# BAB II LANDASAN TEORI

## 2.1 Sistem Informasi

Sistem informasi adalah sebuah gabungan dari orang, perangkat keras, perangkat lunak, dan komunikasi jaringan yang terorganisir, yang mengumpulkan sumber daya data, mengubah, dan menyebarkan informasi ke dalam sebuah organisasi (O’Brien 1999).

## 2.2 Basis Data

Basis data adalah sebuah kumpulan bagian dari data yang berhubungan secara logis (dan sebuah gambaran dari data), dan dirancang untuk memenuhi kebutuhan informasi pada sebuah organisasi (Connolly dan Begg 1995).

## 2.3 Database Management System

*Database Management System* (DBMS) adalah sebuah sistem perangkat lunak yang membolehkan pengguna untuk mendefinisikan, membuat, dan memelihara suatu basis data, serta menyediakan akses kontrol untuk basis data tersebut (Connolly dan Begg 1995).

## 2.4 Kamus Data

Kamus data adalah suatu ensiklopedi dari informasi mengenai tiap elemen data. Kamus data dapat berupa kertas atau *file* komputer. Jika berupa *file,* perangkat lunak khusus diperlukan untuk menciptakan dan memeliharanya, serta mempersiapkanya untuk digunakan. Perangkat lunak tersebut disebut sistem kamus data *(data dictionary system),* atau DDS. DDS dapat diperoleh sebagai paket perangkat lunak terpisah atau sebagai modul-modul di dalam sistem seperti DBMS dan peralatan *computer-aided software engineering* (CASE) (McLeod 1996).

## 2.5 Entity Relationship Diagram

Diagram hubungan entitas *(Entity Relationship Diagram)* atau ERD, mendokumentasikan data organisasi dengan mengidentifikasi jenis entitas dan hubungannya. ERD disiapkan pada suatu titik dalam proses pengembangan sistem saat “gambaran besar” data ditemukan (McLeod 1996). Simbol-simbol ERD ditampilkan pada Tabel 1.

Tabel 1 Simbol-simbol ERD

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Nama** | **Simbol** | **Keterangan** |
| 1 | Entitas |  | Jenis entitas (entitas *type*) dapat berupa suatu elemen lingkungan, sumber daya, atau transaksi. |
| 2 | Hubungan |  | Hubungan *(relationship)* adalah suatu asosiasi yang ada antara dua jenis entitas. Hubungan digambarkan dengan bentuk belah ketupat. |
| 3 | Keterkaitan |  | Banyaknya suatu entitas berhubungan dengan entitas lain disebut keterkaitan. Ada tiga keterkaitan yaitu: keterkaitan satu ke satu, keterkaitan satu ke banyak, dan keterkaitan banyak ke banyak. |
| 4 | Kardinalitas |  | Menentukan jumlah *instance* dari satu entitas yang dapat (atau harus) dihubungkan dengan *instance* di entitas lain. |

## 2.6 Hypertext Markup Language

*Hypertext Markup Language* disingkat HTML*.* Istilah *hypertext* mengacu pada *cross-link*, juga disebut *hyperlink* antara halaman-halaman *web*. Istilah *markup language* mengacu pada perintah yang memformat halaman-halaman *web* (Perry 2000).

## 2.7 PHP Hypertext Preprocessor

PHP merupakan singkatan dari *PHP Hypertext Preprocessor.* PHP merupakan bahasa berbentuk skrip yang ditempatkan dalam *server* dan diproses di *server.* Hasilnya yang dikirimkan ke klien, tempat pemakai menggunakan *browser*.

Secara khusus, PHP dirancang untuk membentuk aplikasi *web* dinamis. Artinya, PHP dapat membentuk suatu tampilan berdasarkan permintaan terkini. Misalnya, bisa menampilkan *database* ke halaman *web* (Kadir 2008).

