BAB 2

LANDASAN TEORI

2.1. Kajian Pustaka

2.1.1. E-Learning

Menurut Randy Garrison (2011), *E-Learning* didefinisikan sebagai komunikasi asinkron dan sinkron yang dimediasi secara elektronik untuk tujuan membangun dan mengkonfirmasikan pengetahuan. Landasan teknologi e-*learning* adalah *internet* dan teknologi komunikasi yang terkait. Dua aplikasi utama yang membentuk *e-learning* adalah *online learning* dan *blended learning*. *Blended learning* artinya pendidikan dilakukan dengan kombinasi antara pembelajaran tatap muka secara langsung (*face-to-face*) dan pembelajaran secara *online learning*.

2.1.2. System Development Life Cycle (SDLC)

Proyek adalah sebuah usaha terencana yang memiliki sebuah awal dan akhir, dan menghasilkan hasil atau produk yang sudah ditentukan sebelumnya. Proyek pengembangan sistem menjelaskan mengenai sebuah usaha terencana untuk menghasilkan sistem informasi baru. Kesuksesan sangat bergantung pada ketersediaan rencana dan urutan dari tugas dan kegiatan yang harus dilaksanakan yang akan berujung pada sebuah sistem informasi yang handal, kuat, dan efisien.

Untuk mengelola proyek dengan analisis, desain, dan kegiatan pengembangan lainnya, diperlukan kerangka kerja mengelola proyek untuk memandu dan mengkoordinasikan pekerjaan dalam tim proyek. Menurut Satzinger (2012), *Systems Development Life Cycle* mengidentifikasi semua kegiatan yang diperlukan untuk membangun, meluncurkan, dan memelihara sistem informasi. Biasanya, SDLC mencakup semua kegiatan yang merupakan bagian dari analisis sistem, desain sistem, pemrograman, pengujian, dan pemeliharaan sistem serta proses pengelolaan proyek lainnya yang diperlukan untuk berhasil

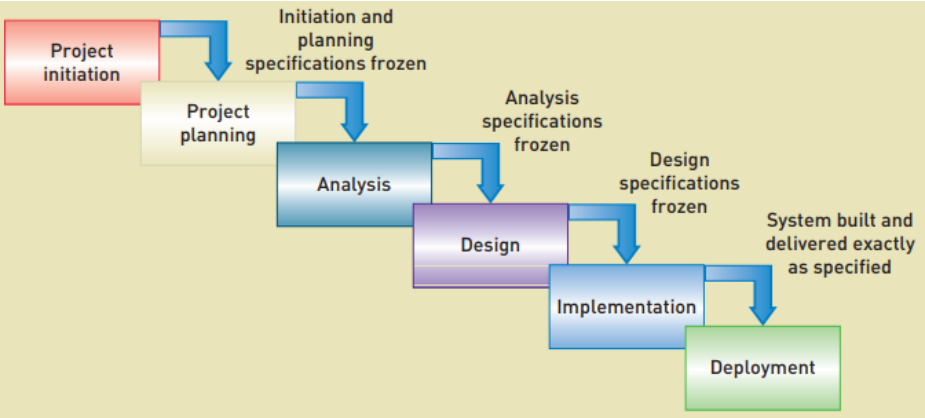
meluncurkan dan menyebarkan sistem informasi baru. Berikut adalah enam proses inti yang diperlukan dalam SDLC:

1. Mengidentifikasi masalah atau kebutuhan dan mendapatkan persetujuan untuk melanjutkan proyek.
2. Merencanakan dan memantau proyek, apa yang harus dilakukan, bagaimana melakukannya, dan siapa yang melakukannya.
3. Menemukan dan memahami detail masalah atau kebutuhannya.
4. Mendesain komponen sistem untuk memecahkan masalah atau memenuhi kebutuhan.
5. Membangun, menguji, dan mengintegrasikan komponen sistem.
6. Menyelesaikan pengujian sistem dan menggunakan solusi.

Terdapat dua tipe pendekatan SDLC yaitu prediktif dan adaptif. Tidak ada pendekatan yang terbaik, semuanya harus disesuaikan dengan kondisi lingkungan proyek. Pendekatan prediktif mengasumsikan bahwa proyek pengembangan direncanakan dengan baik dan sistem informasi yang baru dapat dikembangkan sesuai dengan rencana. Pendekatan adaptif digunakan saat kebutuhan atau persyaratan dari pengguna tidak jelas atau tidak dipahami. Pendekatan yang lebih fleksibel diperlukan untuk memungkinkan rencana proyek dimodifikasi sesuai dengan perkembangan proyek.

2.1.3. Waterfall Model

Menurut Satzinger (2012), Waterfall Model sebuah model SDLC yang mengasumsikan beberapa fase dalam proyek dapat diselesaikan secara berurutan, satu fase menuju ke fase berikutnya dan tidak diijinkan untuk kembali ke fase sebelumnya. *Waterfall Model* diibaratkan seperti air terjun yang terus mengalir dari atas sampai kebawah. Diawali dengan membuat perencanaan secara terperinci, kemudian persyaratannya ditentukan secara menyeluruh, kemudian sistem dirancang ke dalam algoritma hingga akhir, dan kemudian diprogram, diuji, dan diterapkan. *Waterfall Model* ini merupakan tipe pendekatan prediktif SDLC yang paling banyak digunakan.



Gambar 2.1. Model *Waterfall* pada SDLC

Sumber: Satzinger (2012)

Satzinger (2012), menyebutkan *Waterfall Model* terbagi menjadi 6 tahap yaitu:

1. Inisiasi, mengidentifikasi latar belakang dan permasalahan yang telah terjadi.
2. Perencanaan, yaitu tahap untuk merencanakan sistem yang ingin dibangun sesuai dengan kebutuhan berdasarkan tujuan yang ingin dicapai. Pada tahap ini perencanaan spesifikasi sistem ditetapkan untuk dilanjutkan.
3. Analisis, yaitu menganalisa proses bisnis dan kebutuhan proyek. Ini adalah tahap pengolahan data dan menetapkan *requirement* yang menjadi persyaratan dalam sistem.
4. Desain, yaitu perancangan yang dibuat secara keseluruhan untuk menjembatani antara pemahaman dengan pembangunan sistem. Hal ini bertujuan untuk menyelaraskan antara kebutuhan dengan sistem yang akan dikembangkan.
5. Implementasi atau pelaksanaan pembangunan sistem dari desain yang telah dibuat.
6. *Deployment* atau penyebaran dan penyerahan sistem yang telah selesai dibangun kepada pengguna agar dapat digunakan.

2.1.4. Object Oriented Approach

Menurut Satzinger (2012), *Object Oriented Approach* adalah sebuah pendekatan yang melihat sistem informasi sebagai kumpulan dari objek yang saling berinteraksi yang bekerja sama untuk menyelesaikan tugas. Secara konseptual, tidak ada proses atau program, tidak ada entitas data atau berkas. Sistem ini terdiri atas objek. Suatu objek adalah sesuatu dalam sistem komputer yang dapat merespon pesan. Pandangan yang sangat berbeda dari sistem komputer ini membutuhkan pendekatan yang berbeda untuk analisis sistem, desain sistem, dan pemrograman.

