BAB 2

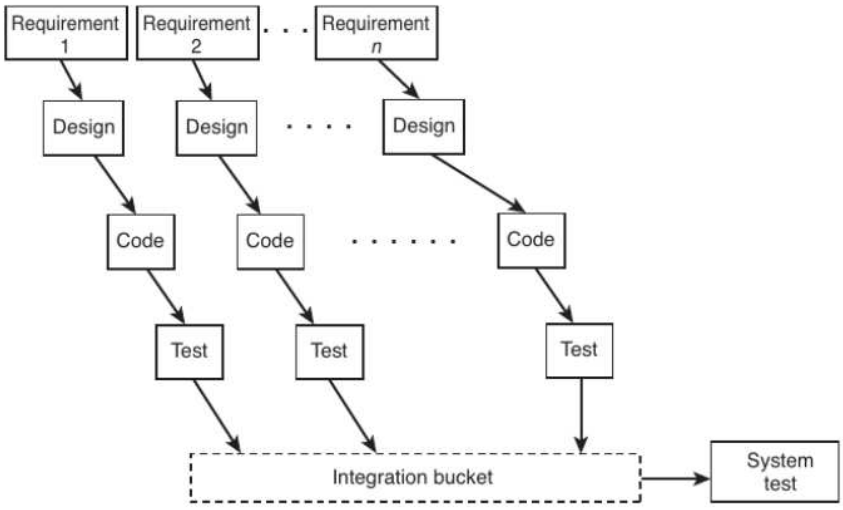
TINJAUAN REFERENSI

2.1. Landasan Teori

Penulis menggunakan beberapa sumber referensi yang membantu dalam penulisan karya ilmiah ini.

2.1.1. *Incremental Model*

Incremental model adalah perkembangan dari model *waterfall*. Dalam metode ini, pembuatan sistem menjadi lebih mudah, dikarenakan sistem yang besar dipecah menjadi komponen kecil, yang akan dibuat secara bertahap dan berulang. Setiap komponen akan mengikuti proses *waterfall*, dan melewati setiap tahap secara berulang. (Tsui, Karam, & Bernal, 2014, pp. 61-63)



Gambar 2.1 Penggambaran dari *Incremental Model*

(Tsui, Karam, & Bernal, 2014, p. 62)

2.1.2. *Unified Modeling Language* (UML)

Dalam UML, terdapat beberapa jenis diagram yang saling berkaitan satu dengan yang lainnya. Tujuan pembuatan diagram ini adalah untuk mempermudah pembuatan sistem, sehingga sistem dapat terdefinisi dengan

baik. Terdapat 4 (empat) diagram yang akan digunakan penulis dalam penulisan ini.

2.1.2.1 *Use Case Diagram*

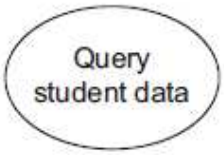
Use case diagram digunakan untuk memperjelas kegunaan dari sistem yang akan dibuat. *Use case diagram* membantu mengekspresikan hal yang dapat dilakukan oleh sistem, tetapi tidak menyatakan penjelasan nyata yang detail, seperti struktur data, algoritma yang digunakan, dll. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 23)

Hal-hal yang harus diperhatikan pada saat pembuatan *use case diagram* adalah:

1. ***Use Case***

   Use case menjelaskan perkiraan mengenai fungsi dari sistem yang akan dibuat. *Use case* menyediakan keuntungan dari *actor* yang berhubungan dengan *use case* tersebut.

   Use case ditentukan dengan cara mengumpulkan kemauan dari pengguna dan menganalisa masalah yang spesifik berdasarkan hal tersebut. *Use case* direpresentasikan dengan oval yang berisikan kata kerja. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 24)

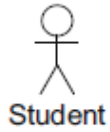


Gambar 2.2 Contoh dari sebuah *use case*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 24)

2. ***Actor***

   Actor adalah semua pihak yang berinteraksi dengan sistem yang sedang dirancang. Proses interaksi antara *actor* dengan sistem direpresentasikan dalam *use case*. (Seidl, Scholz, Huemer, & Kappel, 2012, pp. 25-26)



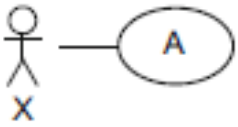
Gambar 2.3 Contoh dari sebuah *actor*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 26)

3. **Relasi antara *Use Case* dan *Actor* (*Associations*)**

   Associations adalah hubungan antara *actor* dengan *use case*. *Associations* mengekspresikan cara *actor* berkomunikasi dengan sistem dan menggunakan fungsi tertentu untuk melakukannya.

   Berikut adalah relasi *actor* X dengan *use case* A:



Gambar 2.4 Contoh dari sebuah *association*

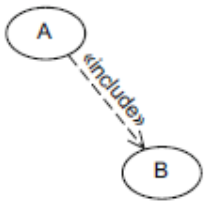
(Seidl, Scholz, Huemer, & Kappel, 2012, p. 27)

4. **Relasi antar *Use Case***

   Dalam relasi antar use case, terdapat 2 jenis, yaitu:

   4.1. *Include*

   Dalam relasi *include*, Terdapat 2 *use case* yang terlibat. *Use case* pertama adalah *base use case* dan *use case* kedua adalah *included use case*. *Base use case* adalah *use case* yang mewajibkan fungsi dari *included use case* agar dapat menjalankan fungsinya sendiri. Singkatnya, *included use case* harus dijalankan setelah *base use case* dijalankan. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 30)



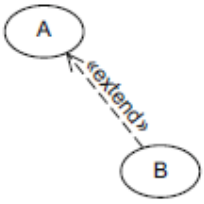
Gambar 2.5 Contoh dari relasi *include*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 30)

Dari gambar diatas, A merupakan *base use case*, dan B merupakan *included use case*. Pada saat *use case* A dijalankan, *use case* B juga harus dijalankan.

4.2. *Extend*

Sama seperti relasi *include*, relasi *extend* juga melibatkan 2 *use case* yang berbeda. *Use case* pertama disebut *base use case*, dan *use case kedua* disebut *extended use case*. Tidak seperti *include*, *base use case* tidak harus mengeksekusi *extended use case*. Kedua *use case* dapat dieksekusi secara terpisah. (Seidl, Scholz, Huemer, & Kappel, 2012, pp. 30-31)



Gambar 2.6 Contoh dari relasi *extend*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 31)

Dari gambar diatas, *use case* A merupakan *base use case*, dan *use case* B merupakan *extended use case*. Pada saat *use case* A dijalankan, *use case* B boleh dijalankan, atau boleh juga tidak dijalankan.