## 2.8 Cascading Style Sheet

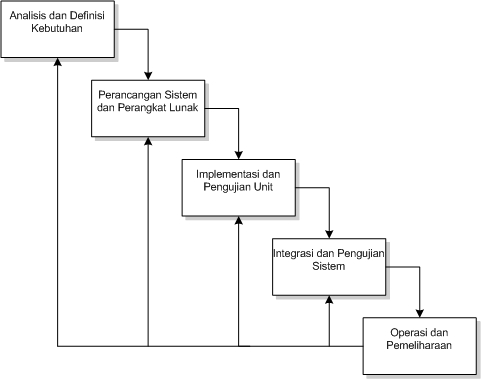
CSS singkatan dari *Cascading Style Sheet*. Penggunaan CSS dilakukan untuk memperluas kemampuan HTML dalam memformat dokumen *web* atau untuk mempercantik tampilan *web*. Penulisan kode CSS dapat langsung pada dokumen HTML atau disimpan dalam dokumen tersendiri kemudian dipanggil untuk digunakan (Prihatna 2005).

## 2.9 Struktur Navigasi

Struktur navigasi merupakan alur yang digunakan dalam perancangan halaman. Sebelum membangun website terlebih dahulu kita menentukan struktur navigasinya agar penyampaian informasi dapat lebih mudah dan terarah. Selain itu, struktur navigasi juga menggambarkan hubungan antara halaman yang satu dengan yang lain. Struktur navigasi yang ada di website ini terdiri dari dua bagian yaitu, struktur navigasi user dan struktur navigasi admin.

## 2.10 Model Pengembangan Perangkat Lunak

Metode *Waterfall* merupakan model proses pengembangan perangkat lunak yang pertama kali dipublikasikan dari model lainnya dan membagi tahapan utamanya menjadi lima tahapan seperti ditunjukkan pada Gambar 1. Model ini dinamakan *Waterfall* (air terjun) karena penurunan dari satu fase ke fase yang lainnya. Tahap-tahap utama dari model ini memetakan kegiatan-kegiatan pengembangan dasar, yaitu (Sommerville 2001):



Gambar 1 Metode *Waterfall*.

1. Analisis dan definisi kebutuhan

Pelayanan, batasan, dan tujuan sistem ditentukan melalui konsultasi dengan *user system*. Persyaratan ini kemudian didefinisikan secara rinci dan berfungsi sebagai spesifikasi sistem.

1. Perancangan sistem dan perangkat lunak

Proses perancangan sistem membagi persyaratan dalam sistem perangkat keras atau perangkat lunak. Kegiatan ini menentukan arsitektur sistem secara keseluruhan. Perancangan perangkat lunak melibatkan identifikasi dan deskripsi abstraksi sistem perangkat lunak yang mendasar dan hubungan-hubungannya.

1. Implementasi dan pengujian unit

Pada tahap ini, perancangan perangkat lunak direalisasikan sebagai serangkaian program atau unit program. Pengujian unit melibatkan verifikasi bahwa setiap unit telah memenuhi spesifikasinya.

1. Integrasi dan pengujian sistem

Unit program atau program individual diintegrasikan dan diuji sebagai sistem yang lengkap untuk menjamin bahwa persyaratan sistem telah dipenuhi. Setelah pengujian sistem, perangkat lunak dikirim kepada pelanggan.

1. Operasi dan pemeliharaan

Biasanya (walaupun tidak seharusnya), ini merupakan fase siklus hidup yang paling lama. Sistem diinstal dan dipakai. Pemeliharaan mencakup koreksi dari berbagai *error* yang tidak ditemukan pada tahap-tahap terdahulu, perbaikan atas implementasi unit sistem dan pengembangan pelayanan sistem, sementara persyaratan-persyaratan baru ditambahkan.

## 2.11 Object-oriented

*Object oriented* dibangun di atas beberapa prinsip dasar. Obyek adalah contoh/*instance* dari sebuah *class*. Beberapa obyek yang mempunyai atribut dan *operation* yang sama akan membentuk *class* (Munawar 2005).

Ada tiga konsep dasar dalam *Object-Oriented*, yaitu pemodulan (*encapsulation*), pewarisan (*inheritance*), dan *polymorphism*.

1. *Encapsulation*

*Encapsulation* sering juga dinyatakan sebagai penyembunyian informasi. *Encapsulation* memungkinkan seseorang untuk melakukan sesuatu tanpa perlu mengetahui secara mendetail tentang bagaimana sesuatu tersebut dilakukan.

2. *Inheritance*

*Inheritance* adalah pewarisan struktur dan *behavior* dari *superclass* ke *subclass*. Gambar 2menunjukkan bahwa *Class SavingAccount* dan *Class CurrentAccount* mewarisi operasi-operasi *Class Account* seperti *Deposit*(), *Withdraw*() dan *CreateStatement*().