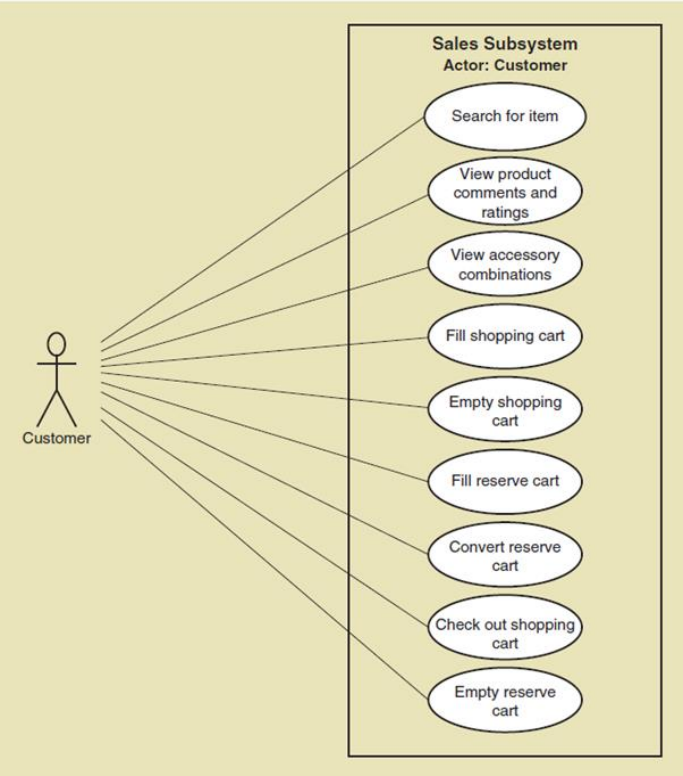
Mengingat bahwa *Object Oriented Approach* memandang sistem informasi sebagai kumpulan objek yang berinteraksi, *Object Oriented Analysis (OOA)* mendefinisikan objek yang melakukan pekerjaan dan menentukan interaksi pengguna yang disebut *use cases* yang diperlukan untuk menyelesaikan tugas. *Object Oriented Design (OOD)* mendefinisikan semua tipe tambahan objek yang diperlukan untuk berkomunikasi dengan orang dan perangkat dalam sistem, ini menunjukkan bagaimana objek berinteraksi untuk menyelesaikan tugas, dan mendefinisikan setiap jenis objek sehingga dapat diimplementasikan dengan bahasa atau lingkungan tertentu. *Object Oriented Programming (OOP)* adalah penulisan pernyataan dalam bahasa pemrograman untuk mendefinisikan apa yang setiap jenis objek dapat lakukan.

Objek adalah benda, bisa pelanggan atau karyawan atau bisa juga berupa tombol atau menu. Mengidentifikasi jenis objek berarti mengelompokkan hal-hal ke dalam klasifikasi tertentu. Beberapa hal, seperti pelanggan, ada di luar dan di dalam sistem. Ada pelanggan, yang berada di luar sistem, dan representasi komputer pelanggan, yang ada di dalam sistem.

Dalam *Object Oriented Approach* ada banyak *tools* dan *diagram* yang dapat digunakan untuk menggambarkan sistem, meliputi *Use Case Diagram*, *Activity Diagram*, *Sequence Diagram dan Class Diagram.*

2.1.5. Use Case

Use Case adalah suatu aktivitas yang dilakukan sistem, biasanya sebagai tanggapan atas permintaan oleh pengguna. Biasanya *use case* akan digambarkan dalam suatu bentuk diagram yang disebut *Use Case Diagram*, sebuah model *UML (Unified Modeling Language)* yang digunakan untuk menunjukkan secara grafis kasus penggunaan dan hubungannya dengan aktor. Selain itu *Use Case Diagram* juga dapat dijabarkan secara lebih lengkap dan mendetail melalui *Use Case Description*, sebuah model tekstual yang mencantumkan dan menjelaskan detail pemrosesan untuk setiap *Use Case*.

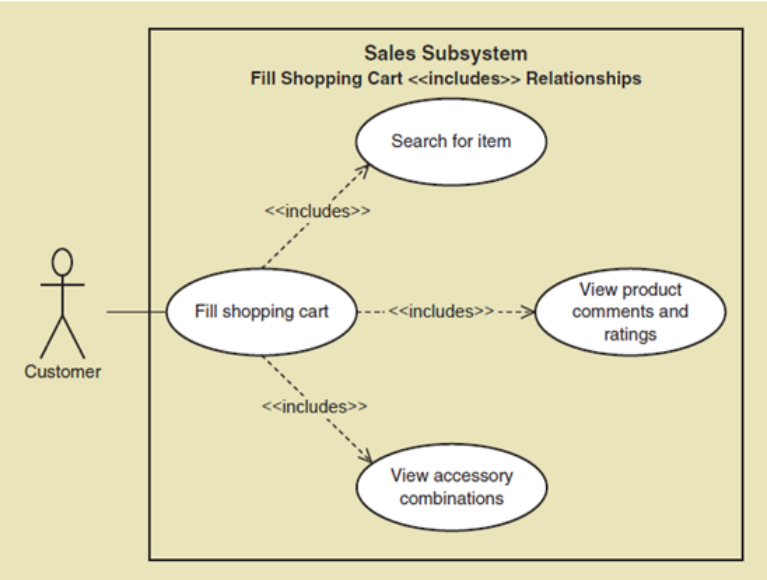


Gambar 2.2. *Use Case Diagram*

Sumber: Satzinger (2012)

Use Case Diagram pada penelitian ini menggunakan relasi *includes* atau biasa ditulis <<*includes*>>, yang menurut Satzinger (2012) <<*includes*>> adalah relasi *use case* tambahan ke sebuah *use case* dimana

use case yang ditambahkan memerlukan *use case* ini untuk menjalankan fungsinya. Hubungan antara kedua *use case* tersebut dilambangkan dengan garis putus-putus dengan simbol panah yang menunjuk ke *use case* lain yang disertakan, seperti gambar 2.3 di bawah ini. Kata <<includes>> dapat diartikan sebuah hubungan stereotip antara satu *use case* dengan *use case* lainnya. Cara membaca relasi ini adalah Mengisi keranjang belanja (*Fill shopping cart*) termasuk atau *includes* Mencari item (*Search for item*), Melihat komentar dan peringkat produk (*View product comments and ratings*), dan Melihat aksesori kombinasi (*View accessory combinations*).



Gambar 2.3. *Use Case Diagram* Menggunakan *Include* Pada Subsistem

Sumber: Satzinger (2012).

Selain relasi <<*includes*>>, penelitian ini juga menggunakan relasi <<*extends*>>, yaitu relasi *use case* tambahan ke sebuah *use case* dimana *use case* yang ditambahkan dapat berdiri sendiri tanpa harus melalui proses yang lain.

Menurut Satzinger (2012), bergantung pada kebutuhan analis, deskripsi *use case* cenderung ditulis di dua tingkat detail yang terpisah: deskripsi singkat (*brief use case description*) dan deskripsi yang

dikembangkan sepenuhnya (*fully developed use case*). *Brief use case description* dapat digunakan untuk *use case* yang lebih sederhana, terutama ketika sistem yang akan dikembangkan adalah aplikasi kecil yang dipahami dengan baik. Penggunaan *brief use case* biasanya memiliki satu skenario dan sangat sedikit kondisi pengecualian (*exception condition*). Contohnya adalah *use case* "Melihat jenis produk" atau *use case* "Masuk ke sistem atau *Log-in*".

Use case name:	*Create customer account.*	
Scenario:	Create online customer account.	
Triggering event:	New customer wants to set up account online.	
Brief description:	Online customer creates customer account by entering basic information and then following up with one or more addresses and a credit or debit card.	
Actors:	Customer.	
Related use cases:	Might be invoked by the *Check out shopping cart* use case.	
Stakeholders:	Accounting, Marketing, Sales.	
Preconditions:	Customer account subsystem must be available. Credit/debit authorization services must be available.	
Postconditions:	Customer must be created and saved. One or more Addresses must be created and saved. Credit/debit card information must be validated. Account must be created and saved. Address and Account must be associated with Customer.	
Flow of activities:	**Actor**	**System**
	1. Customer indicates desire to create customer account and enters basic customer information.	1.1 System creates a new customer. 1.2 System prompts for customer addresses.
	2. Customer enters one or more addresses.	2.1 System creates addresses. 2.2 System prompts for credit/debit card.
	3. Customer enters credit/debit card information.	3.1 System creates account. 3.2 System verifies authorization for credit/debit card. 3.3 System associates customer, address, and account. 3.4 System returns valid customer account details.
Exception conditions:	1.1 Basic customer data are incomplete. 2.1 The address isn't valid. 3.2 Credit/debit information isn't valid.	

Gambar 2.4. *Fully Developed Use Case Description*

Sumber: Satzinger (2012)

Fully Developed Use Case Description merupakan metode paling formal untuk mendokumentasikan sebuah *use case*. Salah satu kesulitan utama bagi pengembang perangkat lunak (*software developers*) adalah mendapatkan pemahaman yang mendalam tentang kebutuhan pengguna.