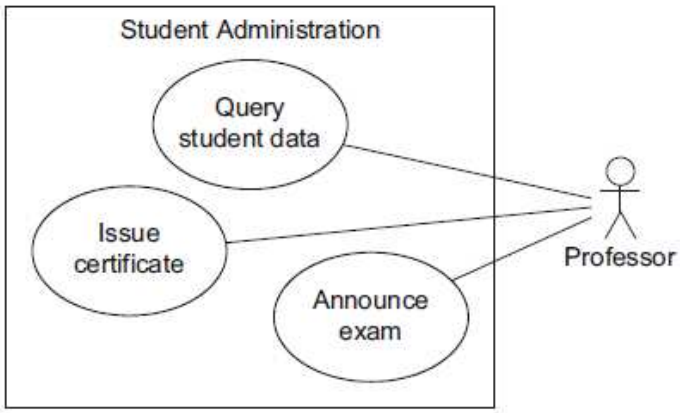
5. ***Boundary***

   Boundary merupakan representasi dari sistem dalam *use case diagram*. *Boundary* biasanya dibuat dengan menggambarkan persegi panjang. *Boudary* membatasi antara sumber luar sistem dengan sumber dalam sistem. Contoh dari sumber luar sistem adalah *actor*, dan contoh sumber dalam sistem adalah *use case*.

Gambar 2.7 Contoh dari sebuah *boundary*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 26)

Berikut ini adalah contoh dari *use case diagram*:



Gambar 2.8 Contoh dari Use Case Diagram

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 25)

Seperti pada *Use Case Diagram* dalam gambar 2.8, terdapat perancangan sistem administrasi mahasiswa, dimana professor, sebagai pengajar dapat

melakukan pencarian data mahasiswa, membuat sertifikat, dan memberitahukan jadwal ujian.

2.1.2.2 *Activity Diagram*

Activity diagram bertugas untuk merepresentasikan aspek prosedur dari sebuah sistem. Diagram ini menjelaskan alur kontrol (*control flow*) dan alur data (*data flow*) melalui beberapa tahap. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 141)

Hal-hal yang harus diperhatikan dalam pembuatan *activity diagram* adalah:

1. ***Action***

   Action direpresentasikan dalam bentuk persegi panjang dengan sudut yang membulat, dan posisi *action* diletakkan di tengah persegi panjang tersebut. *Action* merupakan aksi yang akan dilakukan. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 143)

Gambar 2.9 Contoh dari action

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 143)

2. ***Edges***

   Edges merepresentasikan urutan dari *action* yang akan dieksekusi, dan mendefinisikan urutan dari tahap eksekusi dari sebuah aktivitas. (Seidl, Scholz, Huemer, & Kappel, 2012, pp. 143-144)

Gambar 2.10 Contoh dari *edges*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 143)

3. ***Initial Node***

   Initial node mengindikasikan lokasi awal dari proses eksekusi sebuah aktivitas. *Initial node* direpresentasikan dengan bentuk lingkaran hitam penuh. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 147)

Gambar 2.11 Contoh dari *initial node*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 147)

4. ***Decision Node***

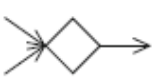
   Decision node bekerja sebagai poin pertukaran dan bekerja seperti pilihan aksi yang harus dipenuhi. *Decision node* digambarkan seperti bentuk berlian dengan satu panah masuk, dan banyak panah keluar. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 149)

Gambar 2.12 Contoh dari *decision node*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 149)

5. ***Merge Node***

   Merge node digunakan untuk menyatukan alur aktivitas menjadi satu. Sama seperti *decision node*, *merge node* juga direpresentasikan dalam bentuk berlian, dimana terdapat banyak panah masuk, dan satu panah keluar. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 150)

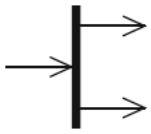


Gambar 2.13 Contoh dari *merge node*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 150)

6. ***Parallelization Node***

   Parallelization node adalah cara merepresentasikan alur aktivitas pada saat alur eksekusi pecah menjadi banyak secara bersamaan. *Parallelization node* digambarkan dengan cara persegi panjang hitam penuh memanjang kebawah, dengan satu arah panah masuk, dan banyak arah panah keluar. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 150)

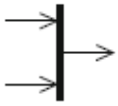


Gambar 2.14 Contoh dari paralleization node

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 150)

7. ***Synchronization Node***

   Synchronization node adalah kebalikan dari *parallelization node*. *Syncrhonization node* menyatukan alur eksekusi yang berbeda menjadi satu. *Synchronization node* digambarkan sama seperti *parallelization node*. Yang membedakan adalah, *synchronization node* memiliki banyak panah masuk, dan hanya memiliki satu panah keluar. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 150)



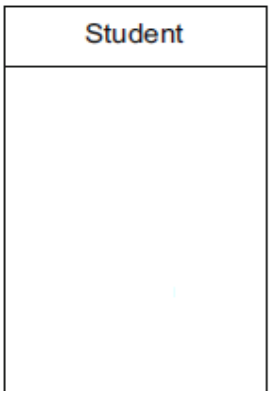
Gambar 2.15 Contoh dari *synchronization node*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 150)

8. ***Partition***

   Partition membantu dalam mebuat grup dari alur aktivitas berdasarkan pelaku dari aktivitas tersebut. Dalam satu *activity diagram*, memungkinkan memiliki lebih dari satu *partition*. Istilah

lainnya dalam *partition* adalah *swimlane*. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 157)



Gambar 2.16 Contoh dari *partition*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 157)

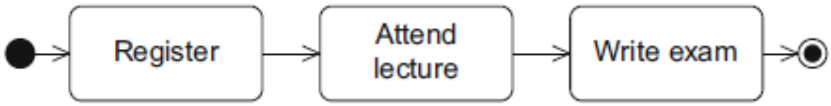
9. ***Final Node***

   Final Node menunjukan akhir dari alur aktivitas. *Final node* direpresentasikan dalam bentuk lingkaran penuh, dan terdapat lingkaran luar. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 151)

Gambar 2.17 Contoh dari *final node*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 151)

Berikut adalah contoh dari *activity diagram*:



Gambar 2.18 Contoh dari *Activity Diagram*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 148)

Dalam *activity diagram* pada gambar 2.18, menandakan bahwa mahasiswa dapat melakukan ujian melalui beberapa tahap. Tahap pertama adalah, mahasiswa harus melakukan registrasi terlebih dahulu. Setelah itu, mahasiswa harus hadir dalam kelas. Setelah itu, mahasiswa harus mengisi kertas ujian.