()

Gambar 2 Contoh pewarisan (*Inheritance*).

3. *Polymorphism*

*Polymorphism* adalah penggunaan *operation* yang berbeda dengan nama *operation* yang sama. Sebagai contoh, *operation* ‘melihat’ bisa dipakai untuk melihat langit, melihat rumah, melihat teman, dan lain-lain. Meski terdengar sama, sebenarnya yang dilakukan berbeda.

## 2.12 Unifield Modeling Language

*Unifield Modeling Language* (UML) adalah salah satu alat bantu yang sangat handal di dunia pengembangan yang berorientasi objek. Hal ini disebabkan karena UML menyediakan bahasa pemodelan *visual* yang memungkinkan bagi pengembang sistem untuk membuat cetak biru atas visi mereka dalam bentuk baku, mudah dimengerti serta dilengkapi dengan mekanisme yang efektif untuk berbagi (*sharing*) dan mengkomunikasikan rancangan mereka dengan yang lain.

UML merupakan kesatuan dari bahasa pemodelan yang dikembangkan oleh Booch, *Object Modeling Technique* (OMT) dan *Object Oriented Software Engineering* (OOSE). Dengan UML, metode Booch, OMT, dan OOSE digabungkan dengan membuang elemen-elemen yang tidak praktis ditambah dengan elemen-elemen dari metode lain yang lebih efektif dan elemen-elemen baru yang belum ada pada metode terdahulu sehingga UML lebih ekspresif dan seragam daripada metode lainnya (Munawar, 2005).

1. *Use Case Diagram*

*Use case* adalah deskripsi fungsi dari sebuah sistem dari perspektif pengguna. *Use case* bekerja dengan cara mendeskripsikan tipikal interaksi antara *user* (pengguna) sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai.

*Use Case Diagram* menunjukkan 3 aspek dari sistem yaitu: *actor*, *use case,* dan sistem/*sub sistem boundary*. *Actor* mewakili peran orang, sistem yang lain atau alat ketika berkomunikasi dengan *use case*. Gambar 3 mengilustrasikan *actor*, *use case,* dan *boundary*.



Gambar 3 *Use Case* *Model*.

2. *Class Diagram*

*Class Diagram* sangat membantu dalam visualisasi struktur kelas dari suatu sistem. Hal ini disebabkan karena *class* adalah deskripsi kelompok objek-objek dengan *property*, perilaku (operasi) dan relasi yang sama. Di samping itu *Class Diagram* dapat memberikan pandangan global atas sebuah sistem. Hal ini tercermin dari *class-class* yang ada dan relasinya satu sama lain. Adapun simbol-simbol yang biasa digunakan pada *Class Diagram* dapat dilihat pada Tabel 2.

Tabel 2 Simbol-simbol pada C*lass Diagram*.

|  |  |
| --- | --- |
| **Simbol** | **Keterangan** |
| NewClass | Class |
|  | Unidirectional association |
|  | Generalization |
|  | Aggregation |

3. Activity Diagram

*Activity Diagram* adalah teknik untuk mendeskripsikan logika prosedural, proses bisnis, dan aliran kerja dalam banyak kasus. *Activity Diagram* mempunyai peran seperti halnya *flowchart,* akan tetapi perbedaannya dengan *flowchart* adalah *activity diagram* bisa mendukung perilaku paralel sedangkan *flowchart* tidak bisa.

Simbol-simbol yang sering digunakan dalam pemodelan *activity diagram* dapat dilihat pada Tabel 3.

*Tabel* 3 *Simbol-simbol pada Activity D*iagram.

|  |  |
| --- | --- |
| **Simbol** | **Keterangan** |
|  | Titik awal |
|  | Titik akhir |
|  | *Activity* |
|  | Pilihan untuk pengambilan keputusan |
|  | *Fork,* digunakan untuk menunjukkan kegiatan yang dilakukan secara paralel atau untuk menggabungkan dua kegiatan paralel menjadi satu. |

4. *Component Diagram*

*Component Diagram* menggambarkan alokasi semua *class* danobjek ke dalam komponen-komponen fisik di sebuah sistem *software*. Diagram ini memperlihatkan pengaturan dan ketergantungan di antara komponen-komponen *software* seperti *Dynamic Library Link* (DLL), *executable component,* dan lain-lain. Adapun simbol-simbol yang biasa digunakan pada *component diagram* dapat dilihat pada Tabel 4.