Akan tetapi jika pengembang memutuskan untuk membuat *Fully Developed Use Case Description*, maka peluang untuk memahami secara penuh proses bisnis dan bagaimana sistem tersebut mendukung kebutuhan pengguna akan meningkat. Contoh dari *Fully Developed Use Case Description* untuk *use case* buat akun pelanggan dapat dilihat pada gambar 2.4

Penelitian ini memilih untuk menggunakan *Fully Developed Use Case Description* sesuai dengan tingkat kerumitannya. Contoh dari Satzinger (2012) dapat menjelaskan lebih rinci mengenai *Use Case Description* untuk *use case* "Membuat akun pelanggan" atau "*Create customer account".*

1. Kompartemen pertama (*Use Case Name*) dan kedua (*Scenario*) digunakan untuk mengidentifikasi *use case* dan skenario dalam *use case* yang sedang didokumentasikan.
2. Kompartemen ketiga (*Triggering Event*) berisi peristiwa yang memicu *use case* mulai terjadi.
3. Kompartemen keempat (*Brief Description*) berisi deskripsi singkat tentang *use case* atau skenario.
4. Kompartemen kelima (*Actors*) mengidentifikasi aktor atau aktor-aktor.
5. Kompartemen keenam (*Related use cases*) mengidentifikasi *use case* lain dan menjelaskan bagaimana keterkaitannya dengan *use case* ini. Referensi silang ini untuk *use case* lainnya membantu mendokumentasikan semua aspek persyaratan pengguna.
6. Kompartemen ketujuh (*Stakeholders*) mengidentifikasi pemangku kepentingan yang merupakan pihak yang berkepentingan selain dari aktor utama. Mereka mungkin pengguna sistem yang tidak benar-benar menggunakan *use case* tetapi yang memiliki minat pada hasil yang dihasilkan dari *use case* tersebut. Contohnya pada departemen Akuntansi (*Accounting*) tertarik secara akurat untuk mengambil informasi tagihan dan kartu kredit. Meskipun di bagian departemen Pemasaran (*Marketing*) tidak ada yang membuat akun pelanggan

baru, mereka harus melakukan analisis statistik dari pelanggan baru dan juga membuat promosi pemasaran. Dengan demikian departemen pemasaran memiliki minat pada data yang ditangkap dan disimpan dari use *case* "pembuatan akun pelanggan baru". Departemen Pemasaran (*Sales*) tertarik untuk memiliki antarmuka pengguna (*user interface*) yang mudah digunakan dan menarik untuk memastikan penjualan tidak berkurang dikarenakan pengalaman pengguna (*user experience*) yang buruk. Mempertimbangkan semua pemangku kepentingan yang ada sangatlah penting bagi pengembang sistem untuk memastikan bahwa mereka telah memahami semua persyaratan bagi pengguna.

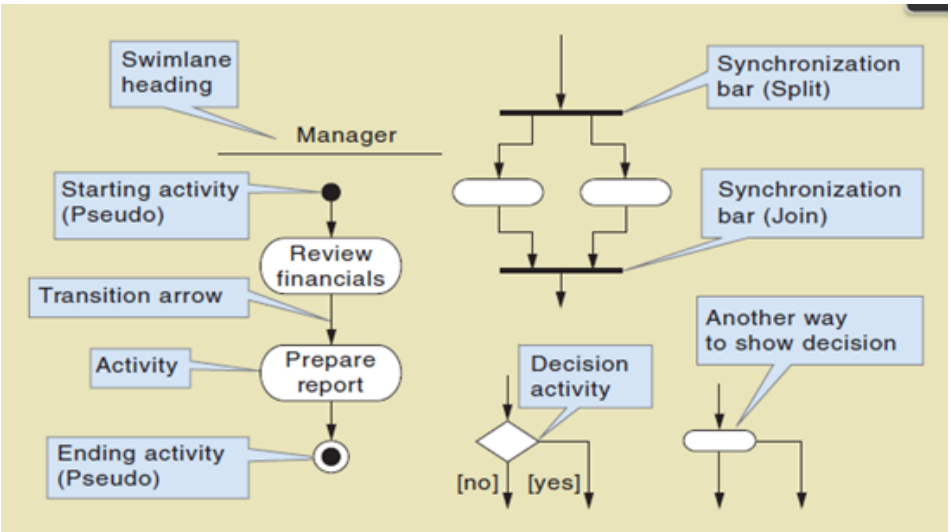
7. Kompartemen kedelapan (*Preconditions*) menyediakan informasi penting tentang keadaan sistem sebelum *use case* dieksekusi. *Preconditions* mengidentifikasi seperti apa keadaan sistem itu agar *use case* dapat dimulai, termasuk objek apa yang harus sudah ada, informasi apa harus tersedia, dan bahkan bagaimana kondisi aktor sebelum awal *use case*.
8. Kompartemen kesembilan (*Postconditions*) menyediakan informasi penting tentang keadaan sistem sesudah *use case* dieksekusi. *Postconditions* mengidentifikasi apa yang harus terjadi setelah selesainya *use case*. Yang paling penting, mereka menunjukkan objek baru apa yang dibuat atau diperbarui oleh *use case* dan bagaimana objek perlu dikaitkan. *Postconditions* sangat penting dikarenakan dua alasan. Alasan pertama *postconditions* dapat membentuk landasan untuk menyatakan hasil yang diharapkan untuk menguji *use case* setelah *use case* tersebut diimplementasikan. Kedua, objek dalam *postconditions* menunjukkan objek mana yang terlibat dalam *use case* yang penting untuk masuk dalam desain.
9. Kompartemen kesepuluh (*Flow of activities*) menggambarkan aliran terperinci dari aktivitas *use case*. Dalam hal ini, terdapat dua versi kolom yaitu mengidentifikasi langkah-langkah yang dilakukan oleh aktor dan respon yang dilakukan oleh sistem.

Nomor-nomor item membantu mengidentifikasi urutan langkah-langkah.

10. Kompartemen kesebelas menjelaskan kegiatan alternatif dan kondisi pengecualian (*Exception Condition*).

2.1.6. Activity Diagram

Ada juga *Activity Diagram*, sebuah diagram yang menjelaskan aktivitas pengguna atau sistem, orang yang melakukan setiap aktivitas, dan aliran berurutan dari aktivitas tersebut. Menurut Satzinger (2012), gambar 2.5 merupakan kumpulan simbol dasar yang dipakai dalam *Activity Diagram.*



Gambar 2.5 Notasi *Activity Diagram*

Sumber: Satzinger (2012)

Pada *activity diagram* di atas terdapat beberapa notasi di antaranya adalah sebagai berikut.

1. *Swimlane Heading* mewakili siapakah aktor yang melakukan aktivitas.
2. *Starting Activity* menjelaskan awal dari *activity diagram.*
3. *Transition Arrow* menjelaskan alur dari aktivitas.
4. *Activity* mewakili aktivitas yang dilakukan oleh setiap aktor.
5. Ending Activity menjelaskan akhir dari *activity diagram.*

6. Syncronization Bar Split menjelaskan adanya pemisahan aktivitas menjadi beberapa aktivitas yang paralel.
7. Synchronization Bar Join menjelaskan adanya mempertemukan beberapa aktivitas yang berjalan paralel menjadi aktivitas yang awalnya terpisah.
8. Decision Activity menjelaskan aktivitas yang akan dilakukan selanjutnya berdasarkan kondisi tertentu.
9. *For each loop* menjelaskan aktivitas yang sama dapat dilakukan berulang kali dimulai dari *for each item* sampai dengan *end for each item* seperti yang ditunjukkan pada gambar 2.6.

Gambar 2.6. Contoh *Activity Diagram*

Sumber: Satzinger (2012)

Activity diagram sangat membantu karena pemahaman aliran aktivitas untuk setiap *use case* akan lebih mudah dengan diagram yang sederhana dan jelas.

