI

2.3. Jenkins

Jenkins dikembangkan oleh Kohsuke Kawaguchi, yang memulai proyek sebagai hobi . Awalnya proyek ini dinamai dengan nama Hudson pada akhir 2004 ketika masih bekerja di Sun. Hudson berkembang selama beberapa tahun, sistem ini diadopsi oleh banyak tim dalam Sun untuk proyek-proyek mereka. Pada awal 2008, Sun mengakui kualitas dan nilai dari Hudson, dan meminta Kohsuke untuk bekerja pada Hudson penuh waktu, serta mulai memberikan layanan profesional dan mendukung Hudson. Pada tahun 2010, Hudson telah menjadi *tool* paling diminati dalam solusi *Continuous Integration* dengan pangsa pasar lebih dari 70%. Pada tahun 2009, Oracle membeli Sun.

Menjelang akhir 2010, ketegangan muncul antara pengembang Hudson masyarakat dan Oracle, awalnya dipicu oleh masalah dengan infrastruktur Java.net, dan diperparah oleh isu-isu yang berkaitan dengan klaim Oracle untuk merek dagang Hudson. Ketegangan ini juga tercermin kuat ketidaksepakatan yang mendasari tentang cara proyek ini dikelola oleh Oracle. Memang, Oracle ingin bergerak ke arah proses pembangunan lebih terkontrol ketat dengan jadwal rilis lambat, sedangkan sebagian besar dari *developer* inti Hudson, yang dipimpin oleh Kohsuke, lebih suka melanjutkan dengan mdel terbuka, fleksibel, dan cepat, berfokus pada masyarakat yang telah bekerja sangat baik bagi Hudson di masa lalu.

Pada bulan Januari 2011, komunitas pengembang Hudson memutuskan untuk mengubah nama proyek menjadi Jenkins. Mereka kemudian membuat salinan kode Hudson asli ke project GitHub baru dan meneruskan pekerjaan mereka di sana. Sebagian besar *developer* inti dan *plugin* beserta Kohsuke Kawaguchi dan kontributor inti bersama-sama mulai mengembangkan Jenkins, di mana sebagian besar kegiatan pembangunan dapat dilihat hari ini.

Hingga apda saat tahun 2011, jajak pendapat menunjukkan bahwa sekitar 75% dari pengguna Hudson telah beralih ke Jenkins, sementara 13% masih menggunakan Hudson, dan lain 12% menggunakan kedua Hudson dan Jenkins atau di proses migrasi ke Jenkins.

Namun demikian, Oracle dan Sonatype (perusahaan di balik Maven dan Nexus)

terus bekerja

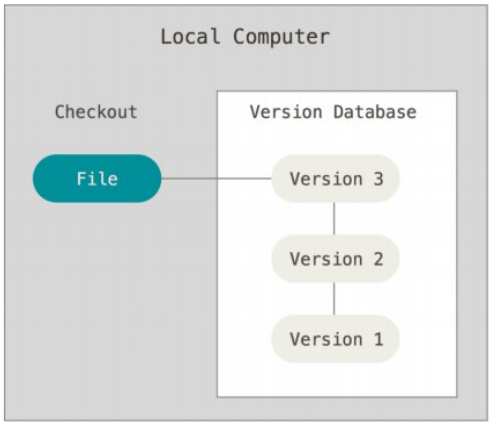
pada basis kode Hudson (https://github.com/hudson), tetapi dengan fokus yang berbeda.

2.4. *Version Control System* (VCS)

Version Control adalah sebuah sistem yang mencatat perubahan *file* dari waktu ke waktu. Sebagai contoh, seorang *web designer* akan menyimpan setiap versi dari rancangan yang dibuatnya. Dengn menggunakan VCS memungkinkan Anda untuk kembali ke berkas rancangan sebelumnya, membandingkan perubahan yang terjadi, melihat siapa yang telah membuat perubahan.