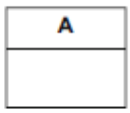
2.1.2.3 *Class Diagram*

Class diagram adalah pemodelan struktur statis dari sebuah sistem, yang menjelaskan elemen dari sebuah sistem dan relasi yang berhubungan antar sesama. Elemen dan relasi antar elemen tidak berganti setiap waktu. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 49)

Hal-hal yang terdapat dalam *class diagram* adalah:

1. ***Class***

   Class adalah perencanaan konstruksi untuk kumpulan dari beberapa objek yang muncul di dalam sistem yang harus dispesifikasikan. *Class* dapat menggambarkan sesuatu, seperti orang (contohnya siswa), sesuatu (contohnya bangunan), peristiwa (contohnya mata pelajaran, atau ujian), atau bahkan objek yang bersifat abstrak seperti grup. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 52)



Gambar 2.19 Contoh dari *class*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 52)

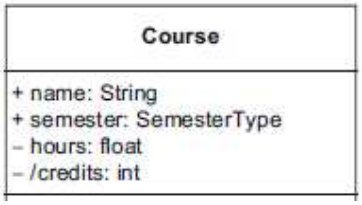
2. ***Attribute***

   Attribute adalah semua hal yang menjelaskan sebuah *class*. Setiap *attribute* setidaknya memiliki nama, dan tipe data. Tipe data yang dimiliki oleh *attribute* dapat beragam, mulai dari tipe data primitif (*integer, string*), atau tanggal, enumerasi, atau *class* yang dibentuk oleh pengguna sendiri. Setiap *attribute* memiliki *visibility*, yaitu

interaksi *attribute* dengan fungsi atau proses lain diluar *class*. (Seidl, Scholz, Huemer, & Kappel, 2012, pp. 54-55, 59)

Terdapat 3 jenis *visibility*, yaitu;

- *Public*, yaitu hak akses dari *attribute* atau *operation* dapat diakses oleh siapapun tanpa terkecuali. Ditandai oleh tanda tambah (+).
- *Protected*, yaitu hak akses dari *attribute* atau *operation* yang hanya dapat diakses oleh *class* itu sendiri, atau anak dari *class* tersebut. Ditandai oleh tanda pagar (#).
- *Private*, yaitu hak akses dari *attribute* atau *operation* yang hanya dapat diakses oleh *class* itu sendiri, dan tidak dapat diakses oleh siapapun selain itu. Ditandai oleh tanda kurang (-).

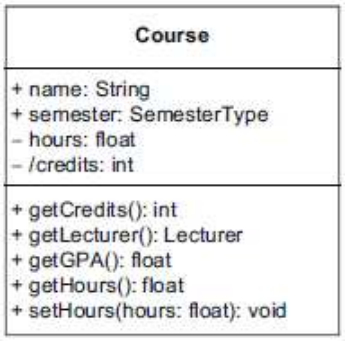


Gambar 2.20 Contoh dari *class* dan *attribute*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 53)

3. ***Operation***

   Operation adalah semua proses yang dapat dilakukan dalam suatu *class*. Proses ini direpresentasikan dalam bentuk fungsi. *Operation* hanya menjelaskan sifat dari sebuah *class*, tanpa menjabarkannya secara detail. (Seidl, Scholz, Huemer, & Kappel, 2012, pp. 56-57)



Gambar 2.21 Contoh dari *class*, *attribute*, dan *operation*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 53)

4. ***Association***

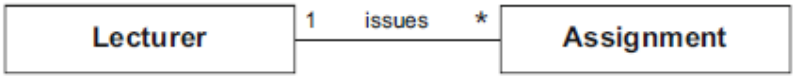
   Association adalah relasi antar *class* yang saling terkait. *Association* menjelaskan *class* mana saja yang akan saling berinteraksi antara satu dengan yang lainnya. Jika *visibility* dari *attribute* dan *operation* mereka saling mendukung, komunikasi antar sesama dapat saling mengakses *attribute* dan *operation* mereka. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 60)

Gambar 2.22 Contoh dari *association* (*class* A dan *class* B)

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 60)

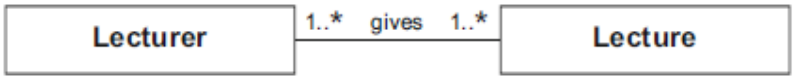
5. ***Multiplicity***

   Multiplicity adalah rentang jumlah dari hubungan antar *class* (*association*) dimana jumlah tersebut menyatakan jumlah objek yang akan berinteraksi antar class. *Multiplicity* didefinisikan dalam *association.* Format dari *multiplicity* adalah minimum..maksimum. Namun, format *multiplicity* juga ada yang hanya menyatakan jumlah objek minimumnya saja. (Seidl, Scholz, Huemer, & Kappel, 2012, pp. 62-63)



Gambar 2.23 Contoh dari *multiplicity* (format minimum saja)

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 62)



Gambar 2.24 Contoh dari *multiplicity* (format minimum..maksimum)

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 62)

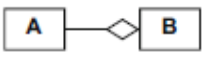
6. ***Aggregation***

   Aggregation adalah bentuk khusus dari *association* yang digunakan untuk mengekspresikan bahwa instansi dari satu kelas merupakan bagian dari kelas lainnya. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 67)

   Terdapat 2 jenis aggregation, yaitu:

   - ***Shared Aggregation***

     Shared aggregation mengekpresikan hubungan yang lemah antara kedua kelas. Sebuah elemen dapat menjadi bagian dari beberapa elemen lainnya secara langsung. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 68)



Gambar 2.25 Contoh dari *shared aggregation*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 68)