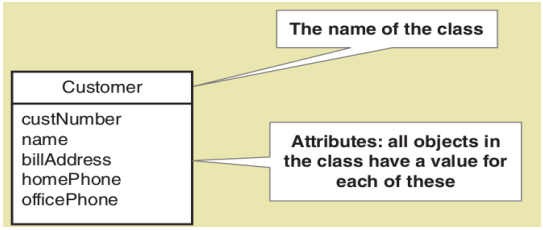
2.1.7. Class Diagram

Banyak pendekatan pengembangan sistem saat ini menggunakan istilah *class* daripada *data entity* dan menggunakan konsep dan notasi berdasarkan UML untuk memodelkan hal-hal dalam domain masalah. Konsep-konsep ini berasal dari *Object Oriented Approach*. *Class* adalah kategori atau klasifikasi yang digunakan untuk menggambarkan kumpulan *object*. Setiap *object* memiliki *class*. Contohnya adalah Daffa, Iqbal, dan Michael adalah *object* yang termasuk dalam *class* Siswa. *Class* yang menggambarkan *object* dalam domain masalah disebut *Domain Classes*. *Domain Classes* memiliki atribut dan asosiasi. *Multiplicity* (yang disebut *cardinality* dalam ERD) berlaku di antara *class*. Awalnya, ketika mendefinisikan persyaratan, pendekatan pemodelan menggunakan ERD atau UML sangat mirip.

Class Diagram digunakan untuk menunjukkan class dari *object* untuk suatu sistem serta asosiasinya diantara *class*. Salah satu jenis *Class Diagram* yang mencakup atau menunjukkan hal-hal dalam domain masalah pengguna disebut *Domain Model Class Diagram*. Jenis lain dari *Class Diagram* adalah *Design Class Diagram* yang digunakan dalam merancang *class* pada perangkat lunak.

Pada *Class Diagram*, persegi panjang mewakili *class* dan garis-garis yang menghubungkan antar persegi panjang menunjukkan asosiasi atau hubungan antar *class*. Gambar di bawah ini menunjukkan simbol untuk *Domain Model Class* tunggal yaitu *Customer*. Simbol *Domain Model Class* adalah persegi panjang dengan dua bagian. Bagian atas berisi nama *class*, dan bagian bawah daftar *attributes class*. Nama *class* dan nama *attribute* ditulis menggunakan notasi *camelback* atau *camelcase*, dimana antar kata ditulis tanpa spasi atau garis bawah. Nama *class* dimulai dengan

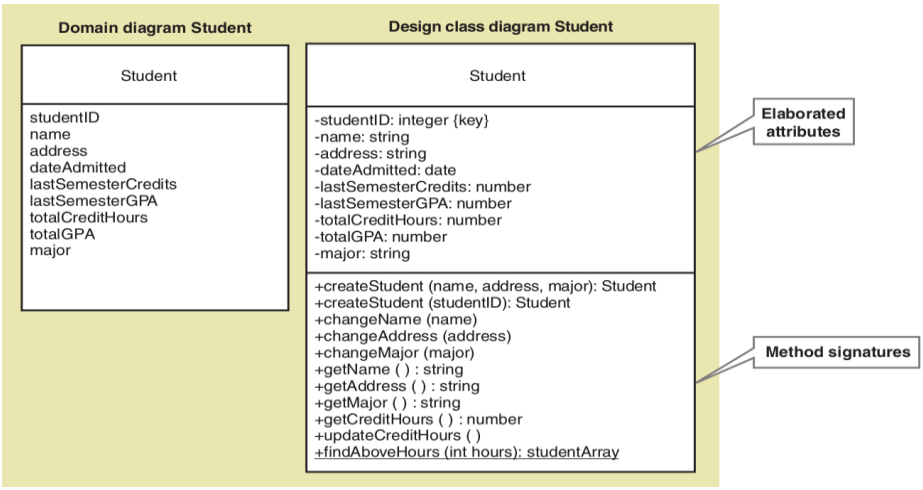
huruf kapital, nama *attribute* dimulai dengan huruf kecil seperti pada gambar 2.6.



Gambar 2.7. Notasi *Domain Class Diagram*.

Sumber: Satzinger (2012).

Sedangkan pada *Design Model Class* juga menggunakan simbol-simbol yang dimiliki *Domain Model Class* dengan tambahan bagian ketiga yaitu daftar *methods class*. Daftar *method class* tidak ada pada *Domain Model Class*.



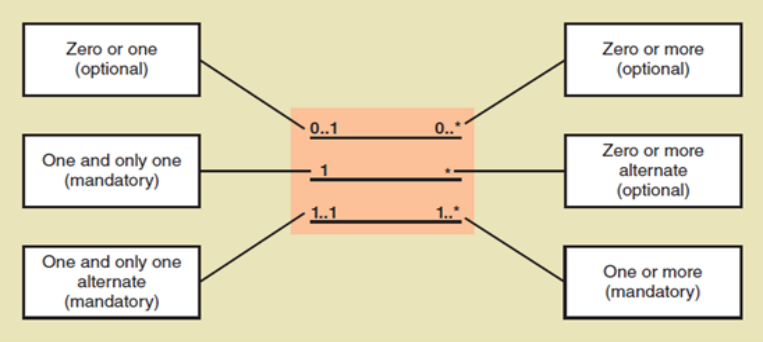
Gambar 2.8. *Domain Model Class dan Design Model Class*

Sumber: Satzinger (2012)

Selain itu *Domain Model Class Diagram* juga memiliki notasi lainnya *multiplicity* untuk menghubungkan antar *class*, yaitu:

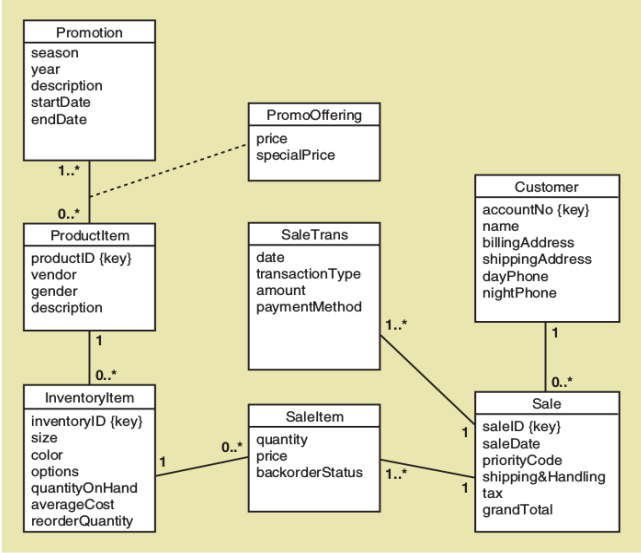
1. *Zero or one* (tidak wajib): boleh nol atau satu.
2. *One and only one* (wajib): wajib satu dan hanya satu.
3. *One and only one alternate* (wajib): wajib satu dan hanya satu alternatifnya.
4. *Zero or more* (tidak wajib): boleh nol atau lebih.
5. *Zero or more alternate* (tidak wajib): boleh nol atau lebih alternatifnya.

.



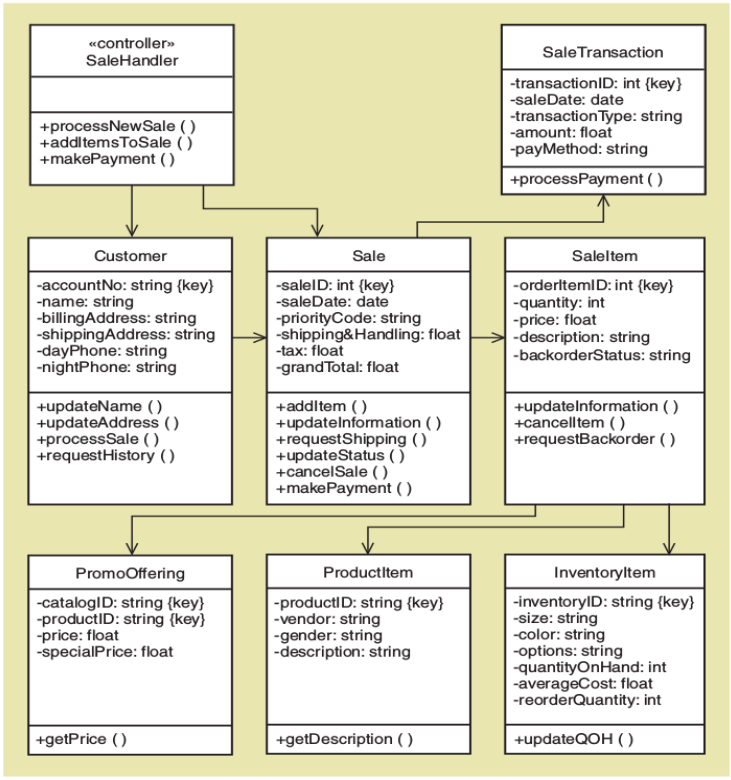
Gambar 2.9. Notasi dari *Multiplicy*

Sumber: Satzinger (2012)