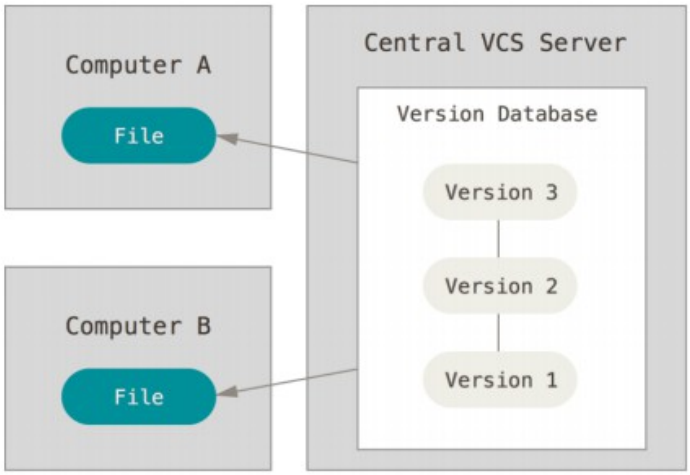
1.Local Version Control Systems

Ini adalah metode VCS yang paling sederhana, dimana setiap *file* yang dimodifikasi disimpan ke dalam *directory* yang berbeda. Metode ini rawan kesalahan, misal lupa nama *folder*, sehingga para *developer* terdahulu mengembangkan local VCS yang memiliki database sederhana untuk mencatat setiap perubahan pada *file*. Salah satu aplikasi local VCS yang terkenal dan masih tersedia hingga saat ini adalah RCS.



Gambar 2.3: *Local Version Control Systems*

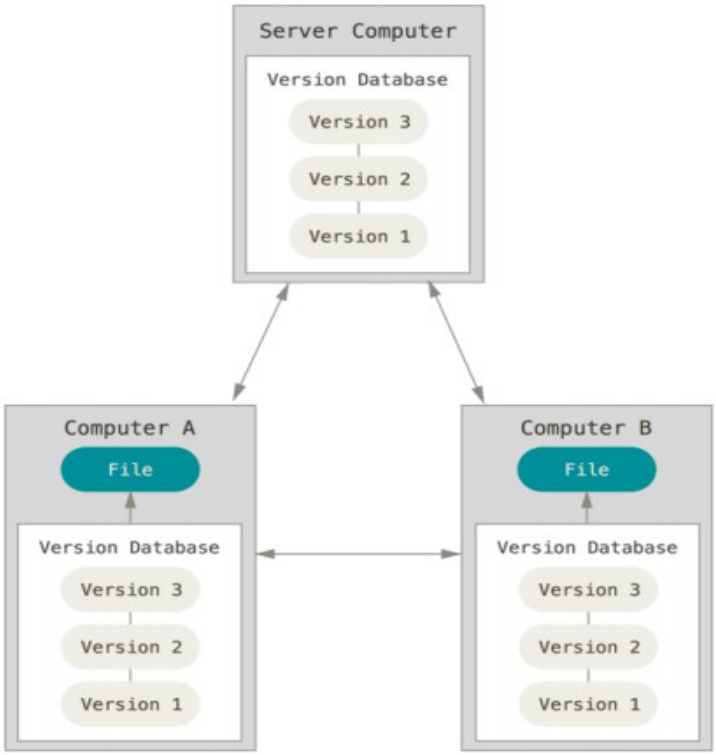
2.*Centralized Version Control Systems* (CVCS)



Gambar 2.4: *Centralized Version Control Systems (CVCS)*

Sistem ini dikembangkan karena adanya keterbatasan dalam sistem lokal CVS, yaitu tidak bisa berkolaborasi dengan *client* yang berada pada komputer yang berbeda. Sistem SCVCS seperti CVS, Subversion, dan Perforce, memiliki sebuah *server* yang memiliki *file* yang telah diberi versi. Sistem ini telah digunakan selama bertahun-tahun. Namun, sistem ini memiliki kelemahan, yaitu, jika *database*-nya rusak, maka pekerjaan Anda akan hilang, karena *client* lain tidak memiliki *cloning*-annya

3.*Distributed Version Control Systems* (DVCS)



Gambar 2.5: *Distributed Version Control Systems* (DVCS)

Menjawab masalah pada CVCS, maka dikembangkanlah DVCS. Dalam sistem ini, *client* tidak hanya melakukan *checkout* dari satu *server*, tetapi melakukan *clone*, menyalin seluruh *database* yang ada pada *server* pusat. Jadi, jika salah satu komputer *server* rusak, maka komputer lain masih memiliki *database* yang sama. Contoh DVCS adalah Git, Mercurial, Subversion, Bazaar atau Darc.

2.5. Unified Modeling Language (UML)

UML menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu:

1. *Use Case Diagram* untuk memodelkan proses bisnis.
2. *Conceptual Diagram* untuk memodelkan konsep-konsep yang ada di dalam aplikasi.
3. *Sequence Diagram* untuk memodelkan pengiriman pesan (*message*) antar

objects

4. *Collaboration Diagram* untuk memodelkan interaksi antar *objects*.
5. *State Diagram* untuk memodelkan perilaku *objects* di dalam sistem.
6. *Activity Diagram* untuk memodelkan perilaku Use Cases dan *objects* di dalam sistem.
7. *Class Diagram* untuk memodelkan struktur kelas.
8. *Object Diagram* untuk memodelkan struktur *object*.
9. *Component Diagram* untuk memodelkan komponen *object*.
10. *Deployment Diagram* untuk memodelkan distribusi aplikasi.