- ***Composition***

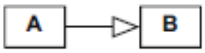
  Composition mengekspresikan sebuah elemen hanya dapat dimiliki tidak lebih dari satu objek (objek komposit) dalam waktu yang spesifik. Jika objek komposit dihapus, maka bagian dari objek tersebut juga terhapus (Seidl, Scholz, Huemer, & Kappel, 2012, p. 68)

Gambar 2.26 Contoh dari composition

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 68)

7. ***Generalization***

   Hubungan *generalization* mengekspresikan bahwa karakteristik dan *association* yang telah ditentukan untuk *general class* (*superclass*) akan diberikan atau diwariskan kepada *subclass* (anak kelas)

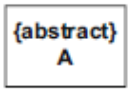


Gambar 2.27 Contoh dari *generalization*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 70)

8. ***Abstract Class***

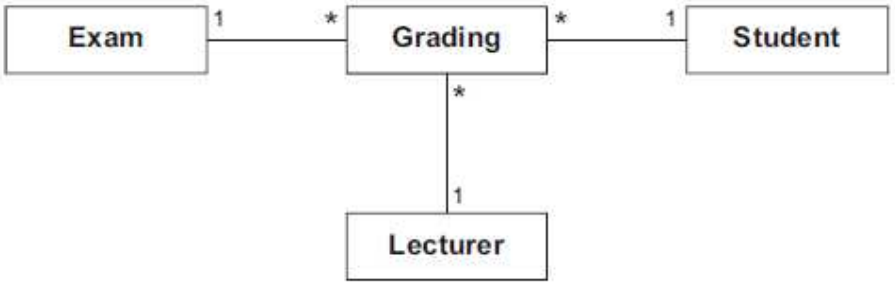
   Abstract class adalah *class* yang tidak dapat direpresentasikan dalam bentuk objek (tidak dapat diinstansikan). *Class* ini tidak memiliki objek. Hanya subclass saja yang dapat dibuat objeknya. (Seidl, Scholz, Huemer, & Kappel, 2012, pp. 72-73)



Gambar 2.28 Contoh dari *abstract class*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 72)

Berikut adalah contoh dari *class diagram*:



Gambar 2.29 Contoh *Class Diagram*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 65)

Terdapat 4 kelas yang terdapat dalam sistem, yaitu kelas *Exam*, *Grading*, *Student*, dan *Lecturer*. Kelas *Exam* digunakan untuk menampung semua hal yang berhubungan dengan ujian, seperti tanggal ujian, nama ujian, dan lain-lain. Kelas *Grading* digunakan untuk penilaian ujian, seperti bobot nilai, syarat kelulusan, dll. Kelas *Student* digunakan untuk menampung data yang berhubungan dengan siswa, seperti nama siswa, tanggal lahir, dll. Kelas *Lecturer* digunakan untuk mengetahui hal yang berhubungan dengan dosen, seperti nama dosen, kode dosen, dll.

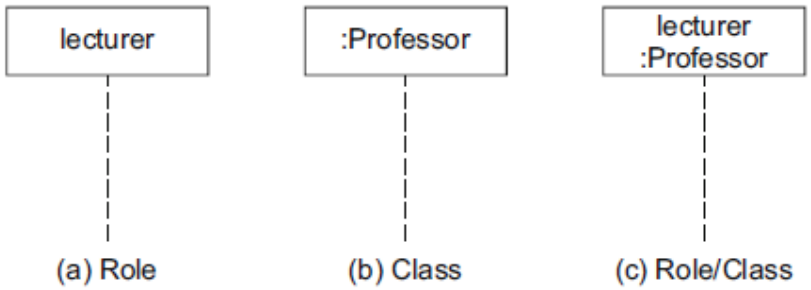
2.1.2.4 *Sequence Diagram*

Sequence diagram menjabarkan interaksi antara objek untuk memenuhi tugas yang spesifik. Fokus dari diagram ini adalah urutan kronologi dari pertukaran pesan antara interaksi pasangan. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 20)

Hal yang harus diperhatikan dalam *sequence diagram* adalah:

1. ***Lifeline***

   Lifeline ditunjukkan dalam bentuk vertikal, biasanya dengan garis putus-putus yang direpresentasikan sebagai waktu hidup dari objek yang berhubungan dengannya. Di kepala *lifeline*, terdapat persegi panjang yang merepresentasikan nama *Role*:*class*. Ekspresi ini menunjukkan nama dari peran dari objek dan class dari objek yang terkait. (Seidl, Scholz, Huemer, & Kappel, 2012, pp. 108-109)

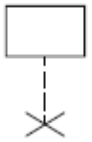


Gambar 2.30 Contoh dari *lifeline*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 109)

2. ***Destruction Event***

   Destruction event adalah proses sebuah objek dihapus pada saat melakukan interaksi dengan objek lainnya. *Destruction event* ditandakan dengan *lifeline* yang memiliki tanda X besar di bagian paling bawah. Jika *destruction event* tidak terjadi, *lifeline* terus memanjang hingga daerah paling bawah dalam sequence diagram.



Gambar 2.31 Contoh dari *destruction event*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 113)

3. ***Message***

Terdapat 3 jenis *message* berdasarkan arah dan waktu dari *message* tersebut dikirim, yaitu:

- ***Synchronous Message***

  Synchronous message adalah pesan yang dikirim oleh pengirim pesan, dan pengirim pesan harus menunggu sampai pesan respon (*response message*) didapatkan sebelum melanjutkan proses. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 112)

Gambar 2.32 Contoh dari *synchronous message*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 112)

- ***Asynchronous Message***

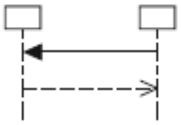
  Asynchronous message adalah pesan yang dikirim oleh pengirim pesan, dan pengirim pesan tetap melanjutkan prosesnya setelah pesan teresebut dikirim. (Seidl, Scholz, Huemer, & Kappel, 2012, p. 112)

Gambar 2.33 Contoh dari *asynchronous message*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 112)