Gambar 2.10. Contoh dari *Domain Model Class Diagram*

Sumber: Satzinger (2012)

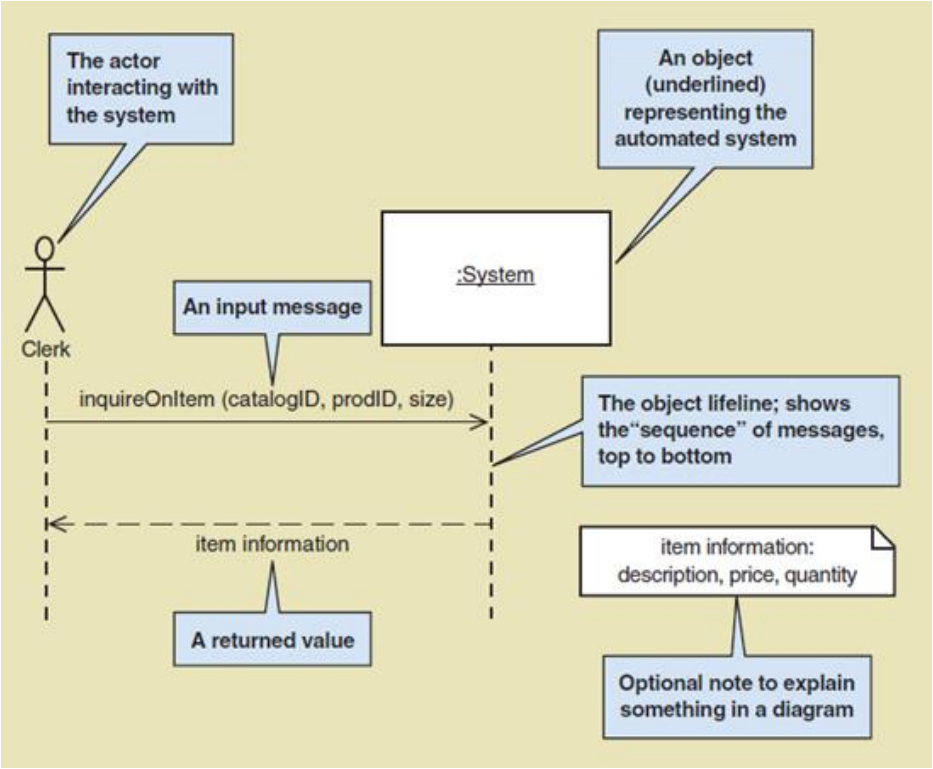


Gambar 2.11. Contoh dari *Design Model Class Diagram*

Sumber: Satzinger (2012)

2.1.8. System Sequence Diagram

Dalam pendekatan berorientasi objek, aliran informasi dapat dilakukan melalui pengiriman pesan baik ke dan dari aktor atau bolak-balik antara objek internal. *System Sequence Diagram* adalah diagram yang menunjukkan urutan pesan antara aktor eksternal dan sistem pada *use case* atau skenario. Dengan demikian, *System Sequence Diagram* mendokumentasikan input dan output serta mengidentifikasi interaksi antara aktor dan sistem. *System Sequence Diagram* merupakan salah satu jenis *Interaction Diagram. Interaction Diagram* lainnya adalah *Communication Diagram.*



Gambar 2.12. Notasi dari *System Sequence Diagram*

Sumber: Satzinger (2012)

Dalam notasi di atas, dapat dijabarkan dengan penjelasan diantaranya sebagai berikut.

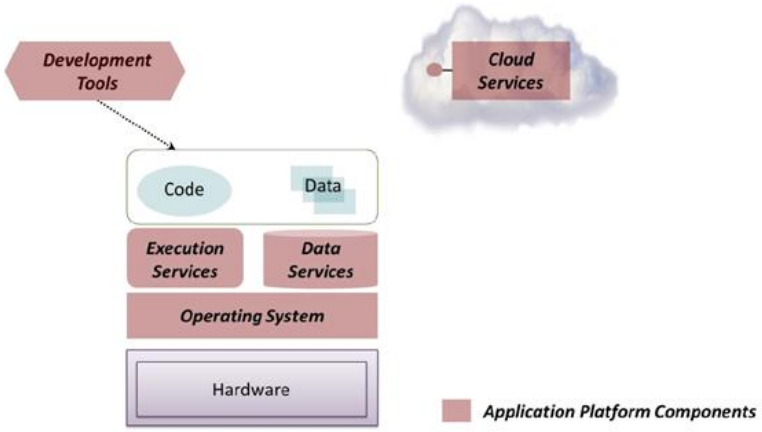
1. Aktor (actor) dalam contoh ini berperan sebagai *Clerk* yang berinteraksi dengan sistem.
2. Sistem (*system)* di sini adalah objek yang mewakili sistem otomatis.
3. Garis putus-putus vertikal (*lifeline / object lifeline*) yang berfungsi sebagai pembatas antara aktor dengan sistem yang ada.
4. Panah utuh (*solid arrow)* di antara dua garis *lifeline* mewakili pesan yang dikirim oleh aktor. Panah utuh ini berfungsi sebagai penjelas mengenai pesan-pesan atau fungsi-fungsi yang dijalankan oleh aktor kepada sistem. Pesan ini disebut dengan *input message.*

5. Panah putus-putus (*dashed arrow)* yang berasal dari system mengarah ke aktor berfungsi sebagai penjelas dari tanggapan system kepada actor.
6. *Message* adalah pesan untuk menjelaskan tujuan dari data yang dikirim.
7. *Input message* adalah pesan dari aktor ke sistem. Contoh *input message* pada gambar 2.11 di atas adalah "inquireOnItem (catalogID, prodID, size)".
8. *Return message* adalah pesan ketika sistem memberikan respon balik ke aktor. Contoh *return message* pada gambar 2.11 di atas adalah "item information".
9. *Optional note* merupakan catatan tambahan bersifat opsional (tidak wajib) yang berfungsi untuk menjelaskan lebih rinci tentang hal-hal terkait *message* pada diagram.

2.1.9. Platform

Platform merupakan istilah yang sering digunakan dalam dunia teknologi informasi. *Platform* aplikasi adalah *platform* di mana program aplikasi dapat beroperasi. *Platform* aplikasi memainkan peran dasar karena aplikasi dan data yang mereka dukung pada lapisan atasnya merupakan nilai yang diberikan kepada pengguna teknologi informasi. Menurut David Chappel (2011), *platform* aplikasi memiliki lima komponen, yaitu:

1. *Operating System*, terletak pada lapisan yang paling bawah dari *platform* dan berantarmuka dengan perangkat keras di bawahnya.
2. *Execution Service*, menyediakan *library* untuk aplikasi yang sedang aktif.
3. *Data Service*, menyediakan *storage* untuk penyimpanan pemrosesan data yang dilakukan oleh aplikasi.
4. *Cloud Service*, dari jarak jauh (*remote*) menyediakan fungsi-fungsi yang dapat dipakai oleh aplikasi.
5. *Development tools*, yang dipakai oleh untuk membuat dan mengembangkan aplikasi.



Gambar 2.13 Komponen *Platform*
Sumber: David Chappel (2011)

Sistem yang dikembangkan dalam penelitian ini memiliki dua *platform* yaitu:

1. *Web-based platform application*

   Web-based platform adalah *platform* yang terhubung dengan internet berdasarkan *open standard* dari konsorsium WWW (*World Wide Web*) yang memfasilitasi aplikasi agar dapat diakses dimanapun di dunia selama terhubung dengan jaringan internet. Sistem yang berjalan saat ini adalah salah satu aplikasi yang berbasis web. *Content Management System* (CMS) yang dibangun juga termasuk dalam kategori ini.
2. *Mobile Platform application*

   Mobile platform application digunakan untuk merancang, membuat, dan memelihara aplikasi seluler. LINE adalah salah satu aplikasi seluler yang dapat digunakan di *platform mobile* seperti iOS, Android, dan Blackberry.

