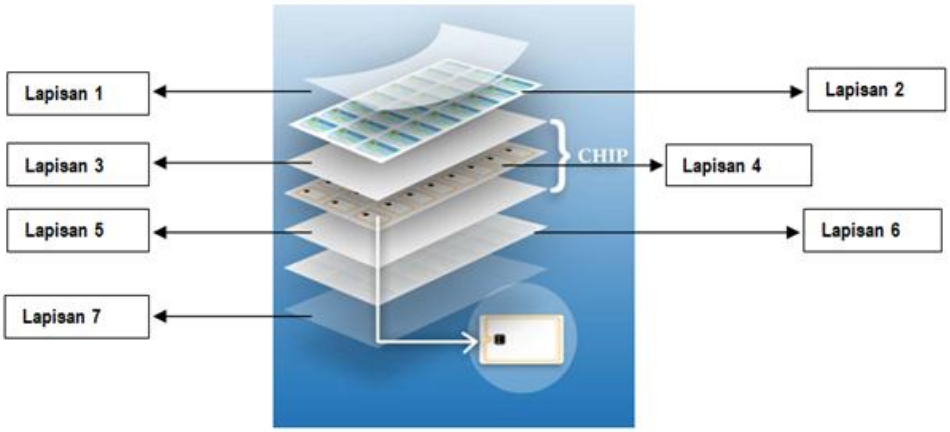
BAB II
LANDASAN TEORI

2.1. E-KTP

KTP (Kartu Tanda Penduduk) adalah bukti identitas masyarakat berkewarganegaraan Indonesia. Sedangkan e-KTP adalah KTP yang dielektronikkan, sehingga menjadi sebuah *smart-card* bertipe *contactless-card*, yaitu *chip smart-card* yang mampu berkomunikasi dengan *reader* tanpa kontak langsung secara fisik melainkan menggunakan gelombang radio dengan frekuensi 13,56MHz [5]. E-KTP dikembangkan sejak tahun 2011 dan hingga tahun 2014 dinyatakan memiliki potensi penyebaran hingga 175 juta orang atau sekitar 95% dari seluruh warganegara Indonesia yang wajib memiliki KTP.

Gambar 2.1. KTP Elektronik

Walaupun *chip* dalam e-KTP masih bagian dari keluarga RFID (*Radio Frequency Identification*) namun e-KTP tidak dapat dibaca oleh sembarang *reader* yang berada di pasaran karena dari awal dirancang dengan prosedural dan modul pengamanan tertentu, bekerjasama dengan Lembaga Sandi Negara. Secara umum pengamanan tersebut diberikan dalam kartu, dan juga dalam *reader*.



Gambar 2.2. Lapisan Blangko KTP Elektronik

E-KTP menggunakan kartu berjenis *Secure Microcontroller Integrated Circuit Chip Smart Card* dimana memiliki *microcontroller*, sistem operasi, dan *memory*. Layaknya sebuah mini-komputer, e-KTP dilengkapi *processor* untuk pengolahan data. Dengan *memory* berukuran EEPROM 8 Kbyte, e-KTP menyimpan informasi pemilik kartu yang meliputi NIK, nama, tempat dan tanggal lahir, jenis kelamin, golongan darah, alamat lengkap, agama, status perkawinan, pekerjaan, kewarganegaraan, tanggal berlaku, hingga pas foto, tanda tangan, dan dua sidik jari pemilik [5].

2.2. E-KTP *Reader*

E-KTP *Reader* atau pemindai e-KTP dibangun sesuai standar ISO 14443 A dan ISO 14443 B berada dalam koridor pengamanan tertentu, salah satunya dengan mekanisme autentifikasi dua arah di mana *chip* harus dapat mengenali *reader* dan *reader* harus dapat mengenali *chip*, sehingga tidak sembarang orang dapat membuatnya. Ada tiga jenis e-KTP *Reader* yang resmi dikeluarkan oleh Kemendagri, yaitu buatan Amerika Serikat, Korea, dan yang terbaru dari BPPT.

Karena e-KTP berjenis *contactless-card*, cara pembacaan katu yaitu dengan cara disentukan atau didekatkan kurang dari 10cm dengan *reader*. Bukan ditancapkan atau digesekkan.

