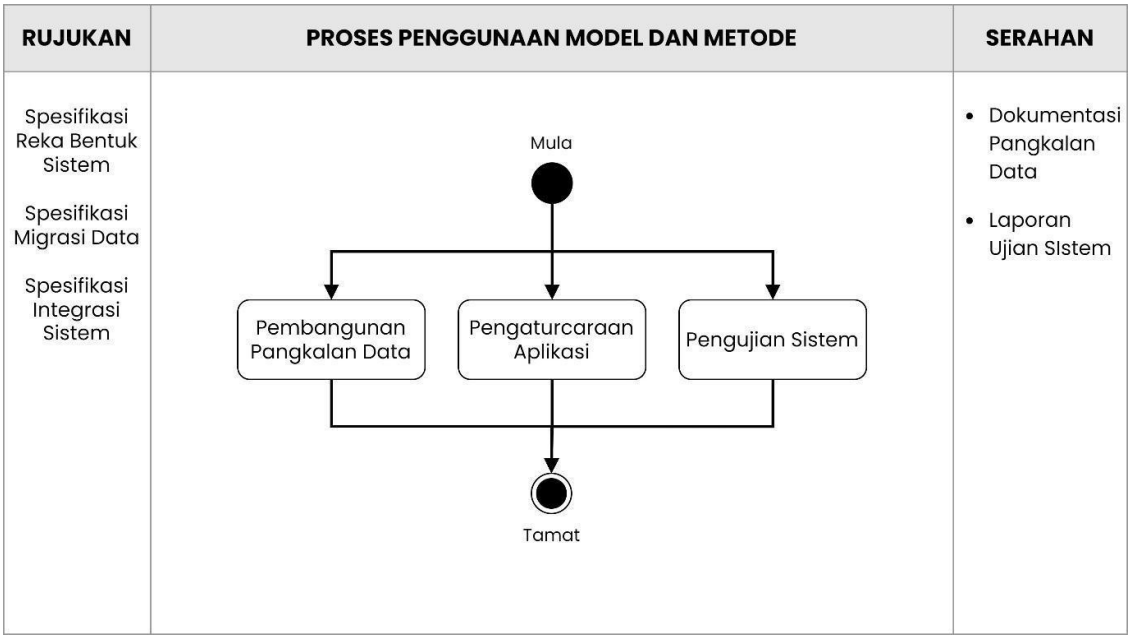
# 5 FASA PEMBANGUNAN

## 5.1 GAMBARAN KESELURUHAN




**Rajah 5.1 : Gambaran Keseluruhan Fasa 4 - Pembangunan**

## 5.2. PENGENALAN

Fasa pembangunan merupakan fasa yang sangat penting dalam Kitar Hayat Pembangunan Sistem (SDLC) di mana pada fasa inilah bermulanya aktiviti pembinaan sistem aplikasi yang ingin dibangunkan. Pada fasa ini, keperluan dan reka bentuk fungsi-fungsi sistem yang akan dibangun telah pun dimuktamatkan. Pelaksanaan aktiviti-aktiviti dalam fasa ini memerlukan kerjasama erat antara pasukan pembangunan bagi memastikan setiap modul berfungsi dengan baik dan selaras dengan spesifikasi yang telah ditetapkan.

Matlamat utama fasa ini ialah untuk menterjemah dan merealisasikan Spesifikasi Reka Bentuk Sistem (SDS), Spesifikasi Migrasi Data, serta Spesifikasi Integrasi Sistem yang dihasilkan dalam fasa reka bentuk ke dalam kod pengaturcaraan menggunakan teknologi yang telah dipilih. Selain itu, fasa ini turut melibatkan pelaksanaan pengujian sistem bagi memastikan sistem yang dibangunkan bebas daripada sebarang ralat, berfungsi sepenuhnya, dan memenuhi keperluan pembangunan sebenar. Proses ini memastikan sistem mencapai kriteria kualiti yang ditetapkan serta meningkatkan tahap keyakinan pengguna terhadap produk yang dihasilkan.

Tiga (3) aktiviti utama dalam Fasa Pembangunan adalah seperti berikut:

a) Pembangunan Pangkalan Data

b) Pengaturcaraan Aplikasi

c) Pengujian Sistem

**Dokumen Rujukan** kepada Fasa Pembangunan adalah seperti berikut:

a) D05 Spesifikasi Reka bentuk Sistem

b) D06 Spesifikasi Migrasi Data

c) D07 Spesifikasi Integrasi Sistem

**Dokumen Serahan** kepada Fasa Pembangunan adalah seperti berikut:

a) D08 Dokumentasi Pangkalan Data

b) D09 Laporan Ujian Sistem

### 5.3 PENGLIBATAN PEMEGANG TARUH

Pemegang taruh utama yang terlibat dalam fasa pembangunan adalah Pengaturcara, Penguji dan Juruanalisis Sistem. Pengaturcara bertanggungjawab dalam penulisan kod pengaturcaraan berdasarkan kepada reka bentuk yang telah disediakan, manakala penguji memastikan setiap modul yang dibangunkan berfungsi dengan betul dan bebas daripada ralat.

Cadangan penglibatan kategori pemegang taruh adalah seperti berikut:

a) **Pengaturcara** yang bertanggungjawab menulis, membangunkan, dan menguji kod pengaturcaraan berdasarkan reka bentuk sistem yang telah disediakan.

b) **Penguji** yang berperanan menjalankan ujian sistem bagi memastikan setiap fungsi beroperasi dengan betul serta bebas daripada ralat sebelum sistem diserahkan untuk digunakan.

c) **Juruanalisis Sistem** yang bertanggungjawab untuk memberikan penerangan serta menjadi pakar rujuk kepada pasukan pengaturcara mengenai reka bentuk sistem yang telah dihasilkan, selain menjadi penghubung di antara pasukan pembangunan dan pihak berkepentingan lain untuk memastikan pelaksanaan pembangunan mengikut keperluan dan spesifikasi yang ditetapkan.

d) **Pengurus Projek** yang memastikan penyelarasan, pemantauan kemajuan, serta pematuhan kepada skop dan jadual projek bagi menjamin keberhasilan proses pembangunan.

## 5.4 FAKTOR KEJAYAAN

Untuk memastikan aktiviti pembangunan berjaya dilaksanakan, faktor-faktor kejayaan utama yang perlu dipertimbangkan sebelum dan semasa aktiviti dilaksanakan adalah seperti berikut:

a) **D05 Spesifikasi Reka Bentuk Sistem (SDS)** adalah lengkap dan perlu mendapatkan pengesahan pemegang-pemengang taruh yang terlibat.

b) Pasukan pengaturcara berupaya menterjemahkan SDS kepada kod dan logik pengaturcaraan, mempunyai kemahiran dan kepakaran dalam bahasa pengaturcaraan yang telah ditetapkan.

c) Juruanalisis sistem berkemampuan memberikan penerangan dan menjadi pakar rujuk pasukan pengaturcara mengenai reka bentuk sistem yang telah dihasilkan, dan menjadi penghubung antara pasukan pembangunan dan pihak berkepentingan lain.

d) Bilangan pengaturcara yang mencukupi dan bersesuaian dengan jangka masa pembangunan.

e) Kelengkapan *tools* dan persekitaran pembangunan yang sempurna.

f) **D08 Dokumentasi Pangkalan Data** dan **D09 Laporan Ujian Sistem** disemak dan mendapat pengesahan pemegang-pemegang taruh yang ditetapkan oleh organisasi.