Berikut akan dijelaskan empat macam diagram yang paling sering digunakan dalam pembangunan aplikasi berorientasi *object*, yaitu *use case diagram, sequence diagram, collaboration diagram*, dan *class diagram.*

2.5.1. *Use Case Diagram*

Use case diagram digunakan untuk memodelkan bisnis proses berdasarkan perspektif pengguna sistem. *Use case diagram* terdiri atas diagram untuk *use case* dan actor. Actor merepresentasikan orang yang akan mengoperasikan atau orang yang berinteraksi dengan sistem aplikasi. *Use case* merepresentasikan operasi-operasi yang dilakukan oleh aktor. *Use case* digambarkan berbentuk elips dengan nama operasi dituliskan di dalamnnya. Actor yang melakukan operasi dihubungkan dengan garis lurus ke usecase.

2.5.2. Sequence Diagram

Sequence diagram menjelaskan secara detil urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari *use case*: interaksi yang terjadi antar class, operasi apa saja yang terlibat, urutan antar operasi, dan informasi yang diperlukan oleh masing-masing operasi.

2.5.3. Collaboration Diagram

Collaboration diagram dipakai untuk memodelkan interaksi antar objek di dalam sistem. Berbeda dengan sequence diagram yang lebih menonjolkan kronologis dari operasi-operasi yang dilakukan, collaboration diagram lebih fokus

pada pemahaman atas keseluruhan operasi yang dilakukan oleh objek.

2.5.4. *Class Diagram*

Class diagram merupakan diagram yang selalu ada di permodelan sistem berorientasi objek. Class diagram menunjukkan hubungan antar class dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan.

2.6. JavaScript *Object* Notation (JSON)

JSON (JavaScript *Object* Notation) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (generate) oleh komputer. Format ini dibuat berdasarkan bagian dari Bahasa Pemprograman JavaScript, Standar ECMA-262 Edisi ke-3 - Desember 1999. JSON merupakan format teks yang tidak bergantung pada bahasa pemprograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh programmer keluarga C termasuk C, C++, C#, Java, JavaScript, Perl, Python dll. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran-data.

JSON terbuat dari dua struktur:

- Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (record), struktur (struct), kamus (dictionary), tabel hash (hash table), daftar berkunci (keyed list), atau associative array.
- Daftar nilai terurutkan (an ordered list of values). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (array),

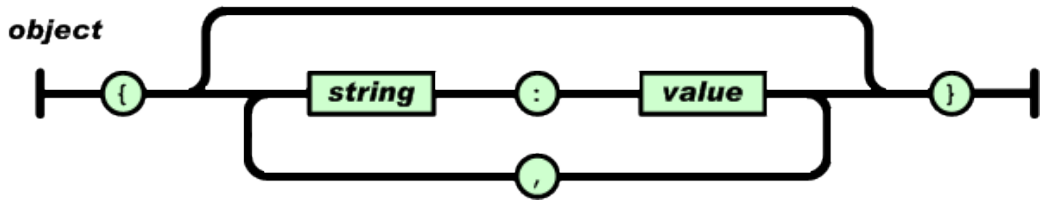
vektor (vector), daftar (list), atau urutan (sequence).

Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemprograman moderen mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemprograman yang juga berdasarkan pada struktur data ini.

JSON menggunakan bentuk sebagai berikut:

1. Objek (*object*)

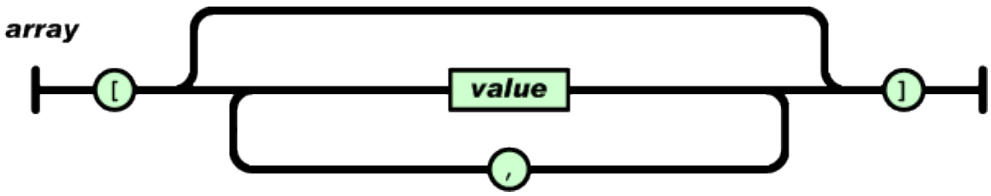
Objek adalah sepasang nama/nilai yang tidak terurutkan. Objek dimulai dengan { (kurung kurawal buka) dan diakhiri dengan } (kurung kurawal tutup). Setiap nama diikuti dengan : (titik dua) dan setiap pasangan nama/nilai dipisahkan oleh , (koma).