Alur dasar penggunaan e-KTP *Reader* merupakan wewenang dan kebijakan Kemendagri. Namun, secara umum alur pembacaan e-KTP dapat diilustrasikan sebagai berikut.

a. Kartu e-KTP diletakkan di e-KTP *Reader*.
b. Setelah 10 detik akan ada indikasi e-KTP tersebut dapat dibaca atau tidak. Jika dapat dibaca,
c. Selanjutnya akan ada indikasi untuk meletakkan salah satu jari lain pada bidang *fingerprint-scanner*.
d. E-KTP *Reader* memverifikasi sidik jari selama 1-3 detik.
e. Jika verifikasi gagal, akan diminta sidik jari lain satu kali lagi.
f. Jika verifikasi berhasil, selanjutnya data dalam e-KTP akan tampil pada layar e-KTP *Reader* dan/atau dapat diunduh selama 11 detik.

Gambar 2.3. Perangkat e-KTP *Reader*

2.3. *Optical Mark Recognition*

Optical Mark Recognition (OMR) adalah metode pengolahan citra untuk mendapatkan data sebuah tanda markah dari sebuah gambar sebagai *input*-nya. OMR merupakan teknologi yang sudah usang namun penggunaannya masih sangat dibutuhkan. Kebutuhan teknologi berbasis OMR kali pertama datang dari sektor pendidikan [6]. Biasanya teknologi ini efektif digunakan untuk jumlah formulir di atas 500 lembar. Perekrutan CPNS sudah menggunakan teknologi OMR sejak 2004 dan sekarang dapat

digunakan untuk berbagai kepentingan seperti proses penelitian, survei, ujian, evaluasi, dan formulir pendaftaran.

Formulir yang digunakan biasa disebut LJK (Lembar Jawaban Komputer) dengan format yang biasanya sudah disesuaikan. Secara umum, cara pengisian form LJK dengan cara menghitamkan penuh suatu lingkaran atau hanya memberi tanda dalam bidang-bidang tertentu yang telah teratur.

Secara garis besar, algoritma OMR memproses tingkat kontras yang dipantulkan dari sebuah citra dokumen. Area dengan nilai cahaya rendah digunakan untuk mendeteksi area yang ditandai. Ini berarti kertas harus memiliki kontras yang besar agar dapat dikenali dengan mudah oleh aplikasi.

2.4. Metoda Pendekatan Berorientasi Objek

Dalam suatu proses pengembangan sistem informasi, analisis dan perancangan merupakan terminologi yang sudah sangat tua. Sebenarnya pada saat masalah ditelusuri dan spesifikasi dinegoisasikan, dapat dikatakan kita berada pada tahap perancangan. Merancang adalah menemukan suatu cara untuk menyelesaikan masalah, salah satu *tool*/model untuk merancang *software* yang berorientasi objek (*object oriented*) adalah UML.

2.4.1. Konsep Objek

Objek dalam "*software analysis & design*" adalah sesuatu yang berupa konsep, benda, dan sesuatu yang membedakannya dengan lingkungannya. Objek biasanya adalah kata benda, namun dalam konteks OOP objek bukan saja yang bisa dilihat atau diraba, tapi objek dapat pula merupakan sesuatu yang abstrak yang hidup di dalam sistem seperti *file*, tabel, *database*, *event*, *system messages*.

Objek dapat dikenali dari keadaannya dan juga operasinya, sebagai contoh sebuah mobil dikenali dari warnanya, bentuknya, sedangkan manusia dari suaranya, namanya, serta ciri-ciri yang akan membedakan objek tersebut dari objek lainnya.

Alasan mengapa pada saat ini memilih pendekatan berorientasi objek dalam pengembangan sistem informasi, pertama adalah *scalability* dimana objek lebih mudah dipakai untuk menggambarkan sistem yang besar dan komplek. Kedua, *dynamic modeling*, dimana objek dapat dipakai untuk pemodelan sistem dinamis dan *real time*.