## 5.5 PEMBANGUNAN PANGKALAN DATA [F4.1]

### 5.5.1 PENGENALAN

Pembangunan pangkalan data adalah proses membina pangkalan data fizikal berdasarkan reka bentuk logikal dan seni bina pangkalan data yang telah dirancang. Proses ini melibatkan pembangunan struktur data yang sesuai bagi memastikan data dapat disimpan, dicapai, dan diurus dengan cekap.

Model data fizikal pula menentukan bagaimana data disusun, disimpan, dan dicapai dalam sistem pangkalan data. Pembangunan model data ini bergantung kepada jenis pangkalan data yang dipilih, sama ada Pangkalan Data Hubungan (Relational Database Management System, RDBMS), Pangkalan Data NoSQL, atau Pangkalan Data Berorientasikan Objek (OODBMS). Pemilihan model yang bersesuaian bergantung kepada keperluan sistem dan jenis data yang dikendalikan.

Pembangunan Pangkalan Data dilaksanakan oleh Pentadbir Pangkalan Data atau lebih dikenali sebagai *Database Administrator* (DBA). Terdapat beberapa bahasa pengisytiharan yang boleh digunakan untuk membangunkan pangkalan data seperti contoh yang berikut:-

a) Structured Query Language (SQL) – digunakan pada Pangkalan Data Hubungan (RDMS) seperti MySQL, PostgreSQL, MariaDB, dan ORACLE bagi tujuan manipulasi dan pengurusan data;

b) MongoDB Query Language (MQL) dan Cassandra Query Language (CQL) – yang digunakan pada pangkalan data NoSQL seperti MongoDB dan Apache Cassandra.

Buku panduan ini hanya mengkhususkan kepada pembangunan Pangkalan Data Hubungan (RDBMS) dan Pangkalan Data NoSQL sahaja.

### 5.5.2 OBJEKTIF

Membangunkan pangkalan data fizikal RDBMS dan NoSQL untuk pembangunan sistem.

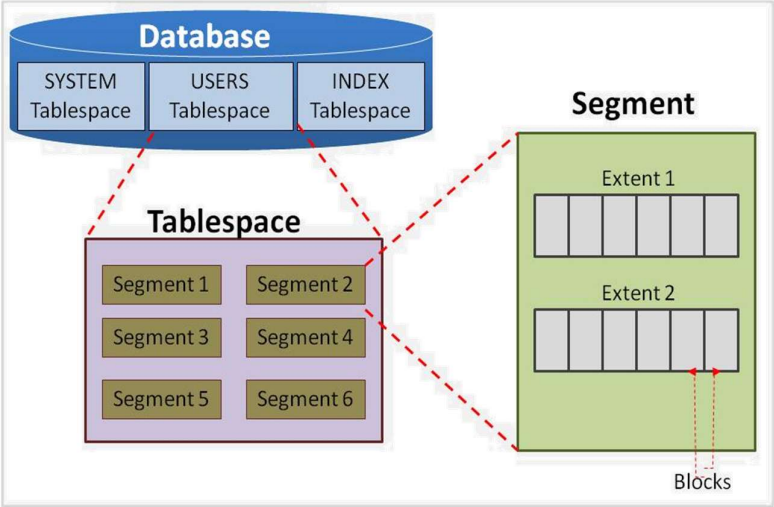
### 5.5.3 LANGKAH-LANGKAH PEMBANGUNAN PERISIAN RDBMS

**Langkah 1 : Pemasangan (*Install*) Perisian RDBMS**

Bagi membangunkan pangkalan data fizikal, perisian RDBMS yang dipilih perlu dipasang terlebih dahulu. Sekiranya MySQL yang dipilih, perisian tersebut boleh dimuat turun daripada laman web MySQL dan rujuk dokumentasi pemasangan langkah demi langkah daripada laman web yang sama.

**Langkah 2 : Peruntukan Ruang Storan**

a) Struktur storan pangkalan data terdiri daripada struktur storan fizikal dan struktur storan logikal. Struktur fizikal terdiri fail-fail seperti *datafiles, redo log files* dan *control files.* Manakala struktur logikal pula terdiri daripada beberapa *tablespace* iaitu ruang sebenar bagi menyimpan beberapa *datafile*.




**Rajah 5.2 : Struktur Storan Secara Logikal Dalam Pangkalan Data**

Berdasarkan rajah di atas, *tablespace* adalah storan logikal yang mengandungi beberapa *segment. Segment* adalah objek pangkalan data yang terdiri daripada jadual-jadual dan *index*. Satu *segment* mengandungi beberapa set *extent* yang diperuntukkan bagi objek pangkalan data secara spesifik seperti jadual. Setiap *extent* terdiri daripada beberapa data *block* yang diperuntukkan untuk menyimpan data yang spesifik. Satu data *block* adalah ruang cakera yang spesifik kepada jumlah bait (*byte)*. Sebagai contoh, satu data *block* adalah ruang cakera fizikal berjumlah 2KB. Peruntukan bagi 24KB dalam satu *extent,* sebanyak 12 data *block* diperlukan. Data *block* adalah unit terkecil dalam storan pangkalan data yang diperuntukkan untuk menyimpan *datafile* secara fizikal.

b) Ruang storan adalah lokasi simpanan bagi data sebenar yang terdapat dalam objek pangkalan data. Ruang storan hanya memperuntukkan lokasi simpanan bagi pangkalan data. Dengan menggunakan ruang storan, DBA boleh mengawal bagaimana reka bentuk penyimpanan dilakukan semasa proses pembangunan pangkalan data. Fungsi penggunaan ruang storan adalah untuk mengoptimumkan prestasi pangkalan data iaitu seperti pengasingan lokasi atau jenis cakera bagi penyimpanan jenis capaian ke atas data. Iaitu seperti data yang kerap diindeks atau dicapai perlu disimpan dalam cakera yang lebih stabil seperti *solid-state drive* (SSD) manakala data yang mengandungi data atau fail arkib disimpan dalam cakera yang biasa seperti *standard hard-disk drive* (HDD).

**Langkah 3 : Ciptakan Pangkalan Data (*Create A Database*)**

a) Pangkalan data diperlukan untuk penyimpanan dan pengurusan data. Ia membolehkan perancangan struktur data dapat dilaksanakan secara optimum bagi memenuhi keperluan aplikasi.

b) Hanya pengguna yang mempunyai capaian *root* atau *superuser* sahaja dibenarkan untuk mencipta pangkalan data.

c) Namakan pangkalan data yang dibina menggunakan nama yang deskriptif yang menggambarkan domain atau bisnes aplikasi yang ingin dibangunkan. Elakkan menggunakan nama yang sama dengan kata kunci (*reserve word*) pangkalan data untuk mengelakkan konflik.

**Langkah 4 : Wujudkan Jadual (*Create Table*)**

a) Jadual diwujudkan setelah pangkalan data siap dibangunkan. Jadual adalah terdiri daripada baris dan lajur yang mengandungi rekod maklumat dalam pangkalan data. Dalam reka bentuk pangkalan data logikal (sila rujuk **Reka Bentuk Pangkalan Data Logikal [F3.3]**) Entiti adalah merujuk kepada jadual (*table*) bagi pangkalan data fizikal. Manakala atribut pula adalah medan (*field*). Jadual berikut adalah pemadanan secara teknologi antara reka bentuk pangkalan data logikal dan fizikal.