- ***Response Message***

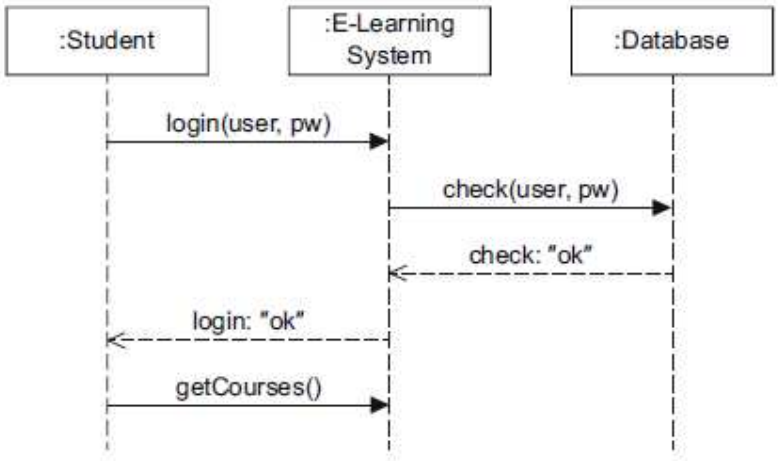
  Response Message adalah pesan yang didapatkan oleh pengirim, dan pesan ini dibutuhkan pengirim untuk melanjutkan prosesnya. *Response message* hanya dapat terjadi setelah *synchronous message.*



Gambar 2.34 Contoh dari *synchronous message* dan *response message*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 112)

Berikut adalah contoh dari *sequence diagram*:



Gambar 2.35 Contoh *Sequence Diagram*

(Seidl, Scholz, Huemer, & Kappel, 2012, p. 137)

Dalam *sequence diagram* pada gambar …, menunjukkan bahwa siswa ingin masuk ke dalam sistem dengan cara *login*. Pada saat siswa *login*, fungsi *login* dalam sistem berjalan dan masuk ke kelas *E-Learning System*. Lalu, sistem akan mengecek ke basis data, apakah siswa memasukkan nama pengguna dan kata sandi dengan benar. Basis data lalu memberikan pesan ok kepada sistem, dan sistem memberikan pesan ok kepada siswa. Lalu, siswa meminta semua mata pelajaran yang sedang diambil olehnya.

2.1.3. *Eight Golden Rules* (Delapan aturan emas)

Menurut Shneiderman (Shneiderman, et al., 2016), terdapat delapan aturan emas dalam perancangan antarmuka antara pengguna dengan sistem, yaitu:

1. Berusaha untuk konsisten (*strive for consistency*)

   Konsisten dalam aksi yang berurutan sangat dibutuhkan dalam kondisi yang serupa. Terminologi yang identik harus digunakan dalam menu, dan tampilan bantuan. Konsisten terhadap warna, letak, kapitalisasi, tulisan, dan lain-lain. Penambahan *exception* (pemberlakuan disaat pengguna atau sistem melakukan kesalahan), seperti penambahan konfirmasi pada saat data dihapus.

2. Menyediakan kemudahan pengunaan yang universal (*seek universal usability)*

   Pemberian kemudahan pada pengguna sangatlah penting. Pengguna tidak terbatas oleh usia, jenis kelamin, disabilitas, atau berbataskan budaya, dan lainnya. Penambahan fitur seperti penjelasan, atau *shortcut* akan meningkatkan kemudahan pengguna.

3. Memberikan umpan balik secara informatif (*offer informative feedback*)

   Pada setiap aksi pengguna, harus terdapat penerimaan pesan dari sistem. Dalam aksi yang kecil, respons dapat dalam bentuk sederhana. Sedangkan pada aksi yang cukup besar, respons harus lebih besar.

4. Pemberian dialog dalam penutupan (*design dialogue to yield closure*)

   Urutan aksi harus diatur menjadi grup dengan urutan awal, tengah, dan akhir. Umpan balik informatif setelah terjadinya grup aksi akan memberikan kepuasan pada hal yang telah dicapai pengguna, kelegaan, dan juga sebagai penanda untuk menjalankan aksi selanjutnya.

5. Mencegah kesalahan (*prevent errors*)

   Sebanyak mungkin, buatlah desain antar muka sehingga pengguna tidak membuat kesalahan fatal. Contohnya, berikan warna berbeda pada menu yang tidak dapat dibuka, dan berikan validasi pada kolom entri yang hanya menerima data numerik.

6. Menyediakan pengembalian aksi yang mudah (*permit easy reversal of actions*)

Sebanyak mungkin, aksi dapat dikembalikan. Maksudnya, pada saat pengguna ingin melakukan sesuatu, sistem akan memberikan jalur alternatif bagi pengguna jika pada saat proses berlangsung, proses dapat dibatalkan. Fitur ini membantu menghilangkan kecemasan kepada pengguna, karena penguna mengetahui bahwa kesalahan tidak bisa dihilangkan.

7. Memberikan kontrol kepada pengguna (*Keep users in control*)

   Pengguna yang sudah sering menggunakan aplikasi akan memiliki kemauan besar untuk mengatur antar muka, dan antar muka tersebut memberikan balasan kepada aksi yang telah dilakukan.

8. Mengurangi beban daya ingat pendek (*Reduce short-term memory load*)

   Manusia memiliki keterbatasan kapasitas untuk menerima informasi memori jangka pendek. Ini membuat perancang aplikasi harus menghindari perancangan antar muka jika pengguna harus mengingat informasi dalam satu tampilan, dan menggunakan informasi tersebut dalam tampilan selanjutnya.

2.1.4. *Five Measurable Human Factors Issues* (Lima faktor manusia terukur)

Shneiderman (Ben & Plaisant, 2010) mengemukakan lima faktor manusia terukur yang digunakan untuk membantu dalam meningkatkan hubungan interaksi antara manusia dengan komputer, yaitu:

1. Waktu untuk belajar (*time to learn*)
   Berapa lama waktu yang diperlukan untuk semua anggota komunitas untuk belajar bagaimana cara menggunakan kumpulan dari perintah
2. Kecepatan performa (*speed of performance*)
   Berapa lama waktu yang diperlukan sistem untuk mengeksekusi kumpulan dari tugas yang diberikan pengguna.
3. Tingkat kesalahan oleh pengguna (*rate of errors by users*)
   Berapa lama dan jenis kesalahan seperti apa yang terjadi pada saat proses eksekusi tugas.
4. Kepuasan subjektif (*subjective satisfaction*)

Sebaik apa pengguna suka dengan sistem yang telah dibuat? Hal ini dapat dikonfirmasi dengan cara wawancara, atau survey.