2.1.10. Mobile Apps

Menurut Dave Chaffey (2015), *mobile apps* adalah perangkat lunak yang dirancang untuk digunakan pada smartphone, biasanya diunduh dari *App store* atau *Google Playstore*. Penggunaan perangkat seluler telah mengubah cara konsumen mengakses konten dan layanan *online* untuk keputusan hiburan, bersosialisasi, pendidikan dan membeli. Manfaat yang

ditawarkan oleh koneksi seluler atau nirkabel kepada pengguna adalah *ubiquity* (dapat diakses dari mana saja), *reachability* (pengguna dapat dijangkau ketika tidak berada di lokasi normal) dan *convenience* (tidak perlu memiliki akses ke catu daya atau koneksi kabel). Mereka juga memberikan keamanan dimana setiap pengguna dapat diautentikasi karena setiap perangkat nirkabel memiliki kode identifikasi unik, lokasi mereka dapat digunakan untuk menyesuaikan konten, dan dapat memberikan tingkat privasi lebih dibandingkan dengan *PC desktop*. Keuntungan lainnya adalah akses instan atau selalu aktif, di sini tidak perlu *dial up* koneksi nirkabel.

2.1.11. Antarmuka

Menurut Satzinger (2012) *user interface* atau antarmuka berperan melibatkan secara langsung kepada pengguna internal maupun eksternal. Jadi antarmuka adalah bagaimana cara program aplikasi berkomunikasi dengan pengguna. Faktor-faktor penting dalam membuat sebuah antarmuka adalah sebagai berikut.

1. Tujuan pembuatan antarmuka.
2. *Usability* yaitu tingkat kemudahan sebuah sistem untuk dipelajari dan dipakai.
3. Karakteristik dari pengguna dan dari perangkat yang ada.
4. Jelas, menarik, dan efisien.
5. Mudah dilihat (visible).
6. Responsive dan komunikatif.
7. Konsisten.

2.1.12. Pengujian atau Testing

Pengujian (*testing*) merupakan salah satu cara untuk menilai fungsionalitas daripada sebuah *software*. Proses *testing* berfungsi untuk memeriksa fungsi dari komponen *software* untuk memastikan sistem bekerja dengan benar. Dalam metode SDLC *waterfall model*, pengujian dilakukan pada tahap implementasi dan juga pada tahap *deployment*. Menurut Satzinger (2012) cara-cara pengujian perangkat lunak (*software*

testing techniques) dan pada tahapan mana pengujian itu dilakukan dapat dijelaskan di bawah ini.

1. *Unit Testing*

   Unit testing adalah proses pengujian metode individu, kelas, atau komponen sebelum mereka terintegrasi dengan perangkat lunak lain. Tujuan dari *unit testing* adalah untuk mengidentifikasi dan kemudian memperbaiki kesalahan sebanyak mungkin sebelum modul digabungkan ke unit perangkat lunak yang lebih besar, seperti program, kelas, dan subsistem.

2. *Integration Testing*

   Integration testing (tes integrasi) mengevaluasi perilaku (*behavior*) sekelompok metode, kelas, atau komponen. Tujuan dari *integration testing* adalah untuk mengidentifikasi kesalahan yang tidak dapat dideteksi oleh *unit testing*. Kesalahan seperti itu mungkin disebabkan oleh beberapa hal seperti: *interface compatibility*, *parameter values*, *run-time exception*, *unexpected state interaction.*

3. *Usability Testing*

   Usability Testing (test kegunaan) adalah tes untuk menentukan apakah suatu metode, kelas, subsistem, atau sistem memenuhi persyaratan pengguna (*user*). Karena ada banyak jenis persyaratan, seperti fungsional dan non-fungsional, maka banyak *usability testing* dilakukan di banyak waktu yang berbeda.

4. *System Testing*

   System Testing (tes sistem) adalah test perilaku seluruh sistem secara terintegrasi. *System testing* yang dilakukan bersama pengguna atau calon pengguna biasanya disebut *User Acceptance Testing* (UAT).

2.1.13. Media Sosial

Menurut Dave Chaffey (2015), media sosial adalah kategori media yang memfokuskan pada partisipasi dan komunikasi *peer-to-peer* antar individu, dengan situs yang menyediakan kemampuan untuk mengembangkan konten yang dibuat pengguna atau dikenal dengan *User*

Generated Content (UGC) dan untuk bertukar pesan dan komentar di antara pengguna yang berbeda.

Menurut Shabnoor Siddiqui (2016), pada saat ini media sosial telah menjadi bagian penting hidup seseorang dari belanja, pendidikan dan alat bisnis. Media sosial memainkan peran penting di dalamnya sehingga mengubah gaya hidup masyarakat.

Peradaban manusia sedang mengalami revolusi informasi yang lebih besar daripada Revolusi Industri. Revolusi ini tidak saja merubah bagaimana kita bekerja maupun berkomunikasi, tetapi juga bagaimana berhubungan satu sama lain dengan adanya media sosial di internet. Media sosial di *internet* adalah sebuah situs media dipakai untuk bersosialisasi secara *online* dan ini memungkinkan manusia untuk saling berinteraksi tanpa dibatasi oleh ruang dan waktu. Situs media sosial terkemuka dan memiliki anggota yang besar diantaranya adalah Facebook, Twitter, LinkedIn, Instagram, dan Snapchat. Sedangkan *platform* yang dipakai di perangkat seluler contohnya dan LINE.

Media sosial dipakai oleh masyarakat untuk mendapatkan informasi politik seperti hasil debat capres dan cawapres. Twitter contohnya dipakai oleh politikus untuk menggalang dukungan dan menyampaikan pesan-pesan politiknya. Media sosial juga menjadi sumber utama mendapatkan berita sehari-hari dan menggantikan media tradisional seperti surat kabar dan majalah. Banyak dampak positif dari penggunaan media sosial termasuk tumbuhnya minat para individu untuk mengembangkan profesi dan pendidikannya dengan terlibat di komunitas maya yang tepat.

Semua dapat diakses gratis dan *non-stop* 24 jam sehari. Selain dampak positif, ada juga dampak negatif dari media sosial terutama untuk kalangan remaja yang mengakses media sosial melalui *smartphone* dan mereka sangat intensif dalam menggunakan layanan ini.

2.1.14. Media Chatting LINE

Menurut Kefa Olang (2018), definisi chatting melalui internet adalah peristiwa pertukaran pesan antara dua orang atau lebih secara langsung di kamar *chat*. Dengan teknologi jaringan komputer maka pengguna dapat

mengirimkan pesan kepada sesama pengguna yang sedang *online*. Pesan tidak hanya dalam bentuk teks tapi juga dalam bentuk foto, video, stiker atau *emoticon*, dan file dokumen. *Chatting* tidak hanya populer dikalangan anak muda saja tetapi jaman sekarang ini sudah merambah ke kalangan orang tua sekalipun. Dengan adanya chatting ini kita dapat dengan bebas mengobrol mengenai apa saja mulai dari persahabatan, pekerjaan, pelajaran di sekolah, mata kuliah, sampai masalah percintaan dan perjodohan, bahkan sampai dengan hal-hal bersifat pribadi sekali pun.

LINE adalah aplikasi *mobile messenger* yang paling cepat berkembang di dunia. LINE dapat menghubungkan pengguna ke orang-orang yang dikenalnya dan dapat menerima informasi berita terkini. LINE tersedia di smartphone dengan sistem operasi Android dan iPhone. Menurut situs resmi LINE (https://line.me) beberapa fitur layanan yang disediakan adalah sebagai berikut.

1. Fitur chat ke satu orang maupun banyak orang (*multiple chat*) sampai 200 orang. Di sini orang bisa berbagi pesan tulisan, pesan suara, foto, video, stiker, dan berbagi lokasi.
2. Fitur panggilan suara gratis (*free voice call*) dan video gratis (*free video call*) ke satu orang
3. Fitur *Group Call*, yaitu panggilan suara dan video ke banyak orang
4. Fitur *International Call* atau panggilan internasional ke nomor telepon atau ponsel dengan menggunakan LINE *Out*.
5. Toko stiker.
6. Tempat penyimpanan file pribadi (*personal storage space*).
7. Terhubung ke *gateway* aplikasi kategori hiburan dan gaya hidup.