2.4.2. Pengembangan Berorientasi Objek

Pengembangan berorientasi objek adalah suatu cara untuk pengembangan perangkat lunak dan sistem informasi berdasarkan abstraksi objek-objek yang ada di dunia nyata. (Brooks 1987, dikutip dari buku *Object Oriented Modeling And Design*, tulisan James Rumbaugh, dkk) mengemukakan bagian tersulit dari pengembangan perangkat lunak dan/atau sistem informasi adalah tahapan analisis dimana kita harus menganalisis masalah yang sangat rumit, yang kita jumpai di dunia nyata serta melakukan abstraksi terhadap masalah itu, kemudian melakukan perancangan agar kelak dapat diimplementasikan dengan cepat serta akurat pada komputer (baik secara perangkat keras maupun perangkat lunak) menjadi suatu perangkat lunak yang sesuai kebutuhan serta harapan pengguna.

2.4.3. Teknik Dasar OOA/D (*Object-Oriented Analysis/Design*)

Dalam dunia pemodelan, metodologi berorientasi objek walaupun terikat kaidah-kaidah standar, namun teknik pemilihan objek tidak terlepas pada subyektifitas *software analyst* dan *designer*. Beberapa objek akan diabaikan dan beberapa objek menjadi perhatian untuk diimplementasikan di dalam sistem, hal ini sah-sah saja karena kenyataan bahwa suatu permasalahan sudah tentu memiliki lebih dari satu solusi.

Ada 3 teknik/konsep dasar dalam OOA/D, yaitu pemodulan (*encapsulation*), penurunan (*inheritance*) dan *polymorphism*.

a. *Encapsulation*

Encapsulation (pengkapsulan) merupakan dasar untuk pembatasan ruang lingkup program terhadap data yang diproses.

b. *Inheritance*

Inheritance (pewarisan) adalah teknik yang menyatakan bahwa anak (turunan) dari objek akan mewarisi data/atribut dan metoda dari induknya langsung.

c. *Polymorfhism*

Polimorfisme adalah konsep yang menyatakan bahwa sesuatu yang sama dapat mempunyai bentuk dan perilaku yang berbeda.

2.4.4. Pengenalan UML

UML (*Unified Modeling Language*) adalah sebuah bahasa yang berdasarkan grafik/gambar untuk memvisualisasi, menspesifikasi, membangun, dan mendokumentasikan dari sebuah pengembangan sistem berorientasi objek. UML sendiri juga memberikan standar penulisan sebuah *system blue print*, yang meliputi konsep proses bisnis, penulisan kelas-kelas dalam bahasa program yang spesifik, skema *database*, dan komponen-komponen yang diperlukan dalam sistem *software*.

UML merupakan bahasa pemodelan yang memiliki pembendaharaan kata dan cara untuk mempresentasikan secara fokus pada konseptual dan fisik dari suatu sistem. Contoh untuk sistem software yang *intensive* membutuhkan bahasa yang menunjukan pandangan yang berbeda dari arsitektur sistem, ini sama seperti mengembangkan *software develpoment life cycle*. Dengan UML akan memberitahukan kita bagaimana untuk membuat dan membaca bentuk model yang baik, tetapi UML tidak dapat memberitahukan model apa yang akan dibangun dan kapan akan membangun model tersebut. Ini merupakan aturan dalam *software development process.*

UML tidak hanya merupakan sebuah bahasa pemograman *visual* saja, namun juga dapat secara langsung dihubungkan ke berbagai bahasa pemograman, seperti JAVA, C++, *Visual Basic*, atau bahkan dihubungkan secara langsung ke dalam sebuah *object-oriented database*. Begitu juga mengenai pendokumentasian dapat dilakukan seperti *requirements, arsitektur*, *design*, *source code*, *project plan*, *tests*, dan *prototypes.*

Dalam kerangka spesifikasi, UML menyediakan model-model yang tidak ambigu, serta lengkap. Secara khusus, UML menspeifikasi langkah-langkah penting dalam pengambilan keputusan analisis, perancangan, serta implementasi dalam sistem yang sangat bernuansa perangkat lunak (*software intensive system*).