5. Daya ingat pada waktu (*retention over time*)
   Sebaik apa pengguna menjaga pengetahuan mereka setelah sejam, sehari, atau seminggu. Daya ingat memiliki hubungan pada kemudahan untuk belajar.

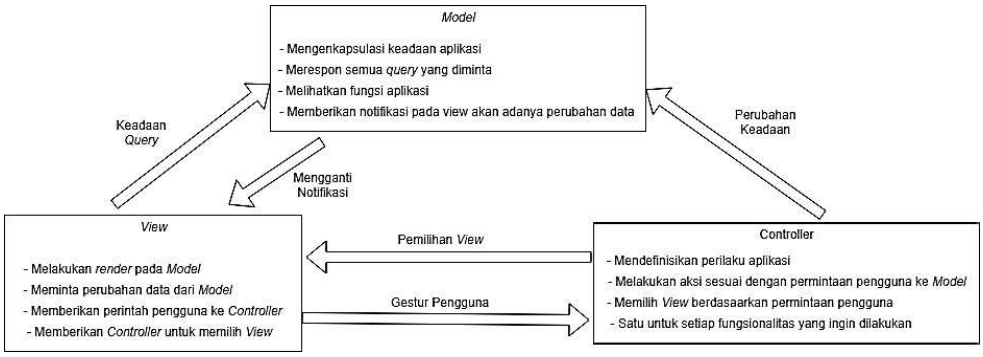
2.1.5. MVC (*Model-View-Controller*)

MVC digunakan untuk membagi aplikasi, atau bagian kecil dari sebuah aplikasi menjadi tiga bagian, yaitu *model*, *view*, dan *controller*. Secara aslinya, MVC membuat developer untuk melakukan proses pemetaan tradisional, yaitu input, proses, dan *output*. (Qureshi & Sabir, 2013, p. 7)

Model bertanggung jawab dalam proses data, seperti koneksi basis data, *query* basis data, dan mengimplementasi aturan bisnis. Semua data yang diproses akan melalui lapisan ini.

View digunakan sebagai lapisan presentasi data secara grafis terlepas dari pemrosesan data. Semua presentasi, format, dan pengaturan tampilan data merupakan tanggung jawab dari lapisan ini. View tidak terikat oleh koneksi basis data, dan *query* dalam basis data.

Controller digunakan untuk menerjemah dan menangkap input pengguna menjadi aksi yang dapat dilakukan oleh model. Tanggung jawab lainnya adalah untuk memilih *view* mana yang akan ditampilkan berdasarkan input pengguna.



Gambar 2.36 Diagram MVC (*Model-View-Controller*)

2.1.6. NET *Framework*

.NET *Framework* adalah pembantu perangkat lunak untuk membuat sistem dalam ruang lingkup operasi sistem *Windows*, dan operasi sistem bukan berasal dari *Microsoft*, seperti Mac OS X, dan variasi Unix/Linux lainnya. .NET *Framework* diperkenalkan pertama kali pada tahun 2002, dan memberikan fasilitas yang kuat, fleksibel, dan lebih mudah untuk melakukan pemrograman aplikasi. (Troelsen & Olsen, 2012, p. 3)

Berikut adalah fitur-fitur yang disediakan dalam .NET *Framework*:

1. *Interoperability with existing code*

   Pembuatan aplikasi yang telah dibuat dapat diintegrasikan kedalam versi terbaru tanpa ada kendala. Sesuai dengan .NET versi 4.0, proses integrasi telah disederhanakan dengan tambahan kata kunci dinamis.
2. *Support for numerous programming languages*

   Aplikasi berbasis .NET dapat dibuat menggunakan bahasa pemrograman yang cukup banyak (C#, Visual Basic, F#, dan lain-lain).
3. *A common runtime engine shared by all .NET-aware languages*

   Salah satu aspek dari penggunaan *framework* ini adalah proses integrasi yang mudah jika bahasa tersebut dapat mengenal .NET Framework.
4. *A comprehensive base class library*

   Framework ini memudahkan pengguna dalam melakukan proses implementasi dengan cara menyediakan kelas dasar yang dapat digunakan seperti *library*. Sehingga, jika developer ingin melakukan sesuatu yang

kompleks, developer hanya memasukkan kelas dasar tersebut ke dalam *code*.

5. *A simplified deployment model*

   .NET *library* tidak terdaftar dalam *system registry*. Sehingga, .NET *platform* memperbolehkan proses eksekusi lebih dari satu .dll *extension* dalam satu mesin.

2.1.7. ASP.NET

ASP.NET adalah bagian dari .NET *Framework* yang memungkinkan penggunannya untuk membuat aplikasi berbasis web yang canggih. (Spaanjaars, 2014, pp. 1-2)

Menurut Spanjaars, ASP.NET (*Active Server Pages* .NET) memberikan keuntungan kepada developer sebagai berikut:

1. Pemisahan antara bagian *presentation* dan bagian *code*. *Presentation* adalah bagian yang digunakan untuk tampilan web, sedangkan *code* adalah bagian yang digunakan untuk melakukan logika pemrograman Pemisahan kedua bagian ini menguntungkan, karena dalam ASP (Versi sebelum ASP.NET) bagian *code* sering kali tersebar dalam bagian HTML, sehingga sangat sulit untuk menggubah halaman.
2. Pembuatan aplikasi model yang serupa dengan pembuatan aplikasi *desktop*. Hal ini membuat aplikasi *desktop* berbasis *Visual Basic* dapat dipindahkan menjadi aplikasi web secara mudah.
3. Fitur pembuatan aplikasi yang sanggat baik (dinamai *Visual Studio* .NET) yang memudahkan developer untuk membuat aplikasi berbasis web secara visual.
4. Terdapat pilihan bahasa program berbasiskan objek (OOP) yang sangat populer sekarang, seperti *Visual Basic* .NET dan C#
5. Adanya akses penggunaan .NET *framework*, sehingga memudahkan developer untuk bekerja dengan basis data, dokumen, *e-mail*, alat jaringan, dll.