2.1.15. Chatbot

Menurut Rashid Khan (2018), *chatbot* atau *chatterbot* adalah program komputer yang memiliki kemampuan untuk memproses masukan dari pengguna dan menghasilkan tanggapan yang kemudian dikirim kembali ke pengguna. *Chatbot* memiliki kemampuan untuk menangani percakapan antara mesin dan manusia menggunakan bahasa yang dimengerti oleh manusia. *Chatbot* merupakan sebuah layanan yang

didukung oleh mesin berbasis aturan *(rules-driven)* ataupun mesin berbasis kecerdasan buatan *(Artificial Intelligent),* dimana pengguna dapat berinteraksi melalui antarmuka berbasis teks atau percakapan. Setelah *chatbot* memahami apa yang dimaksudkan pengguna, ia mengambil informasi yang diperlukan dengan menjalankan *API* atau melakukan pencarian ke dalam basis data.

Chatbot memiliki beragam jenis bergantung dengan tujuan dan jenis layanan yang disediakan contohnya respon dari tombol pemicu yang dapat memunculkan informasi sampai dengan program *machine learning* yang dapat membuat komputer memberikan respon yang tak terbatas berdasarkan data yang dimilikinya sebelumnya.

Chatbot sebagai program komputer yang independen dapat diintegrasikan pada salah satu dari beberapa *platform* perpesanan yang telah dibuka untuk pengembang melalui *API* seperti LINE, Facebook Messenger, Slack, Skype, dan *platform* perpesanan lainnya.

Chatbot yang akan menjadi objek penelitian ini adalah *chatbot* berbasis aturan *(rules-driven)* yang memiliki fungsi sebagai penghubung antara *Content Management System* dengan aplikasi LINE milik siswa. *Bot* pada sistem yang diciptakan belum didukung oleh kecerdasan buatan, namun hanya berisikan pemicu berupa kata kunci yang telah disiapkan. Kata kunci tersebut dapat berupa tombol dan gambar interaktif agar mengurangi adanya kesalahan *input*, proses, dan *output*. *Chatbot* ini tidak memiliki tujuan untuk membuat guru *virtual* yang dapat menjawab setiap pertanyaan siswa, melainkan sebagai asisten yang membantu siswa untuk belajar dan latihan Ujian Nasional dengan memberikan materi dan soal yang tersedia dalam sistem.

Dalam pendaftaran bot pada aplikasi LINE diharuskan memiliki nama oleh sebab itu *chatbot* yang akan dikembangkan diberi nama Rogu yang memiliki arti robot guru.

2.1.16. Content Management System (CMS)

Menurut Lyza Danger (2012), *Content Management System* adalah sebuah sistem yang membuat pengguna dapat menambahkan,

memperbarui, menghapus, dan mengelola konten web melalui sebuah *web browser* tanpa perlu memahami pemrograman seperti HTML dan bahasa pemrograman lainnya.

CMS memiliki manfaat dalam mengelola data, mengatur siklus hidup layanan, mendukung standarisasi, personalisasi, dan akuntabilitas. Manfaat itu sangat berguna bagi pihak pengguna, sistem, dan pemilik layanan dalam bertransaksi data secara mandiri namun terkoordinir oleh sistem aplikasi.

2.1.17. Application Programming Interface (API)

Menurut Lyza Danger (2012), *Application Programming Interface* adalah sebuah antarmuka yang didefinisikan dengan jelas dan sistematis yang dibuat dengan tujuan supaya sistem perangkat lunak yang berbeda dapat berbicara satu sama lain. Contoh API yang populer pada *web* adalah API Twitter. API Twitter mendefinisikan seperangkat metode yang dapat digunakan oleh programmer *web* untuk mengambil dan mengubah data pada sistem Twitter.

API bukanlah sepotong perangkat lunak melainkan merupakan sepotong kumpulan kode yang diciptakan dan mempresentasikan suatu kolaborasi untuk menghubungkan perangkat tersebut dengan *server* yang telah tersedia. Keuntungan dari hal ini adalah tidak diperlukan membuka *database* secara penuh untuk membuat koneksi ke aplikasi yang lain.

Penggunaan aplikasi *third party* tidak memungkinkan untuk mengakses sistem secara menyeluruh. Efektifitas dari API membuat aplikasi mengintegrasikan data *input* dan *output* dengan *server* secara aman. Proses transaksi data hanya pada *record database* tertentu.

2.1.18. Database Management System (DBMS)

Database dan *Database Management System* adalah komponen penting dari sistem informasi modern. Menurut Satzinger (2012), *Database Management System* berperan dalam menyediakan perancang, programmer, dan pengguna akhir dengan kemampuan canggih untuk menyimpan, mengambil, dan mengelola data. Berbagi dan mengelola sejumlah besar data yang dibutuhkan oleh organisasi modern tidak akan

mungkin tanpa DBMS. *Class Diagram* pada *Object Oriented Approach* merupakan model konseptual dari informasi yang dimanipulasi dan disimpan oleh sistem informasi. Untuk merancang dan membangun sistem, pengembang harus mengubah model konseptual menjadi model desain yang lebih detail. Dibutuhkan transformasi dari *class diagram* ke dalam model *database* terperinci dan mengimplementasikan model itu dengan menggunakan *DBMS*.

Database adalah kumpulan data yang tersimpan terintegrasi yang dikelola dan dikontrol secara terpusat. *Database* dikelola dan dikendalikan oleh sistem DBMS. *Database* dapat diibaratkan sebagai jantung atau pusat dari organisasi modern, mendukung aktivitas internal dan interaksi dengan pemasok, pelanggan, dan pihak eksternal lainnya. Perancang *database* mengeluarkan banyak upaya untuk memastikan bahwa *database* berisi data yang benar dan lengkap dan bahwa *database* dan sistem yang berinteraksi dengannya dapat diakses dengan aman. Pertukaran data dengan melihat aspek kebenaran, kelengkapan, aksesibilitas, dan keamanan adalah masalah kompleks yang harus ditangani dengan sistem pengendalian dan tindakan keamanan yang dirancang dengan hati-hati.

Keunggulan adanya DBMS sebagai media pengelola data adalah praktis, cepat, dan *up-to-date*, selain itu penggunaan DBMS juga sangat luas. Pada dasarnya DBMS berguna sebagai media penyimpanan data yang tersusun dengan rapi sehingga aplikasi dapat mengakses dan bertransaksi data dengan aman dan cepat tanpa khawatir adanya kesalahan atau bahkan kerusakan data dan informasi yang berguna bagi pengguna.

Contoh DBMS diantaranya *Microsoft SQL Server*, Oracle, dan MySQL. MySQL merupakan salah satu *platform* DBMS yang paling sering digunakan oleh programmer di dunia. Hampir semua aplikasi berbasis web ataupun aplikasi berbasis *desktop* menggunakan *platform database* gratis ini. DBMS ini mendukung banyak aplikasi lain dan menjadi ketergantungan dalam sistem. Contohnya *Apache* dan PHP sebagai *web service*, *Java* di aplikasi *Android*, atau bahasa pemrograman lain di berbagai *platform* teknologi saat ini. Kemudahan mengintegrasikan MySQL dengan *platform* lain menjadikan MySQL salah satu pilihan

terbaik bagi programmer dalam mendukung pengembangan sebuah sistem.

2.2. Literature Review

Table 2.1 *Literature Review* 1

Judul Penelitian	*Chatbot in Education. A passing trend or a valuable pedagogical tool?*
Peneliti	Sofie Roos
Tahun	2018
Variable Terkait	Fungsi chatbot dalam edukasi
Hasil/Temuan	*Chatbot* dapat digunakan sebagai tutor, *evaluator* siswa, berkomunikasi dengan seorang guru (secara *virtual*). Kemampuan *chatbot* bisa diperluas dengan memasukkannya ke sistem lain seperti sistem *e-learning* dan sistem perpustakaan.
Persamaan	*Chatbot* yang dibangun bermaksud untuk memberikan pelayanan belajar mandiri seakan berkomunikasi dengan guru.
Keterbatasan	Studi ini menggunakan pendekatan kualitatif dan dengan melakukan tinjauan pustaka, sehingga diakui oleh penulisnya mungkin terdapat bias dalam hasil analisanya.