**Jadual 5.1 : Pemadanan Istilah antara reka bentuk logikal dan fizikal pangkalan data**

| Istilah dalam reka bentuk logikal | Istilah dalam reka bentuk fizikal |
|---|---|
| Entiti | Jadual (*table*) |
| Atribut | Medan (*column* atau *field*) |
| *Primary UID* | *Primary Key* |
| *Secondary UID* | *Unique Key* |
| *Relationship* | *Foreign Key* |

b) Perkara yang perlu diberi perhatian semasa mencipta jadual:-

i) Untuk mengelakkan data tidak diisi, ciri-ciri *NOT NULL* diberikan kepada medan tersebut

ii) Ciri medan *AUTO_INCREMENT* adalah bagi menambahkan satu nilai seterusnya dalam medan dengan mengambil kira bahawa jenis medan adalah *INT*

iii) Medan yang ditakrif sebagai *PRIMARY KEY* adalah menunjukkan bahawa medan tersebut adalah kunci utama dalam jadual. *PRIMARY KEY* boleh ditakrifkan kepada satu atau lebih medan.

iv) Penggunaan *PRIMARY KEY* dan *FOREIGN KEY* untuk melaksanakan normalisasi (hingga tahap ke3 atau ke-4) untuk mengurangkan pengulangan data dan meningkatkan integriti data.

**Langkah 5 : Wujudkan *VIEW* (*Create VIEW*)**

a) *VIEW* adalah merujuk kepada jadual maya yang dihasilkan melalui arahan *SQL*. Jadual adalah terdiri daripada baris dan lajur yang mengandungi rekod maklumat berdasarkan arahan *SQL* yang telah dilaksanakan.

b) Dalam satu pangkalan data, *view* dan jadual berkongsi ruang jadual. Namun begitu, *view* dan jadual tidak boleh mempunyai nama yang sama.

c) Beberapa fungsi *SQL* boleh dimasukkan ke dalam arahan seperti *WHERE* dan arahan *JOIN* daripada beberapa jadual lain kepada jadual maya yang mana fungsi arahan tersebut dipaparkan sebagai satu jadual.

d) Penggunaan *view* juga boleh meningkatkan keselamatan data dengan menghadkan capaian mengikut *view* yang telah diwujudkan.

**Langkah 6 : Wujudkan *INDEX* (*Create INDEX*)**

a) Kebiasaannya data disimpan tidak mengikut urutan. Data baru yang dimasukkan tidak disimpan mengikut susunan berdasarkan data yang dimasukkan terdahulu. Oleh yang demikian, tempoh untuk menemui data berdasarkan arahan adalah kurang pantas berbanding kemasukan data. Dengan itu, index perlu dilakukan bagi membolehkan data ditemui dengan lebih cepat.

b) Arahan *INDEX* digunakan untuk medan-medan tertentu dalam sesuatu jadual bagi mempercepatkan carian data. Lokasi sesuatu data lebih pantas ditemui berbanding carian satu-persatu baris yang terdapat dalam jadual jika tidak menggunakan *INDEX*. Sebaiknya, gunakan *INDEX* pada lajur (column) yang sering dicari, dihubungkan (*JOIN*), atau ditapis (*WHERE*).

**Langkah 7 : Memuat Masuk (*Load*) Data Ke Dalam Pangkalan Data**

Sekiranya terdapat data daripada pangkalan data lama, data tersebut boleh dimuat masuk ke dalam pangkalan data yang baru dibina. Terdapat dua cara untuk memuat masuk data yang sedia ada ke dalam pangkalan data iaitu:

a) Menggunakan Penyataan *INSERT* untuk memasukkan rekod tunggal ke dalam jadual.

b) Menggunakan Penyataan *LOAD DATA* membolehkan semua data yang terdapat dalam fail teks dimasukkan ke dalam pangkalan data menggunakan satu arahan sahaja.

**Langkah 8 : Wujudkan Pengguna Dan Kawalan Capaian**

a) Pengguna diwujudkan untuk mengakses sesuatu pangkalan data. Dengan itu, pengguna perlu mempunyai capaian *root* atau *superuser* untuk melaksanakan aktiviti-aktiviti pembangunan pangkalan data dan juga mencipta pengguna.

b) Pentadbir pangkalan data (*Database Administrator* – DBA) boleh memberi kebenaran capaian kepada pengguna melalui arahan *GRANT*.

c) Berikut adalah senarai arahan kebenaran (*GRANT*) yang digunakan mengikut kesesuaian capaian.

i) *ALL PRIVILEGES* – membenarkan semua aktiviti
ii) *CREATE* – membenarkan pengguna mencipta pangkalan data dan jadual
iii) *DROP* – membenarkan pengguna menghapus pangkalan data dan jadual
iv) *DELETE* – membenarkan pengguna menghapuskan baris rekod dalam jadual
v) *INSERT* – membenarkan pengguna menambah baris rekod dalam jadual
vi) *SELECT* – membenarkan pengguna membaca rekod dalam pangkalan data
vii) *UPDATE* – membenarkan pengguna mengemaskini rekod dalam jadual

d) Pentadbir pangkalan data (*Database Administrator* – DBA) boleh menarik balik kebenaran pengguna melalui arahan REVOKE.

**Langkah 9 : Dokumenkan Pangkalan Data**

Dokumenkan maklumat pangkalan data fizikal yang dibangunkan ke dalam **D09 Dokumen Pangkalan Data**. Dokumentasi mengikut susunan berikut:

a) Ringkasan maklumat pangkalan data fizikal.

b) Skrip yang mengandungi arahan-arahan *SQL.*

### 5.5.4 LANGKAH-LANGKAH PEMBANGUNAN PANGKALAN DATA NOSQL

**Langkah 1 : Tentukan Pangkalan Data NoSQL Yang Bersesuaian**

Sebelum proses pemasangan perisian dimulakan, penentuan jenis NoSQL yang bersesuaian kepada keperluan projek perlu dikenal pasti sama ada model data jenis *key-value pairs, column, document-based* ataupun model data jenis *graph.* Oleh itu, hanya gunakan pangkalan data berasaskan NoSQL mengikut keperluan fungsian aplikasi yang ingin dibangunkan. Buku

panduan ini hanya menyentuh berkaitan pembangunan pangkalan data NoSQL yang berasaskan dokumen menggunakan perisian MongoDB.

**Langkah 2 : Pemasangan (*install*) Perisian NoSQL**

Pangkalan data fizikal dibangunkan dengan memasang perisian NoSQL yang dipilih. Sekiranya MongoDB yang dipilih, perisian tersebut boleh dimuat turun daripada laman web MongoDB dan rujuk dokumentasi pemasangan langkah demi langkah daripada laman web yang sama.

**Langkah 3 : Cipta Pangkalan Data**

MongoDB membenarkan pangkalan data dicipta menggunakan beberapa kaedah seperti menggunakan arahan *command-line* atau perisian bantuan seperti MongoDB Compass.