2.1.8. ASP.NET MVC

ASP.NET MVC adalah *framework* untuk membuat aplikasi berbasis web yang mengikuti kaidah pola *Model-View-Controller* (MVC) kedalam *framework* ASP.NET. (Galloway, Wilson, Allen, & Matson, 2014)

ASP.NET mendukung dua lapisan abstraksi, yaitu:

1. *System.Web.UI*
   Merupakan lapisan *web form* yang meliputi *server control*, *ViewState*, dan lainnya.
2. *System.Web*
   Menjadi perantara, yaitu menyediakan *basic web stack*, meliputi *module*, *handler* (penanganan), HTTP *stack*, dan lainnya.

MVC memisahkan antarmuka pengguna dari sebuah aplikasi menjadi tiga aspek utama:

1. *Model*
   Kumpulan kelas yang menjabarkan data yang digunakan dan peraturan bisnis yang ingin diberlakukan dan bagaimana cara data dapat diganti atau dimanipulasi.
2. *View*
   Mendefinisikan bagaimana aplikasi merepresentasikan antarmuka pengguna.
3. *Controller*
   Kumpulan dari kelas yang menangani komunikasi dari pengguna, alur aplikasi secara keseluruhan, dan logika dari sebuah aplikasi.

2.1.9. Basis Data

Basis data ialah kumpulan data yang secara logika saling berhubungan dengan deskripsinya, didesain untuk memenuhi kebutuhan informasi yang diperlukan dalam sebuah organisasi. (Connolly & Begg, 2015, p. 63)

Menurut Connolly, terdapat 3 hal penting yang harus diingat dalam basis data, yaitu:

1. Entitas, yaitu objek unik (seperti orang, tempat, barang, konsep, atau peristiwa) dalam organisasi yang direpresentasikan dalam basis data
2. Atribut, yaitu properti yang menjelaskan beberapa aspek dari objek yang mau disimpan. Atribut merupakan penjelasan dari entitas.
3. Relasi, yaitu penjelasan hubungan antara entitas.

2.1.9.1. Data

Menurut Connolly mengenai lingkungan DBMS (*Database Management System*), data merupakan salah satu dari komponen basis data yang terpenting. Data bertindak sebagai jembatan mengenai komponen mesin dan komponen manusia. (Connolly & Begg, 2015, pp. 66-69)

2.1.9.1.1. *Data Definition Language* (DDL)

DDL adalah bahasa yang digunakan untuk menjelaskan dan memberi nama kepada entitas, atribut, dan relasi yang dibutuhkan dalam aplikasi, bersamaan dengan integritas dan peraturan keamanan. (Connolly & Begg, 2015, p. 90)

2.1.9.1.2. *Data Manipulation Language* (DML)

DML adalah bahasa yang menyediakan kumpulan operasi yang digunakan untuk mendukung pemanipulasian data yang terdapat dalam basis data. (Connolly & Begg, 2015, p. 90)

Menurut Connolly, operasi manipulasi data biasanya meliputi:

1. Pemasukkan data baru ke dalam basis data
2. Modifikasi data yang telah ada dalam basis data
3. Pengambilan kembali data yang terdapat dalam basis data
4. Penghapusan data dalam basis data

2.1.9.2. *Database Management System* (DBMS)

DBMS adalah sistem perangkat lunak yang membantu penggunanya untuk mendefinisikan, membuat, menjaga, dan mengontrol akses dari basis data. DBMS berinteraksi dengan pengguna aplikasi program dan basis data. (Connolly & Begg, 2015, p. 64)

Dalam karya ilmiah ini, penulis menggunakan *SQL Server Management System* sebagai DBMS.

2.1.9.3. *Relational Database Management System (RDBMS)*

RDBMS adalah salah satu contoh dari perangkat lunak yang digunakan untuk memproses data, dan sangat banyak digunakan pada saat ini. Dalam model relasi semua data terstruktur secara logika dan direpresentasikan dalam relasi (tabel). Setiap relasi memiliki nama masing-masing dan terbentuk dari atribut (kolom) data. Setiap *tuple* (baris) mengandung satu nilai setiap atribut. Beberapa contoh dari RDBMS adalah MySQL dari *Oracle Corporation*, *Ingres* dari *Actian Corporation*, *SQL Server* dari *Microsoft*, dan *Informix* dari IBM. (Connolly & Begg, 2015, pp. 73, 149)

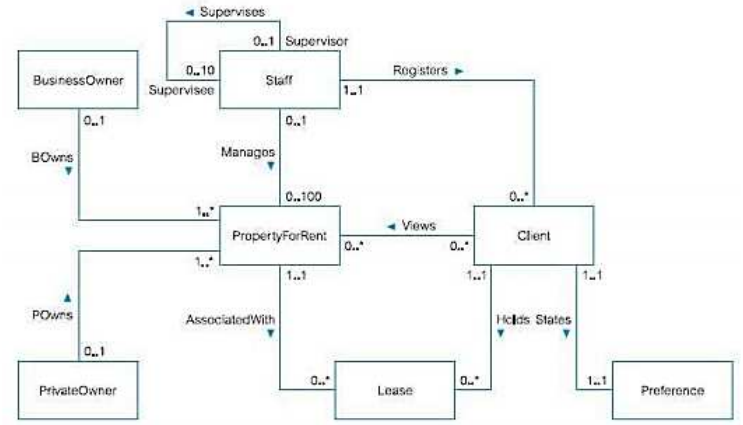
2.1.9.4. *Structured Query Language (SQL)*

SQL adalah bahasa pemrograman yang digunakan dalam proses DML dan DDL. SQL sudah menjadi bahasa pemrograman formal dan secara de facto menjadi bahasa standar untuk setiap RDBMS. (Connolly & Begg, 2015, p. 64)

2.1.9.5. *Entity Relationship Diagram (ERD)*

ERD adalah pendekatan dari atas ke bawah untuk melakukan proses pendesainan basis data yang dimulai dengan mengidentifikasikan bagian data yang penting, yaitu entitias dan relasi antara data yang harus direpresentasikan dalam bentuk model. ERD Sangat penting

untuk pendesainan basis data untuk memahami dan membentuk kerangka dari basis data. (Connolly & Begg, 2015, p. 405)



Gambar 2.37 Contoh dari ERD (Entity Relationship Diagram)

2.1.9.6. *Stored Procedure*

Stored Procedure adalah salah satu dari *subprogram*. *Subprogram* adalah blok PL/SQL (daerah bahasa program prosedural SQL yang dijadikan satu sebagai sebuah grup) yang dapat menerima parameter dan dapat dipanggil. (Connolly & Begg, 2015, p. 280)

Stored Procedure digunakan untuk melakukan sebuah tindakan logika yang terjadi dalam basis data. Tujuan dari penggunaan *Stored Procedure* adalah unuk memudahkan developer dalam proses *maintenance,* sehingga basis data tidak perlu dirubah melainkan hanya *Stored Procedure* saja yang harus diubah.