2.4.5. Bagian-Bagian dari UML

Bagian-bagian utama dari UML adalah *view*, *diagaram*, *model element*, dan *general mechanism*.

a. *View*

View digunakan untuk melihat sistem yang dimodelkan dari beberapa aspek yang berbeda. *View* bukan melihat grafik, tapi merupakan suatu abstraksi yang berisi sejumlah diagram. Beberapa jenis *view* dalam UML antara lain *use case view*, *logical view*, *component view*, *concurrences view*, dan *deployment view*.

1) *Use case view*

Mendeskripsikan fungsionalitas sistem yang seharusnya dilakukan sesuai dengan yang diinginkan *external actors*. *Actor* yang berinterakasi dengan sistem dapat berupa *user* atau sistem lainnya. *View* ini digambarkan dalam *use case diagram* dan kadang–kadang dengan *activity diagrams*. *View* ini digunakan terutama untuk pelanggan, perancang (*designer*), pengembang (*developer*), dan penguji sistem (*tester*).

2) *Logical view*

Mendeskripsikan bagaimana fungsionalitas dari sistem, struktur statis (*class*, *object*, dan *relationship*) dan kolaborasi dinamis yang terjadi ketika *object* mengirim pesan ke *object* lain dalam suatu fungsi tertentu. *View* ini digambarkan dalam *class diagram* untuk struktur statis dan dalam *state sequence*, *collaboration*, dan *activity diagrams* untuk model dinamisnya. *View* ini digunakan untuk perancang (*designer*) dan pengembang (*developer*).

3) *Component view*

Mendiskripsikan implementasi dan ketergantungan modul. Komponen yang merupakan tipe lainnya dari *code module* diperlihatkan dengan struktur dan ketergantungannya juga alokasi sumber daya komponen dan informasi *administrative* lainnya. *View* ini digambarkan dalam *component view* dan digunakan untuk pengembang (*developer*).

4) *Concurrency view*

Membagi sistem ke dalam proses dan *processor*. *View* ini digambarkan dalam diagram dinamis (*state*, *sequence*, *collaboration*, dan *activity diagram*) dan diagram implementasi (*component* dan *deployment diagrams*) serta digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

5) *Deployment view*

Mendeskripsikan fisik dari sistem seperti komputer dan perangkat lain serta bagaimana hubungan dengan yang lainnya. *View* ini digambarkan dalam *deplovment diagram* dan digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

b. Diagram

Setiap sistem yang kompleks seharusnya bisa dipandang dari sudut yang berbeda-beda sehingga kita bisa mendapatkan pemahaman secara menyeluruh. Untuk upaya tersebut UML menyediakan setidaknya sembilan jenis diagram. Diagram-diagram ini tidak mutlak harus digunakan semuanya, hanya dibuat sesuai dengan kebutuhan. Pada pemodelan dengan UML juga dimungkinkan menggunakan diagram-diagram lain sejauh itu memang diperlukan untuk mendapatkan pemahaman mendalam tentang suatu sistem atau suatu perangkat lunak. Diagram-diagram yang disediakan UML antara lain sebagai berikut.

1) *Class diagram*

Diagram ini memperlihatkan himpunan kelas, antarmuka, kolaborasi serta relasi. Diagram ini umumnya dijumpai pada pemodelan berorientasi objek. Meskipun bersifat statis sering pula diagram kelas memuat kelas-kelas aktif.

2) *Object diagram*

Diagram ini memperlihatkan objek-objek serta relasi antar objek. Diagram objek memperlihatkan istansiasi statis dari segala sesuatu yang dijumpai pada diagram kelas.

3) *Use case diagram*

Diagram ini memperlihatkan himpunan *Use Case* dan aktor-aktor. Diagram ini sangat penting terutama untuk mengorganisasi dan memodelkan perilaku-perilaku dari suatu sistem yang dibutuhkan serta diharapkan pengguna.

4) *Sequence diagram*

Diagram ini berfungsi sebagai interaksi yang menekankan pada pengiriman pesan (*message*) dalam suatu waktu tertentu.

5) *Collaboration diagram*

Diagram interaksi yang menekankan organisasi struktural dari objek-objek yang menerima serta mengirim pesan (*message*).

6) *State diagram*

Menggambarkan semua *state* pada sistem, memuat *state*, transisi, *event* serta aktifitas. Diagram ini penting untuk memperlihatkan sifat dinamis dari antarmuka, kelas serta kolaborasi terutama pada pemodelan sistem-sistem yang reaktif.