Arahan *mongosh* digunakan untuk memulakan sesi interaktif MongoDB Shell yang didatangkan bersekali dengan pemasangan perisian MongoDB.

Arahan *use sistem_tempahan* pula digunakan untuk mencipta (jika belum wujud) atau menukar kepada pangkalan data bernama *sistem_tempahan*.

**Langkah 3 : Tentukan Reka Bentuk Struktur Data**

Pangkalan data NoSQL seperti MongoDB tidak menggunakan hubungan tradisional seperti RDBMS. Oleh itu, data boleh disimpan dalam bentuk koleksi (*collection*) yang mengandungi dokumen (*documents*) dalam format JSON/BSON.

Dua jenis reka bentuk struktur data yang boleh dipilih, iaitu secara:

a) Pemodelan dokumen secara penyematan (embeded) yang sesuai untuk data yang sering dicapai bersama; atau
b) Pemodelan dokumen secara rujukan (referencing) untuk data bersaiz besar dan capaian dikongsi oleh banyak dokumen.

Arahan *db.createCollection("bilik_mesyuarat")* digunakan untuk mencipta dokumen / koleksi bagi pangkalan data.

Manakala, arahan *db.bilik_mesyuarat.insertOne({nama: "Bilik Mesyuarat Utama", catatan: "Kapasiti maksima ialah 1000 orang"})* digunakan untuk mencipta serta memasukkan data ke dalam dokumen / koleksi tersebut.

## Langkah 4 : Menentukan Strategi Pengindeksan

Pengindeksan adalah langkah penting dalam reka bentuk skema NoSQL kerana ia membantu mempercepatkan prestasi pertanyaan (*query*) dengan mengurangkan masa carian dalam pangkalan data. Pemilihan indeks yang sesuai bergantung kepada corak pertanyaan yang digunakan dalam aplikasi. Berikut adalah **faktor utama** dan bagaimana ia mempengaruhi pemilihan struktur data dalam NoSQL:

**a) Indeks Medan Tunggal (*Single Field Index*)**

Indeks ini dibuat pada satu medan dalam dokumen. Misalnya, untuk mengindeks medan *user_id* secara menaik dengan menggunakan arahan *db.users.createIndex({"user_id":1})*. Indeks ini berguna untuk pertanyaan (*query*) berdasarkan satu medan tertentu.

**b) Indeks Majmuk (*Compound Index*)**

Indeks ini melibatkan lebih dari satu medan. Sebagai contoh, untuk mempercepat pencarian berdasarkan medan *city* dan *zip*, arahan *db.addresses.createIndex({"city": 1, "zip": 1})* boleh dilaksanakan. Urutan medan dalam indeks majmuk penting dan harus disesuaikan dengan corak pertanyaan (*query*) yang sering digunakan.

**c) Indeks Teks (*Text Index*):**

Digunakan untuk pencarian teks penuh pada medan jenis *string*. Contohnya, untuk mengindeks medan *name* bagi pencarian teks, arahan *db.products.createIndex({"name":"text"})* boleh digunakan untuk memungkinkan pencarian kata kunci dalam medan jenis *string* dengan efisien.

## Langkah 5 : Optimumkan Pertanyaan (*Query*)

Dalam pemodelan data NoSQL, adalah penting untuk memastikan bahawa pertanyaan *(query)* dijalankan dengan cekap bagi mengelakkan kelewatan dalam pemprosesan data. Berikut adalah beberapa prinsip utama untuk mencapai kecekapan tersebut:

a) Memastikan Pertanyaan (*Query*) Efisyen dengan Mengelakkan Saiz Dokumen yang Besar
Dokumen yang terlalu besar boleh menyebabkan penggunaan sumber yang tinggi, meningkatkan masa pemprosesan, dan memperlahankan pertanyaan *(query)* kerana keseluruhan dokumen perlu dimuatkan ke dalam memori. Oleh itu :

i) Simpan hanya data yang diperlukan dalam setiap dokumen.

ii) Gunakan teknik **normalisasi separa**, iaitu menyimpan data yang kerap diakses dalam dokumen utama dan menempatkan data yang kurang digunakan dalam koleksi lain (boleh diakses melalui rujukan).

iii) Pemecahan data kepada bahagian kecil untuk mengelakkan pertumbuhan dokumen yang tidak terkawal.

b) Gunakan *Projection* untuk Mengembalikan (*Returning*) Medan yang Diperlukan Sahaja

Jika anda hanya memerlukan beberapa medan dalam pertanyaan (*query*), mengambil keseluruhan dokumen adalah langkh yang tidak efektif dan boleh menyebabkan penggunaan memori serta data rangkaian yang tinggi. Oleh itu, *Projection* digunakan dalam pangkalan data dokumen suntuk hanya mengambil medan yang diperlukan sahaja semasa melakukan pertanyaan (*query*).

c) Elakkan Struktur Data yang Mempunyai Banyak Lapisan (*Deeply Nested Structure*s)

Mengelakkan struktur berlapis (*deeply nested structures*) dalam pangkalan data NoSQL adalah penting untuk memudahkan pengemaskinian data dan mengoptimumkan prestasi pertanyaan (*query*). Struktur data yang terlalu bersarang (contoh: objek dalam objek atau array dalam array) menyukarkan pengemaskinian kerana:

i) **Penulisan Semula Dokumen**

Dalam pangkalan data NoSQL seperti MongoDB, dokumen disimpan dalam format seperti JSON atau BSON. Jika struktur data yang terlalu berlapis, mengemaskini satu elemen di dalamnya berkemungkinan memerlukan penulisan semula keseluruhan dokumen. Ini kerana NoSQL seringkali tidak menyokong pengemaskinian separa (*partial update*) pada elemen yang terlalu dalam.

ii) **Beban Pemprosesan**

Pengemaskinian pada struktur yang berlapis memerlukan lebih banyak sumber pemprosesan dan memori kerana sistem perlu mengimbas dan memproses keseluruhan dokumen. Ini boleh mengurangkan prestasi sistem, terutamanya jika dokumen tersebut besar atau kerap dikemaskini.

iii) **Kesukaran Pertanyaan (*Query*)**

Pertanyaan *(query)* pada struktur data yang berlapis cenderung menjadi lebih kompleks dan perlahan. Sebagai contoh, proses mencari atau mengemas kini data dalam array bersarang memerlukan penggunaan

operator khusus (seperti `$elemMatch` dalam MongoDB), yang mungkin kurang efisien berbanding pertanyaan pada struktur data yang lebih rata.

**Cara Mengelakkan *Deeply Nested Structures*:**

**i) Gunakan Struktur Rata (Flat)**

Seboleh-bolehnya, reka struktur data yang lebih rata dengan mengurangkan tahap lapisan. Sebagai contoh, daripada menyimpan semua maklumat dalam satu dokumen, anda boleh memecahkan data menjadi beberapa koleksi atau dokumen yang berasingan dan merujuk antara satu sama lain menggunakan identifier (seperti id).

**ii) Denormalisasi Terkawal**

Dalam NoSQL, denormalisasi (menyimpan data yang berulang) kadangkala diperlukan untuk meningkatkan prestasi pertanyaan (*query*). Namun, pastikan denormalisasi dilakukan secara terkawal dan tidak mencipta struktur yang terlalu kompleks.