*Tabel* 4 *Simbol-simbol pada Component D*iagram.

|  |  |
| --- | --- |
| **Simbol** | **Keterangan** |
|  | *Component* |
|  | *Dependency* |

5. *Deployment Diagram*

*Deployment Diagram* menunjukkan tata letak sebuah sistem secara fisik, menampakkan bagian-bagian *software* yang berjalan pada bagian *Deployment Diagram* dapat dilihat pada Tabel 5.

Tabel 5 Simbol-simbol pada *Deployment Diagram*.

|  |  |
| --- | --- |
| **Simbol** | **Keterangan** |
|  | *Device* |
|  | *Connection* |

## 2.13 Metode Pengujian Black box

Pengujian *black box* berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengujian *black box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi *input* yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. Pengujian *black box* bukan merupakan alternatif dari teknik *white box,* tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kelas kesalahan daripada metode *white box.*

Pengujian *black box* berusaha menemukan kesalahan dalam kategori sebagai berikut: (1) fungsi-fungsi yang tidak benar atau hilang, (2) kesalahan *interface*, (3) kesalahan dalam struktur data atau akses basis data eksternal, (4) kesalahan kinerja, (5) inisialisasi dan kesalahan terminasi (Pressman 2002).

## 2.14 *Framework*

Kerangka Kerja Menurut Pratama (2010, hal. 10) *Framework* adalah rangka atau kerangka, arti istilah tersebut dalam dunia pemrograman adalah kumpulan kelas (class) dan fungsi (function, method) yang disusun secara sistematis berdasarkan kegunaan atau fungsi tertentu untuk mempermudah pembuatan atau pengembangan suatu aplikasi. Pratama (2010, hal. 10) juga menjelaskan bahwa sebagian besar *framework* yang beredar saat ini dibangun berdasarkan konsep Object-Oriented Programming. Selain itu banyak manfaat yang didapat saat menggunakan framework. *Framework* menawarkan penghematan waktu kerja dalam penulisan kode dan pengaturan berkas-berkas kode. Programmer tidak perlu susah payah menulis kode dari awal untuk fungsi-fungsi yang sudah disediakan. Selain itu berkas kode akan tersusun secara sistematis sesuai dengan struktur yang ditawarkan framework, dengan demikian akan memberikan kemudahan saat satu softwareharus dikerjakan oleh banyak orang.

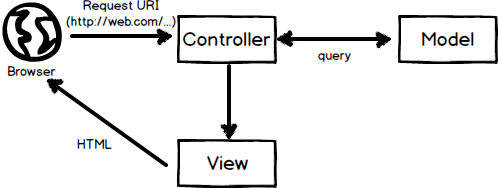
Dari pengertian di atas maka dapat disimpulkan bahwa *framework* merupakan kerangka yang berisi kumpulan *class* dan *function* yang disusun secara sistematis dengan tujuan tertentu agar mempermudah dalam pengembangan suatu aplikasi baik secara individu ataupun kelompok.

## 2.15 **MVC (*Model View Controller*)**

Menurut penjelasan Pratama (2010, hal. 11):

“arsitektur (atau kadang disebut dengan pola) MVCm uncul sejak tahun 1970 atas pemikiran Prof. Trygve Reenskaug, seorang berkebangsaan Norwegia. Dasar arsitektur ini adalah pemisahan antara logika aplikasi dengan tampilan. Dengan menggunakan pola ini diharapkan dapat meminimalisasi penulisan perintah, sehingga risiko terjadinya bug juga minimal, serta meningkatkan efisiensi pembangunan suatu aplikasi”.

Penjelasan mengenai arsitektur MVC adalah seperti pada Gambar 4.

Gambar 4 *Architecture MVC*.

Pratama (2010, hal. 12) menjelaskan fungsi dari masing-masing bagianadalah sebagai berikut ini:

a. Model.

Bertanggung jawab untuk melakukan pengelolaan data dalam basis data, di dalamnya biasa dituliskan perintah untuk mengambil, mengubah, menghapus, dan menambahkan data.

b. View.

Merupakan tempat untuk meletakkan apa yang akan ditampilkan di halaman perambah (browser), sebuah berkas view umumnya berisi kode bahasa pemrograman sisi klien (client-side scripting).

c. Controller.