7) *Aktivity diagram*

Diagram ini adalah tipe khusus dari diagram *state* yang memperlihatkan aliran dari suatu aktifitas ke aktifitas lainnya dalam suatu sistem. Diagram ini penting bagi pemodelan fungsi-fungsi dalam suatu sistem dan memberikan tekanan pada aliran kendali antar objek.

8) *Component diagram*

Diagram ini memperlihatkan organisasi serta ketergantungannya pada komponen-komponen yang telah ada sebelumnya. Diagram ini berhubungan dengan diagram kelas dimana komponen secara tipikal dipetakan ke dalam satu atau lebih kelas, antarmuka serta kolaborasi.

9) *Deployment diagram*

Diagram ini memperlihatkan konfigurasi saat aplikasi dijalankan (*runtime*). Diagram ini memuat simpul-simpul (*node*) beserta komponen-komponen yang ada di dalamnya. *Deployment diagram* berhubungan erat dengan *component diagram* dimana *deployment diagram* memuat satu atau lebih komponen. Diagram ini berguna saat aplikasi akan diterapkan pada banyak komputer (*distributed computing*).

2.5. Perangkat Lunak Pendukung

Dalam pengembangan suatu sistem informasi, diperlukan perangkat lunak lain yang bisa digunakan untuk membangun suatu aplikasi sesuai dengan keinginan user. Berikut adalah beberapa perangkat lunak yang digunakan dalam pembangunan Sistem Pendaftaran Pasien.

2.5.1. Visual Basic .NET

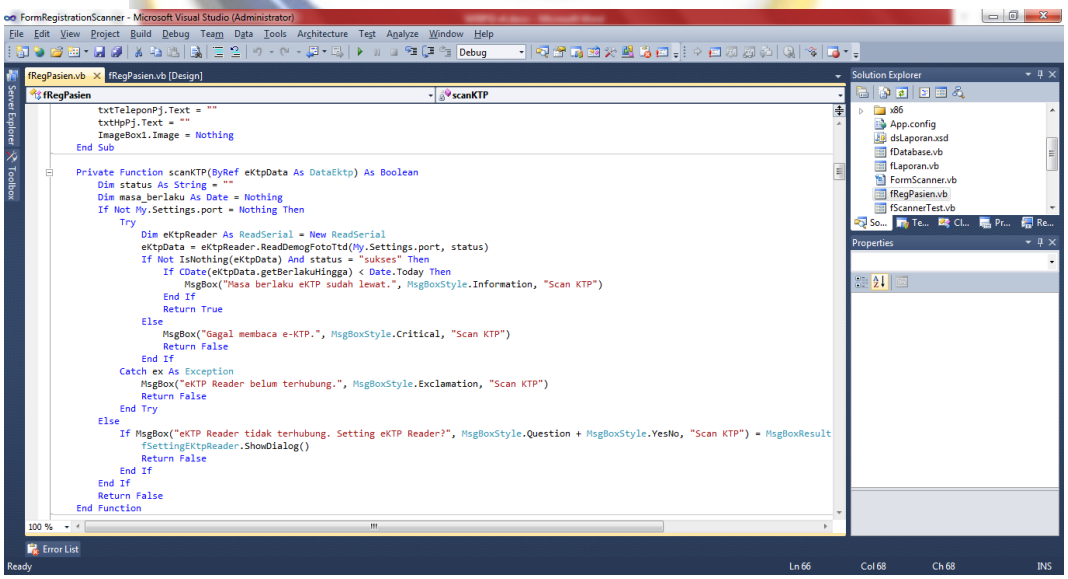
Sekarang ini Microsoft telah dapat merealisasikan visi "Sistem operasi Windows dalam setiap PC dan PC dalam setiap desktop". Salah satu *tool* untuk mengembangkan aplikasi .NET adalah Microsoft Visual Basic .NET (VB .NET). VB .NET bersama dengan Visual C++ .NET, Visual C# .NET, Visual J++ .NET dan Visual J# .NET merupakan bagian dari Microsoft Visual Studio .NET.