**iii) Pecahkan Data Besar**

Jika dokumen terlalu besar atau kompleks, pertimbangkan untuk memecahkannya menjadi dokumen yang lebih kecil. Sebagai contoh, daripada menyimpan semua pesanan pelanggan dalam satu dokumen, simpan setiap pesanan sebagai dokumen berasingan dan hubungkan dengan *customer_id*.

## Langkah 6 : Rancang Skalabiliti dan Pengurusan Data Besar (*Sharding* dan *Partitioning*)

Apabila data bertambah besar dalam pangkalan data NoSQL, anda perlu memastikan **prestasi pangkalan data adalah kekal stabil** dan **data dapat diakses dengan pantas**. Perkara ini boleh dicapai melalui kaedah ***sharding*** dan ***partitioning***.

a) *Sharding*
*Sharding* adalah teknik membahagikan data ke dalam beberapa bahagian yang lebih kecil (disebut "*shard*") dan menyebarkannya ke beberapa pelayan atau *node* dalam sistem. Setiap *shard* mengandungi subset data yang unik, dan setiap *shard* disimpan pada *node* yang berbeza. Ini membolehkan sistem mengagihkan beban dan meningkatkan prestasi.

Kelebihan *Sharding*:

i) **Skalabiliti Mendatar (*horizontal scaling*)**: *Sharding* membolehkan sistem berkembang secara mendatar dengan menambah lebih banyak *node*.
ii) **Peningkatan Prestasi**: Dengan membahagikan data, beban kerja diagihkan ke beberapa *node*, mengurangkan beban pada setiap *node* individu.
iii) **Ketersediaan Tinggi**: Jika satu *node* gagal, hanya *shard* pada *node* tersebut yang terjejas, bukan keseluruhan sistem.

Cabaran *Sharding*:

i) **Kompleksiti Pengurusan**: Menguruskan *shard* dan memastikan data diagihkan dengan betul boleh menjadi kompleks.
ii) **Ketidakseimbangan Beban**: Jika data tidak diagihkan secara betul, beberapa *shard* mungkin menjadi "*hotspot*" yang menyebabkan prestasi menurun.

b) *Partitioning*

*Partitioning* adalah teknik membahagikan data ke dalam *partition* (bahagian) berdasarkan kriteria tertentu, seperti **nilai kunci** atau **julat nilai**. *Partitioning* boleh dilakukan secara menegak (*vertical partitioning*) atau mendatar (*horizontal partitioning*).
Jenis Partitioning:

***Horizontal Partitioning (Sharding)**:* Membahagikan data berdasarkan baris. Setiap *partition* mengandungi subset baris dari jadual atau koleksi.

***Vertical Partitioning***: Membahagikan data berdasarkan lajur. Setiap partition mengandungi subset lajur dari jadual atau koleksi.

Kelebihan Partitioning:

i) **Pengoptimuman Prestasi**: Dengan membahagikan data, *query* boleh dijalankan pada subset data yang lebih kecil, meningkatkan prestasi.
ii) **Pengurusan Data yang Lebih Baik**: *Partitioning* membolehkan pengurusan data yang lebih teratur dan efisien.

Cabaran Partitioning:

i) **Kompleksiti Reka Bentuk**: Memilih kriteria *partitioning* yang betul adalah penting untuk memastikan data diagihkan dengan baik.
ii) **Ketidakseimbangan Data**: Jika kriteria *partitioning* tidak dipilih dengan baik, beberapa *partition* mungkin menjadi terlalu besar atau terlalu kecil.

**Langkah 7 : Dokumenkan Pangkalan Data**

Dokumenkan maklumat pangkalan data fizikal yang dibangunkan ke dalam **D08 Dokumen Pangkalan Data** yang mengandungi maklumat berikut:

a) Maklumat pangkalan data fizikal.
   i) Profil pangkalan data
   ii) Peruntukan ruang storan
   iii) Bilangan table, view
   iv) Stored procedure
   v) Maklumat jadual dan medan yang mengandungi INDEKS
   vi) Senarai pengguna yang mempunyai kawalan akses ke atas pangkalan data / jadual.

b) Skrip yang mengandungi arahan-arahan SQL khusus mengikut perisian NoSQL yang digunakan.

### 5.5.5 AMALAN TERBAIK PEMBANGUNAN PANGKALAN DATA

**a) Gunakan Teknik Pemantauan dan Logging**

Pantau penggunaan CPU, memori, prestasi carian, dan *latency* menggunakan perisian sokongan seperti *Prometheus, Grafana,* atau *New Relic*.

**b) Kawal Akses Keselamatan**

i) Gunakan *Role-Based Access Control (RBAC)* untuk mengehadkan capai pengguna kepada pangkalan data.

ii) Amalkan pengasingan / pemisahan tugas kepada pengguna yang diberikan akses kepada pangkalan data. Pastikan pengguna hanya diberikan akses yang minimum yang diperlukan untuk melaksanakan tugas.

iii) Pastikan penyulitan data dilaksanakan pada data sensitif yang terdapat dalam pangkalan data.

**c) Menapis Input Pengguna**

Gunakan prepared statements / parameterized queries, serta elakkan menggunakan input pengguna secara langsung ke dalam *query*.

**d) Kemaskini Versi dan Patching**

Sentiasa pastikan DBMS adalah versi yang terkini serta sentiasa melaksankan patch keselamatan terkini.

**e) Gunakan *Connection Pooling***

Hadkan bilangan sambungan serentak untuk mengelakkan beban pada pangkalan data.

**f) Lakukan Ujian Prestasi**

Laksanakan ujian bebanan menggunakan perisian seperti *JMeter* atau *Gatling* sebelum pelancaran.

**g) Jalankan Salinan Sandar dan Ujian Pemulihan**

Jika menggunakan MongoDB, boleh menggunakan fungsi mongodump untuk melaksanakan salinan sandar serta fungsi mongostore untuk memulihkan kembali backup.

**h) Pastikan Dokumentasi yang Jelas**

Dokumentasikan skema, indeks, dan pertanyaan penting untuk memudahkan penyelenggaraan.

## 5.6. PENGATURCARAAN APLIKASI

### 5.6.1 KETERANGAN

Pengaturcaraan aplikasi merujuk kepada proses menulis, menguji, dan menyelenggara kod pengaturcaraan dalam pembangunan sistem bagi memenuhi keperluan dan reka bentuk sistem yang telah disediakan pada fasa-fasa sebelumnya. Proses ini melibatkan penggunaan bahasa dan rangka kerja pengaturcaraan tertentu bagi menghasilkan fungsi, antaramuka dan logik supaya sistem yang dibangunkan dapat beroperasi dengan cekap, selamat, dan mudah untuk dikendalikan. Amalan-amalan terbaik dalam pengaturcaraan menjamin kod-kod pengaturcaraan yang dihasilkan adalah berkualiti tinggi, mudah untuk diselenggara, dan meminimakan berlakunya sebarang ralat.

### 5.6.2 OBJEKTIF

a) Memberi penerangan kepada tahap asas-asas pengaturcaraan sistem.

b) Menyenaraikan amalan-amalan terbaik di dalam pengaturcaraan sistem.

c) Melihat kepada amalan pengaturcaraan yang selamat.