2.1.9.7. *View*

View adalah relasi yang dibuat secara virtual atau relasi yang dapat diturunkan dari relasi lainnya. *View* bisa didapat tidak hanya dari satu relasi saja, melainkan dapat diturunkan dari banyak relasi. *View* tidak harus ada dalam basis data, namun dapat diproduksi pada saat

permintaan dari pengguna pada saat waktu permintaan. (Connolly & Begg, 2015, pp. 63, 251)

2.1.10. Visual Studio 2017

Visual Studio adalah IDE (*Integrated Development Environment*) baru untuk developer yang dibuat oleh *Microsoft*. *Visual Studio* tidak hanya menargetkan pembuatan aplikasi yang berbasiskan *Microsoft*, namun dapat juga dipakai untuk membuat aplikasi dalam bahasa lainnya, seperti C++, Python, dan lain lain. Singkatnya, *Visual Studio* akan menjadi IDE untuk setiap developer yang ingin membuat aplikasi dalam *platform* apapun. (Chowdhury, 2017, p. 1)

2.1.11. *Bootstrap*

Bootstrap adalah *framework* yang digunakan untuk membuat web menjadi responsif (aplikasi mengikuti bentuk dari layar) tanpa dibatasi oleh *browser* yang digunakan oleh pengguna.

Saat ini, *bootstrap* memiliki variasi yang banyak. Namun, penulis menggunakan versi *bootstrap* 4.0.0. Versi ini digunakan karena kelengkapan fitur yang dimiliki versi ini. Selain itu, versi 4.0.0 merupakan versi terbaru yang ada pada saat ini. (Lambert, 2016, pp. 7-8)

2.1.12. *Cascading Style Sheet* (CSS)

Cascading Style Sheet adalah alat yang digunakan oleh desainer web dan developer. Penggunaan CSS dibarengi oleh *markup language*, seperti HTML (*Hypertext Markup Language*) dan XHTML untuk membuat web. CSS menyediakan informasi yang dibutuhkan oleh *web browser* untuk mengatur tampilan dari halaman web, seperti posisi dari elemen HTML, gaya tulisan, latar belakang, warna dan gambar, dll. (Pouncey & York, 2011, p. xxiii)

2.1.13. C#

C# adalah bahasa pemrograman modern yang berbasis objek dan digunakan untuk kebutuhan secara umum. C# dibuat oleh *Microsoft* dibarengi dengan .NET *Platform*. C# memiliki kemiripan dengan bahasa pemrograman Java dan C++, setaraf dengan bahasa pemrograman seperti Delphi, VB.NET, dan C.

Pada saat ini, C# menjadi salah satu bahasa program yang populer. Bahasa ini digunakan oleh banyak developer di seluruh dunia. Ini disebabkan karena beberapa perusahaan yang bergantung dengan *Microsoft platform* sangat banyak. (Nakov, et al., 2013, p. 18)

2.1.14. JavaScript

JavaScript adalah salah satu dari bahasa program untuk web. JavaScript memperoleh sintaksisnya dari bahasa Java. JavaScript digunakan untuk menspesifikasikan tingkah laku dari halaman web. (Flanagan, 2011, p. 1)

2.1.15. JQuery

JQuery adalah salah satu dari API (*Application Programming Interface*) yang sangat mudah digunakan untuk *scripting* konten dokumen, *presentation*, dan tingkah laku. Keuntungan dari penggunaan JQuery adalah JQuery dapat digunakan dalam *web browser* yang memiliki pengguna yang banyak, hingga *web browser* yang lama seperti IE6 (*Internet Explorer* 6). (Flanagan, 2011, p. 11)

2.1.16. *NuGet Package Manager*

NuGet adalah ekstensi dari *Visual Studio* yang terdpat dalam *Visual Studio* versi 2012 atau keatas. Memanfaatkan keuntungan dari integrasi antara *NuGet* dengan *Visual Studio* dapat membantu pelancaran proses pembuatan aplikasi (Balliauw & Decoster, 2013)

NuGet Package Manager adalah ekstensi yang dimiliki oleh *Visual Studio* yang terintegrasi dengan *Visual Studio* IDE (*Integrated Development*

Environment), yang memungkinkan penggunanya untuk memasang *package* (perangkat lunak dengan fungsi yang serupa, dan fitur yang serupa yang dijadikan satu menjadi kumpulan program perangkat lunak) semudah menambahkan referensi ke dalam proyek (Balliauw & Decoster, 2013, p. xxx)

2.1.17. *Black-Box Testing*

Black Box Testing adalah teknik dalam melakukan *testing* tanpa mengetahui pengetahuan mengenai proses internal dari sebuah aplikasi. *Testing* ini mengevaluasi aspek mendasar dari sistem dan tidak memiliki keterkaitan dengan struktur logika internal dari sistem. (Arai, et al., 2012, p. 12)

2.1.18. *REST API*

Web service adalah web server yang dibuat untuk digunakan oleh aplikasi yang sedang berjalan. Progrm klien menggunakan aplikasi program antarmuka (API) untuk berinteraksi dengan *web service*. Salah satu cara membuat aplikasi menjadi RESTful adalah dengan menggunakan HTTP *Request.* (Masse, 2012, p. 5)

Terdapat beberapa HTTP Request yang umum digunakan, yaitu:

- GET: HTTP Request yang digunakan untuk mengambil representasi dari sumber data
- POST: HTTP Request yang digunakan untuk membentuk sumber data baru dengan menggunakan koleksi atau menjalankan controller.
- PUT: HTTP Request yang digunakan untuk memasukkan sumber data baru
- DELETE: HTTP Request yang digunakan untuk menghapus data