VB .NET adalah bahasa pemrograman untuk membuat aplikasi berbasis Windows, aplikasi *form* Web ASP .NET, layanan Web XML dan aplikasi mobile seperti komputer *Palm* dan *Pocket* PC. VB .NET dibangun di atas fondasi .NET *Framework*.

Setiap generasi baru dari perangkat lunak bahasa pemrograman datang karena adanya keterbatasan dari generasi sebelumnya. Teknologi *device*, *hardware*, *network* dan internet baru yang muncul menyebabkan bahasa pemrograman yang ada tidak lagi menjadi alat yang ideal untuk mengembangkan perangkat lunak yang dapat bekerja dengan teknologi baru tersebut.

Sekarang untuk pertama kalinya, platform pengembang perangkat lunak yang lengkap, Microsoft .NET telah didesain dari dasar dengan internet sebagai fokus utamanya walaupun tidak secara eksklusif dan hanya untuk pengembang internet. Banyak inovasi baru yang berada dalam *platform* ini akan mengatasi keterbatasan dari *tools* dan teknologi lama.

Visual Basic .NET mempunyai lingkungan pengembangan yang terintegrasi atau sering disebut dengan IDE (*Integrated Developmnet Environment*) dengan lingkungan kerja yang luas. Area kerja adalah jendela yang berguna untuk melakukan kegiatan pengisian kode (*coding*) ketika berupa tampilan kode (*code view*) dan untuk mengatur tampilan desain ketika berupa tampilan desainer (*desaign view*).

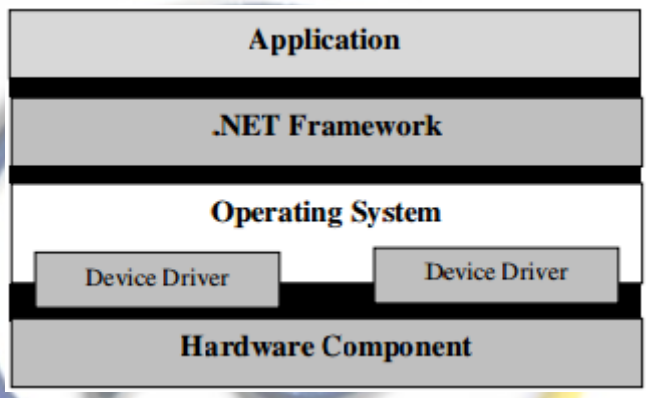


Gambar 2.4. IDE Visual Basic .NET 2010

2.5.2. .NET *Framework*

Dalam pengembangan aplikasi Visual Basic .NET ada banyak perubahan yang cukup radikal, salah satunya adalah pengembangan dasar bagi semua alat-alat pengembangan .NET. Dasar ini disebut .NET *Framework*, yang menyediakan dua hal yaitu *Common Language Runtime* (CLR) dan *class library*.

CLR mirip dengan sistem operasi, menyediakan lapisan antara program dengan kompleksitas dari sistem, menampilkan pelayanan dari aplikasi dan penyederhanaan akses pada fungsi lapisan paling bawah.



Gambar 2.5. Lapisan .NET *Framework*

Class library menyediakan satu set besar fungsi *wrapping* dan abstraksi seperti teknologi *internet protocol*, akses *file system*, manipulasi XML dan masih banyak lagi.

Dengan demikian .NET *Framework* didesain untuk memenuhi tujuan berikut.

a. Menyediakan lingkungan pemrograman berorientasi objek yang konsisten meskipun kode objek disimpan dan dijalankan secara lokal, tetapi disebarkan melalui internet atau dijalankan secara *remote*.
b. Menyediakan lingkungan yang menjalankan kode dengan meminimalkan konflik saat *software* disebarkan.
c. Menyediakan lingkungan yang menjamin keamanan saat kode dijalankan, termasuk kode yang dijalankan oleh pihak ketiga yang tidak diketahui atau kurang dipercaya.

d. Menyediakan lingkungan untuk menjalankan kode yang dapat menghilangkan masalah performa dari lingkungan *scripted* dan *interpreted*.

e. Membangun komunikasi standar industri guna memastikan kode berbasis .NET *Framework* dapat disatukan dengan kode-kode lain.