### 5.6.3 TAHAP-TAHAP PENGATURCARAAN

Dalam pembangunan sistem, bahasa pengaturcaraan boleh diklasifikasikan kepada lima (5) tahap utama berdasarkan kepada kaedah pemprosesan kod yang ditulis supaya dapat dilaksanakan oleh komputer. Setiap tahap ini menggambarkan perbezaan peringkat abstraksi antara kod yang ditulis dengan arahan yang boleh difahami dan dilaksanakan oleh satu-satu perkakasan komputer melalui pemprosesan tertentu.

**a) *Machine Language***

*Machine language* adalah bahasa pengaturcaraan yang paling asas berbentuk kod binari (0 dan 1) yang dapat terus dilaksanakan oleh satu-satu komputer tanpa sebarang lapisan penterjemahan tambahan. Kod pada peringkat ini biasanya sangat sukar untuk ditulis dan dibaca secara manual kerana ia mewakili arahan rendah dan asas yang hanya boleh difahami oleh komputer.

**b) *Assembly Language***

*Assembly language* merupakan bahasa pengaturcaraan pada tahap rendah yang menggunakan notasi simbolik atau mnemonik menggantikan kod binari. Pengaturcaraan pada tahap ini memudahkan penulisan kod arahan menggunakan simbol yang setara dengan arahan mesin, serta memberikan kawalan yang lebih tinggi dalam aktiviti pengaturcaraan. Walaupun tahap bahasa ini adalah lebih

mudah berbanding *machine language*, penggunaan *assembly language* masih memerlukan asas pengetahuan yang mendalam berkenaan dengan struktur-struktur pengkomputeran. Contoh *assembly language* yang lazim digunakan adalah seperti x86 Assembly, ARM Assembly, dan MIPS Assembly.

**c) *High Level Language***

Bahasa pengaturcaraan pada peringkat ini menawarkan tahap abstraksi yang lebih tinggi berbanding *assembly language* dengan mempermudahkan pengurusan aspek teknikal seperti memori dan arahan mesin. Penulisan kod pada tahap ini menggunakan sintaks yang lebih praktikal untuk difahami dan menghampiri bahasa manusia (natural language). Oleh yang demikian, proses pembangunan dapat dilaksanakan dengan lebih cepat dan memudahkan penyelenggaraan sistem aplikasi yang dibangunkan. Kod yang ditulis dalam *high level language* pada kebiasaannya memerlukan proses kompilasi atau interpretasi sebelum ia boleh diproses oleh satu-satu komputer. Contoh *high level language* adalah seperti Java, C++, dan Python.

**d) *Scripting Language***

*Scripting language* merupakan tahap pengaturcaraan yang dijalankan secara terus melalui proses interpretasi tanpa memerlukan kompilasi terlebih dahulu. Bahasa ini sesuai digunakan untuk tugasan-tugasan seperti automasi, prototaip pantas, serta sebagai penghubung antara komponen aplikasi yang berbeza dalam satu-satu sistem. *Scripting language* menggunakan sintaks arahan yang ringkas dan dinamik supaya kod dapat dilaksanakan dengan cepat dan mudah disesuaikan mengikut keperluan. Contoh *scripting language* yang lazim digunakan adalah PHP, JavaScript, dan Ruby.

**e) *Domain-Specific Language***

*Domain-Specific Language* adalah bahasa pengaturcaraan yang direka khusus untuk kegunaan di dalam domain atau bidang tertentu. Bahasa pada tahap ini mempunyai sintaks dan ciri-ciri yang telah dioptimum untuk menyelesaikan masalah atau tugasan khusus dalam konteks domain berkenaan. *Domain-Specific Language* dapat memudahkan pengaturcara menghasilkan kod yang lebih ringkas dan tepat kerana ia direka khas untuk menangani masalah yang terkandung dalam satu-satu domain tertentu, berbanding dengan bahasa pengaturcaraan umum yang lebih bersifat generik. Contoh bahasa *Domain-Specific Language* adalah SQL untuk pengurusan pangkalan data, HTML dan CSS untuk pembangunan web, dan RegEx untuk pengurusan corak teks dan carian.

### 5.6.4 JENIS-JENIS PENGATURCARAAN

Di dalam pembangunan sistem, pelbagai jenis pengaturcaraan kerap dipraktikkan bergantung kepada pendekatan dan kaedah yang dipilih untuk menyelesaikan satu-satu masalah tertentu. Jenis-jenis pengaturcaraan ini mencerminkan gaya dan paradigma yang berbeza dalam menulis kod, mengurus data, dan mengawal aliran kawalan, iaitu susunan langkah-langkah yang dijalankan berdasarkan kepada keputusan (decisions) dan peraturan (conditions) di dalam kod. Memahami jenis pengaturcaraan yang berbeza ini dapat membantu pengaturcara memilih pendekatan yang paling ideal untuk satu-satu projek, di mana keputusan yang dibuat secara langsung mampu meningkatkan keberkesanan serta kualiti hasil sistem yang dibangunkan.

**a) Pengaturcaraan Tidak Berstruktur (*Unstructred Programming*)**

Kod ditulis tanpa berstruktur, semua arahan ditulis dalam fungsi main().

**b) Pengaturcaraan Berstruktur (*Structured Programming*)**

Pengaturcaraan berstruktur merupakan pengaturcaraan bertatacara. Kod yang besar dipecahkan kepada kaedah-kaedah pendek (juga dikenali sebagai fungsi atau tatacara) yang lebih kecil agar mudah difahami.

Pengaturcaraan berstruktur biasanya dikaitkan dengan reka bentuk yang menggunakan pendekatan atas-bawah. Dengan pendekatan ini, pengaturcara memetakan struktur yang besar dalam aturcara kepada bentuk operasi kecil, seterusnya melaksanakannya dan menguji operasi-operasi kecil tersebut, dan akhirnya menggabungkan kepada keseluruhan aturcara.

**c) Pengaturcaraan Berorientasikan Objek (O*bject-Oriented Programming*)**

Pengaturcaraan berorientasikan objek merupakan aturcara komputer yang terdiri daripada sekumpulan unit-unit atau objek. Setiap objek berupaya untuk menerima dan menghantar mesej (pesanan) kepada objek lain. Dengan cara ini, mesej dapat ditangani oleh sebahagian daripada kod. Konsep Asas pengaturcaraan berorientasikan objek adalah seperti berikut:

i) Kelas (Class) mentakrifkan ciri-ciri abstrak bagi sesuatu benda. Ini termasuklah sifat-sifat yang ada padanya dan peranannya.
ii) Objek (Object) merupakan instance bagi suatu kelas.
iii) Kaedah (Method) merujuk kepada fungsi atau prosedur yang dimiliki oleh sebuah objek.
iv) Pewarisan (Inheritance) merupakan konsep di mana sebuah kelas boleh memiliki “subkelas” yang mengkhususkan ciri dan tingkah laku kelas induknya. Semua subkelas tersebut akan mewarisi segala sifat dan kaedah yang terdapat pada kelas induk tersebut.

v) Pengkapsulan (Encapsulation) mengasingkan pelaksanaan (implementasi) dalaman objek daripada antaramuka bertujuan untuk melindungi data dan mengawal akses.
vi) Pengabstrakan (Abstraction) mempermudahkan struktur data dan operasi kepada jenis data yang ringkas serta hanya mengandungi sifat-sifat penting untuk tujuan tertentu.
vii) Polimorfisme (Polymorphism) membolehkan fungsi atau kaedah yang sama digunakan untuk operasi berbeza berdasarkan jenis objek atau data.