Gambar 2.6: Objek

2. Larik (array)

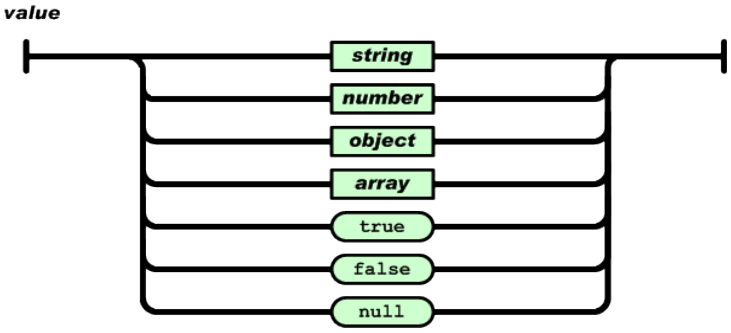
Larik adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [(kurung kotak buka) dan diakhiri dengan] (kurung kotak tutup). Setiap nilai dipisahkan oleh , (koma).



Gambar 2.7: Larik

3. Nilai (value)

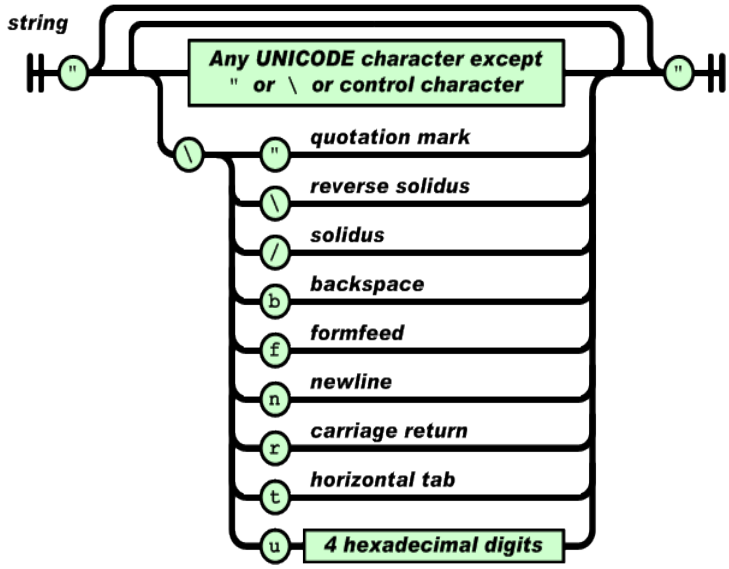
Nilai (value) dapat berupa sebuah string dalam tanda kutip ganda, atau angka, atau true atau false atau null, atau sebuah objek atau sebuah larik. Struktur-struktur tersebut dapat disusun bertingkat.



Gambar 2.8: Nilai

4. String

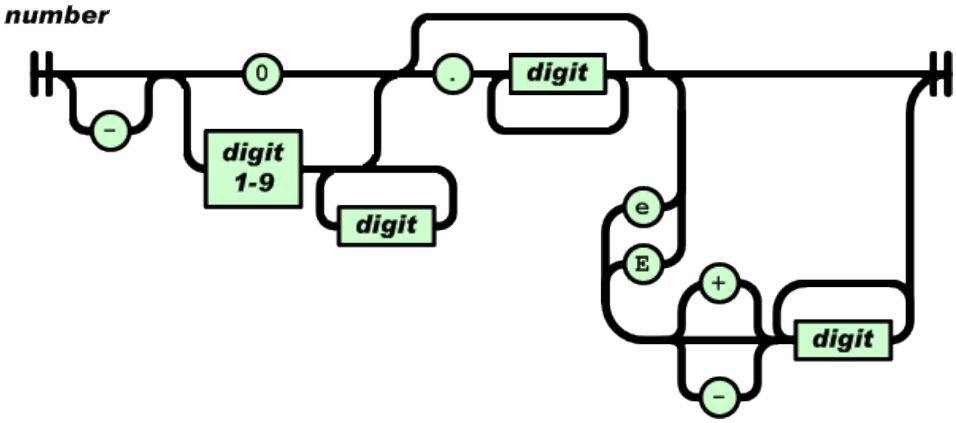
String adalah kumpulan dari nol atau lebih karakter Unicode, yang dibungkus dengan tanda kutip ganda. Di dalam string dapat digunakan backslash escapes "\" untuk membentuk karakter khusus. Sebuah karakter mewakili karakter tunggal pada string. String sangat mirip dengan string C atau Java.



Gambar 2.9: String

5. Angka

Angka adalah sangat mirip dengan angka di C atau Java, kecuali format oktal dan heksadesimal tidak digunakan.



Gambar 2.10: Angka

Spasi kosong (*whitespace*) dapat disisipkan di antara pasangan tanda-tanda tersebut, kecuali beberapa detil *encoding* yang secara lengkap dipaparkan oleh bahasa pemprograman yang bersangkutan.