Merupakan pengatur utama hubungan antara model, view,dan juga sumber daya lain yang tersedia, sumber daya ini diperoleh dari kelompok/tipe kelas yang dapat disebut dengan elemen framework CI. Dari seluruh penjelasan di atas makadapat diambil kesimpulan bahwa MVC adalah dasar arsitektur pemrograman di mana terdapat pemisahanantara logika, basis data, dan tampilan guna meminimalisasi kesalahan.

## 2.16 (CI) *CodeIgniter*

Menurut Elislab (2013), *Codeigniter* merupakan :

*Framework* untuk membangun aplikasi web berbasis PHP. *Codeigniter* menyediakan banyak *library* untuk fungsi-fungsi umum, antar muka yang sederhana, dan struktur yang logis. Programmer dapat membuat aplikasi dengan lebih cepat karena tidak perlu menulis kode dari awal, selain itu Codeigniter juga menyediakan banyak fungsi yang siap digunakan. Seorang programmer bisa lebih fokus dengan aplikasi yang sedang dibangun dan meminimalkan penulisan kode.

a. Kelebihan *Codeigniter*

Elislab (2013) menjelaskan *Codeigniter* memiliki banyak kelebihan dibandingkan dengan framework sejenis, antara lain sebagai berikut:

1) Gratis.

Codeigniter berada di bawah lisensi Apache/BSD-style, sehingga bebas untuk digunakan, disebarkan, dan dimodifikasi sesuai kebutuhan.

2) Ringan.

Codeigniter benar-benar ringan karena library dan fungsi tambahan digunakan secara dinamis sesuai kebutuhan, dengan demikian sistem menjadi ramping dan ringan.

3) Cepat.

Codeigniter merupakan salah satu framework tercepat yang dibangun dengan menggunakan PHP.

4) Menggunakan pendekatan MVC.

Codeigniter menggunakan pendekatan MVC, sehingga memungkinkan pemisahan logika, basis data, dan tampilan.

5) URL (Uniform Resource Locator) berbasis segmen.

Codeigniter tidak menggunakan query-string pada URL, melainkan menggunakan segment-based, di mana setiap variabel dipisahkan dengan slash “/”.

6) Paket lengkap.

Codeigniter dibekali dengan fungsi-fungsi yang sering digunakan dalam membuat sebuah web, seperti mengakses database, mengirim email, memanipulasi gambar, dll.

7) Mudah menambah library.

programmer dapat dengan mudah menambah fungsi yang belum ada atau mengedit fungsi yang sudah ada sehingga sesuai dengan kebutuhan.

8) Tidak membutuhkan Template Engine.

Codeigniter tidak membutuhkan Template Engine untuk menghasilkan tampilan, karena cukup dengan PHP dan HTML.

9) Dokumentasi lengkap.

Dokumentasi merupakan hal yang sangat penting dan Codeigniter memiliki dokumentasi yang lengkap serta mudah dipahami.

b. Alasan Menggunakan *Codeigniter*

Elislab (2013) menerangkan bahwa tujuan utama dari Codeigniter adalah kinerja maksimal, kemampuan, fleksibilitas, dan hasil yang seringan mungkin. Dari sudut teknis, *Codeigniter* diciptakan dengan tujuan sebagai berikut ini:

1) Instalasi dinamis.

komponen yang digunakan oleh *Codeigniter* hanya akan dijalankan jika diminta, dengan demikian sistem akan menjadi seminimal mungkin dan akan berjalan seringan mungkin.

2) Komponen tidak saling ketergantungan.

Semakin sedikit komponen yang saling ketergantungan akan menjadikan sistem semakin fleksibel. *Codeigniter* dibuat agar dapat bekerja selonggar mungkin, jika komponen tertentu tidak sesuai kebutuhan, maka dapat diganti dengan komponen lain.

3) komponen dengan fungsi tertentu, setiap komponen yang ada dalam *Codeigniter* dibuat untuk sebuah tujuan tertentu, dengan demikian sebuah komponen dapat bekerja secara maksimal sesuai tujuan.

Dari seluruh penjelasan di atas maka dapat diambil kesimpulan bahwa *Codeigniter* merupakan *framework* berbasis PHP yang dibuat menggunakan pendekatan MVC. *Codeigniter* memiliki banyak kelebihan, salah satunya adalah cepat, mudah digunakan, serta memiliki dokumentasi yang lengkap.