### 5.6.5 ASAS-ASAS PENGATURCARAAN

Asas-asas pengaturcaraan merujuk kepada elemen-elemen penting yang membentuk kepada asas penulisan kod dalam pembangunan sistem. Pemahaman terhadap konsep-konsep ini amat penting bagi membolehkan pengaturcara membina logik yang tepat dan tersusun serta memastikan sistem yang dibangunkan berfungsi dengan cekap dan mudah diselenggara.

**a) Nilai Tetap (Constant)**

Nilai tetap ialah ruang memori yang diisytiharkan (declare) dan digunakan untuk menyimpan data yang tidak boleh diubah dalam fail pengaturcaraan. Contoh:

```
<?php
    define("MAKSIMUM_BILANGAN_HARI_TEMPATAN", 3);   /* Nilai tetap bagi had
    Maksimum Bilangan Hari Tempahan*/
?>
```

**b) Pemboleh Ubah (Variable)**

Pemboleh ubah ialah ruang memori yang diisytihar (declare) dan digunakan untuk menyimpan serta mengendalikan data dalam fail pengaturcaraan, di mana nilai yang disimpan boleh diubah semasa aplikasi sedang beroperasi. Contoh:

```
<?php
    $namaPengguna = "Ahmad";        // Pemboleh ubah bagi Nama Pengguna
    $bilanganOrang = 10;            // Pemboleh ubah bagi Bilangan Orang
    $tarikhTempahan = "2025-07-18"; // Pemboleh ubah bagi Tarikh Tempahan
    $masaTempahan = "14:00";        // Pemboleh ubah bagi Masa Tempahan
?>
```

**c) Parameter**

Parameter merupakan *placeholder* atau slot khas di dalam sesuatu fungsi atau kelas yang digunakan untuk menerima data input. Berbeza dengan pemboleh ubah biasa yang menyimpan data dalam keseluruhan program, parameter hanya

wujud dalam konteks fungsi dan berperanan sebagai pengantara untuk memindahkan data ke dalam fungsi tersebut. Contoh:

```
<?php
  function tempahBilik($namaPengguna)
   {
     // Proses tempahan bilik bagi pengguna tertentu
     echo "Tempahan bilik diterima untuk " . $namaPengguna;
   }
    tempahBilik("Ahmad");
    // Memanggil fungsi dengan nama pengguna (”Ahmad”) sebagai parameter
?>
```

### d) Jenis Data (Data Type)

Jenis data adalah merujuk kepada kategori nilai yang boleh disimpan dalam satu-satu pemboleh ubah seperti *integer*, *float*, *string* atau *boolean*. Memahami jenis data penting untuk memastikan operasi dan pengiraan dalam pengaturcaraan dilaksanakan dengan tepat. Contoh:

```
<?php
      $bilanganOrang = 10;                  // Jenis data adalah Integer
      $hargaSejam = 25.50;                  // Jenis data adalah Float
      $tarikhTempahan = "2025-07-18";  // Jenis data adalah String
?>
```

### e) Komen

Komen ialah teks penerangan dalam kod pengaturcaraan yang tidak akan dilaksanakan oleh komputer. Ia digunakan untuk menerangkan fungsi kod, memberi nota, atau memudahkan pemahaman seseorang pengaturcara kepada kod yang telah dibina sebelum ini.

```
PHP
// Ini adalah format komen satu baris dalam PHP
/* Ini adalah format komen berbilang baris dalam PHP */

Java
// Ini adalah format komen satu baris dalam Java
/* Ini adalah format komen berbilang baris dalam Java */

JavaScript
// Ini adalah format komen satu baris dalam JavaScript
/* Ini adalah format komen berbilang baris dalam JavaScript */

Python
 # Ini adalah format komen dalam Python

HTML
<!-- Ini adalah format komen dalam HTML -->

CSS
/* Ini adalah format komen dalam CSS */
```

### f) Arimetik

Aritmetik merujuk kepada operasi matematik yang digunakan dalam pengaturcaraan untuk mengira nilai, seperti tambah (‘+’), tolak (‘-’), darab (‘*’), dan bahagi (‘/’). Contoh:

```
<?php
        $masaMula = strtotime("09:30");                          // Masa mula tempahan
        $masaTamat = strtotime("12:15");                         // Masa tamat tempahan
        $jumlahMasa = ($masaTamat - $masaMula) / 3600;   // Penggunaan arimetik
?>
```

### g) Operator Logikal

Operator logikal digunakan untuk menggabung atau membandingkan nilai *boolean* dalam pengaturcaraan. Ia membantu semasa membuat sesuatu keputusan berdasarkan kepada peraturan yang telah ditetapkan. Operator logik yang lazim digunakan adalah seperti AND (&&), OR (||), dan NOT (!). Contoh:

```php
<?php
$bilanganOrang = 8;
$kapasitiBilik = 10;
$tempohTempahanValid = true;

// Penggunaan operator logikal
if ($bilanganOrang <= $kapasitiBilik && $tempohTempahanValid)
{
   echo "Tempahan diterima.";
}
else
{
   echo "Tempahan tidak sah.";
}
?>
```

### i) Pernyataan Bersyarat (Conditionals)

Pernyataan bersyarat membolehkan pengaturcara melaksanakan kod tertentu berdasarkan sama ada sesuatu syarat dipenuhi atau tidak. Struktur bersyarat yang biasa digunakan termasuk *if, elseif*, dan *else* yang membolehkan aliran program berubah mengikut nilai logik yang ditetapkan. Contoh:

```php
<?php
$bilanganOrang = 12;
$kapasitiBilik = 10;

// Penggunaan pernyataaan bersyarat if, elseif, dan else
if ($bilanganOrang > $kapasitiBilik) {
   echo "Tempahan melebihi kapasiti bilik.";
} elseif ($bilanganOrang == $kapasitiBilik) {
   echo "Tempahan mencapai kapasiti maksimum bilik.";
} else {
   echo "Tempahan diterima.";
}
?>
```

**j) Pernyataan Pilihan**

Pernyataan pilihan membolehkan pengaturcara memilih salah satu antara beberapa blok kod untuk dilaksanakan mengikut kepada nilai sesuatu pemboleh ubah. Pernyataan pilihan biasanya digunakan sebagai alternatif kepada pernyataan bersyarat *if-else* apabila sesuatu proses perlu membuat keputusan berdasarkan kepada nilai atau pilihan yang banyak. Dalam kaedah pernyataan pilihan *switch*, setiap kebarangkalian nilai yang dibandingkan adalah dikenali sebagai *case*. Contoh:

```
<?php
$kodStatus = 2;

// Penggunaan pernyataan pilihan switch/case
switch ($kodStatus) {
   case 1:
      echo "Tempahan diterima.";
      break;
   case 2:
      echo "Tempahan dalam proses.";
      break;
   case 3:
      echo "Tempahan ditolak.";
      break;
   default:
      echo "Kod status tidak dikenali.";
      break;
}
?>
```