Table 2.2. *Literature Review* 2

Judul Penelitian	*Leveraging chatbots to improve self-guided learning through conversational quizzes*
Peneliti	Juanan Prereira

Tahun	2016
Variable Terkait	Penggunaan chatbot dalam kuis
Hasil/Temuan	Dalam dunia e-*learning*, pembelajaran oleh siswa dapat difasilitasi dengan menggunakan *chatbot*. Jurnal tersebut fokus pada penggunaan *chatbot* dalam bentuk kuis pertanyaan pilihan ganda.
Persamaan	Penelitian ini juga akan memanfaatkan penggunaan *chatbot* sebagai media pembelajaran melalui soal-soal kuis yang dapat diakses oleh siswa.
Keterbatasan	1. Tidak dipakainya kata sandi (*password*), sehingga bisa dipertanyakan siapa yang mengerjakan kuis. 2. Statistik uji coba menunjukkan tidak adanya korelasi positif antara nilai siswa di ujian akhir dengan frekuensi siswa mengerjakan soal kuis di bot. Ini mengimplikasikan bahwa latihan soal kuis saja tidak cukup, dan siswa juga perlu membaca buku materi mata pelajaran (fasilitas ini yang tidak tersedia di sistem chatbot penelitian).

Table 2.3. *Literature Review* 3

Judul Penelitian	*Experiences and perspectives of Technology-enhanced learning and teaching in higher education – Serbian case*
Peneliti	Mirjana Ivanovic, Aleksandra Klasnja Milicevic
Tahun	2018
Variable Terkait	Modernisasi proses pendidikan

Hasil/Temuan	Studi kasus khusus dari penerapan media sosial, pembelajaran berbasis permainan dan berbagai alat pembelajaran yang disempurnakan teknologi di berbagai kursus di beberapa institusi Serbia. Sistem dikembangkan secara in-house. Pengalaman menggunakan berbagai alat dan mekanisme pembelajaran yang ditingkatkan teknologi menunjukkan bahwa proses pendidikan harus dimodernisasi dan ditingkatkan oleh kemajuan teknologi.
Persamaan	Penelitian ini juga menerapkan teknologi dengan dukungan media sosial untuk menghasilkan pembelajaran yang lebih menarik, interaktif, dan modern.
Keterbatasan	Beragamnya TEL (*Technology Enhanced Learning*) *tools* yang merupakan program aplikasi sebagai alat pendukung pembelajaran ternyata membutuhkan pelatihan khusus buat staf akademis yang kurang termotivasi karena sudah cukup berat beban kerja rutin yang dihadapinya

Penelitian ini mengadopsi jurnal penelitian dari Juanan Prereira berjudul *Leveraging chatbots to improve self-guided learning through conversational quizzes* karena konsep untuk mengerjakan kuis pilihan ganda menggunakan chatbot sesuai dengan usulan solusi yang akan dibangun. Sedangkan pembaharuan penelitian dari jurnal yang dipilih ini adalah akan ditambahkan fitur akses materi pelajaran dan fitur forum diskusi. Fitur-fitur ini tidak ada dalam sistem aplikasi chatbot @DAWEBOT di Telegram pada jurnal di atas.

Untuk mengukur minat dalam belajar maka perlu dikembangkan kemampuan mencatat aktivitas siswa dalam sistem, autentikasi sangat diperlukan untuk membedakan antara aktivitas siswa satu dengan siswa yang lain. Aktivitas belajar, diskusi, maupun latihan Ujian Nasional memerlukan

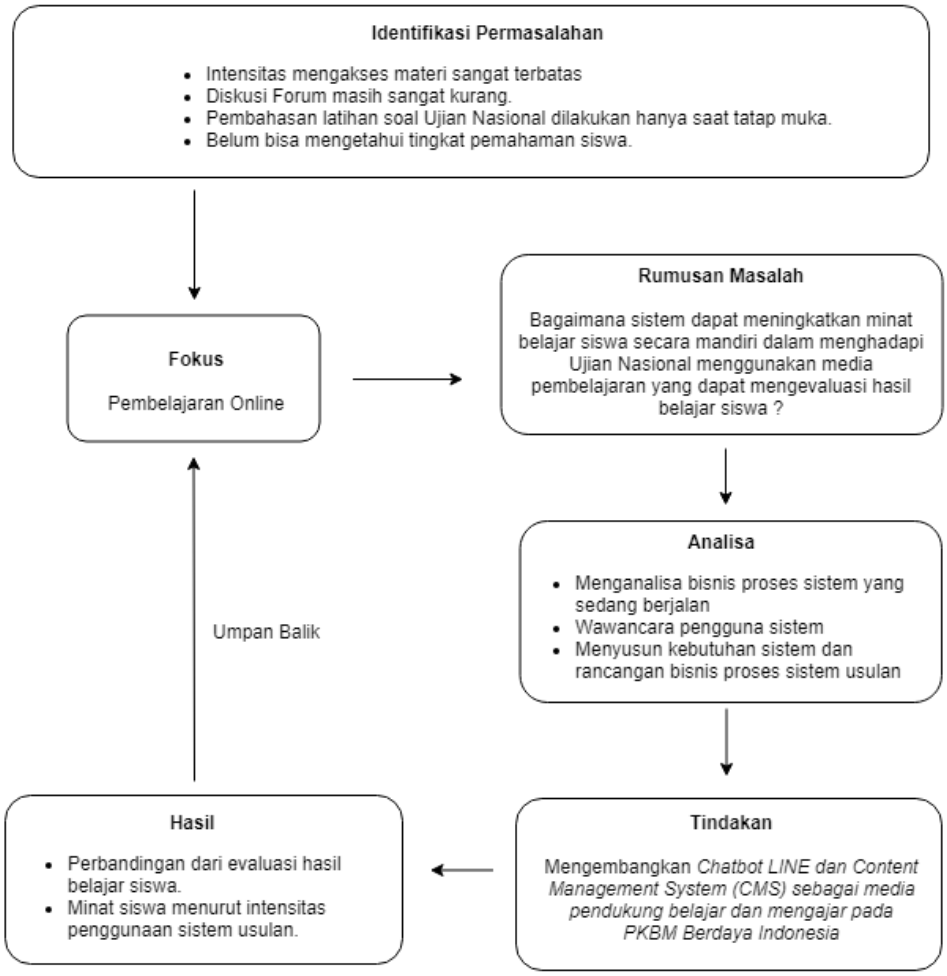
session yang berbeda di setiap siswa. Fitur ini juga belum tersedia pada chatbot jurnal di atas. Secara teknis chatbot menerima kode unik seperti NISN yang dikirimkan oleh siswa satu kali saja lalu disimpan pada *cache* sebagai identitas siswa untuk autentikasi saat menggunakan sistem. Sehingga pada saat terjadi sebuah perintah dan transaksi data tersisipkan data siapa aktor yang menjadi penggunanya.

Pembaharuan penelitian ini adalah menggunakan Redis sebagai *database* yang memiliki fungsi sebagai *cache* yang menyimpan ID siswa untuk membedakan aktivitas dan transaksi data antar siswa. Redis ini mengelola *cache* menggunakan struktur data seperti *database* dan ditempatkan di *memory server* sehingga memiliki performa yang lebih cepat dibanding *database* yang berada di *storage hardisk*. Selain sebagai autentikasi, Redis dapat difungsikan untuk mengacak pertanyaan pada latihan Ujian Nasional. Dengan adanya Redis, sistem tidak perlu mengakses dan membuat *query* terus menerus melainkan *query* tersebut dibuat sekali lalu dikirimkan dan diolah oleh Redis.

Pembaharuan yang lainnya adalah menu navigasi yang dinamis. Sistem menyediakan menu yang mempermudah siswa dalam menggunakan fitur-fitur chatbot. Navigasi akan berubah mengikuti posisi pengguna pada aktivitas tertentu seperti saat login akan memunculkan navigasi X lalu saat membaca materi akan memunculkan navigasi Y.

2.3. Kerangka Pikir

Kerangka berpikir pada penelitian ini berisi alur penggambaran proses secara keseluruhan dari awal sampai akhir dalam menganalisa dan mengembangkan sistem.



Gambar 2.14 Kerangka Pikir Penelitian