**k) Iterasi**

Iterasi merujuk kepada proses untuk mengulangi satu set arahan atau blok kod sehinggalah syarat-sayarat tertentu dipenuhi. Jenis iterasi yang biasa digunakan adalah seperti *for*, *while*, dan *do-while*. Kaedah ini membolehkan pengaturcara melaksanakan tugasan yang sama tanpa perlu menulis kod berulang kali. Contoh-contoh:

```
// Kaedah for loop
<?php
$senaraiPengguna = ["Ahmad", "Siti", "Ali", "Zara"];
for ($i = 0; $i < count($senaraiPengguna); $i++)
{
   echo $senaraiPengguna[$i] . "<br>";
}
?>
```

```
// Kaedah while loop
<?php
$bilangan = 0;
while ($bilangan < 4)
{
   echo "Tempahan ke-" . ($bilangan + 1) . "<br>";
   $bilangan++;
}
?>
```

```
// Kaedah do-while
<?php
$bilangan = 0;
do
{
   echo "Tempahan ke-" . ($bilangan + 1) . "<br>";
   $bilangan++;
} while ($bilangan < 4);
?>
```

## l) Fungsi

Fungsi merupakan blok kod yang dibangun untuk melaksanakan satu-satu operasi atau tugasan tertentu bertujuan untuk diguna pakai semula dalam kod pengaturcaraan yang lain. Penggunaan kaedah fungsi ini menjadikan kod pengaturcaraan lebih teratur, mudah dibaca dan meningkatkan kebolehulangan (reusability). Satu-satu fungsi yang disediakan akan menerima input melalui parameter dan menghasilkan output daripada pengiraan atau pemprosesan data yang dilakukan di dalamnya. Contoh:

```php
<?php

//Penggunaan fungsi dalam kod pengaturcaraan
function kiraTempohTempahan($masaMula, $masaTamat)
{
   $tempoh = strtotime($masaTamat) - strtotime($masaMula);
   $jam = $tempoh / 3600; // Tukar ke jam
   return $jam;
}

// Memanggil fungsi
$jumlahJam = kiraTempohTempahan("09:00", "12:30");
echo "Jumlah tempoh tempahan adalah " . $jumlahJam . " jam.";
?>
```

**m) Kelas**

Kelas merupakan struktur yang menggabungkan maklumat (atribut) dan kaedah (method) dalam satu unit yang mewakili satu komponen di dalam pengaturcaraan berorientasikan objek. Data disimpan sebagai atribut yang kekal di dalam kelas sehingga nilainya diubah melalui arahan atau logik lain. Kaedah pula merupakan fungsi yang didefinisikan dalam kelas dan digunakan untuk melaksanakan sesuatu operasi tertentu. Perbezaan di antara kelas dan fungsi ialah ia menyimpan data secara kekal dalam bentuk atribut bersama fungsi atau kaedah yang berkaitan, manakala fungsi pula hanya melaksanakan satu tugasan tertentu dengan hanya menyimpan data secara sementara. Contoh:

```php
<?php
// Penggunaan kelas dalam kod pengaturcaraan
   class BilikMesyuarat {
   public $nama;
   public $kapasiti;

   public function __construct($nama, $kapasiti) {
      $this->nama = $nama;
      $this->kapasiti = $kapasiti;
   }

   public function info() {
      return "Bilik " . $this->nama . " boleh memuatkan " . $this->kapasiti . " orang.";
   }
}
// Membina objek daripada kelas
$ruangA = new BilikMesyuarat("Bilik Mesyuarat Melur", 40);
         echo $ruangA->info();
?>
```

## n) Struktur Data (Data Structure)

Struktur data ialah kaedah untuk menyimpan dan menyusun maklumat atau data di dalam kod pengaturcaraan supaya ia dapat diakses dan digunakan dengan mudah dan berkesan. Antara kaedah-kaedah struktur data adalah seperti berikut:

### *i) Array / List*

*Array* atau *list* merupakan susunan elemen yang diindeks secara berturutan. Kaedah struktur data ini adalah sesuai untuk menyimpan koleksi data yang mempunyai turutan yang tetap. Contoh:

```
<?php
$bilanganHari = [1, 2, 3, 4, 5];                    // Kaedah array atau list
$senaraiNama = ["Ahmad", "Siti", "Ali"];    // Kaedah array atau list
?>
```

### ii) Kamus (Dictionary)

Kamus merupakan struktur data yang menyimpan pasangan kekunci dengan nilai tertentu. Kaedah ini mampu untuk meningkatkan prestasi sistem oleh kerana ia membolehkan pencarian data dilakukan secara terus tanpa perlu menyemak setiap satu elemen dalam struktur data. Contoh:

```
<?php
// Penggunaan kaedah kamus
$kamusBilikMesyuarat = [
   "BM001" => "Bilik Mesyuarat Melur",
   "BM002" => "Bilik Mesyuarat Cempaka",
   "BM003" => "Bilik Mesyuarat Kenanga",
   "BM004" => "Bilik Mesyuarat Melati"
   "BM005" => "Bilik Mesyuarat Mawar"
   "BM006" => "Bilik Mesyuarat Tulip"
   "BM007" => "Bilik Mesyuarat Sakura"
];
?>
```

### 5.6.6 RANGKA KERJA PENGATURCARAAN (PROGRAMMING FRAMEWORK)

Rangka kerja pengaturcaraan merupakan satu set *tools*, *libraries* dan peraturan yang menyediakan asas-asas bagi melaksanakan aktiviti pengaturcaraan secara teratur dan bersistematik. Rangka kerja ini menyediakan kod sedia ada (pre-built code) dan struktur pengekodan yang teratur bertujuan untuk memudahkan proses pengaturcaraan, penyelenggaraan dan penambahbaikan aplikasi. Jenis rangka kerja pengaturcaraan merangkumi *front end*, *back end*, dan *full stack* yang digunakan dalam pembangunan sistem.

**a) Rangka Kerja *Front-End***

Rangka kerja *front-end* diguna pakai untuk membangunkan antaramuka pengguna sesebuah sistem aplikasi dengan lebih cekap dan teratur. Rangka kerja ini memudahkan pembangunan komponen-komponen visual, interaksi dan maklum balas terhadap input pengguna secara dinamik. Antara contoh rangka kerja *front* end yang sering digunakan adalah seperti React, Angular, dan Vue.js.

**b) Rangka Kerja *Back-End***

Rangka kerja back-end digunakan untuk membangunkan logik server, pengurusan pangkalan data serta penyediaan API bagi menyokong fungsi-fungsi sistem aplikasi. Rangka kerja ini memudahkan pengurusan data, keselamatan dan komunikasi di antara pelayan dengan komponen lain dengan lebih efisien. Antara contoh rangka kerja *back-end* yang sering digunakan ialah seperti Django (Python), Spring (Java), Express.js (JavaScript), dan Laravel (PHP).

**c) Rangka Kerja *Full Stack***

Rangka kerja *full stack* dipakai untuk membangunkan sistem aplikasi yang merangkumi kedua-dua aspek *front-end* dan *back-end* secara menyeluruh. Rangka kerja ini membolehkan pengaturcara membina antaramuka pengguna bersekali dengan logik server dan pengurusan pangkalan data dalam satu platform yang sama. Antara contoh rangka kerja full stack yang popular adalah seperti Meteor.js, Next.js, dan Laravel yang juga menyokong pembangunan secara *full stack*.

Penggunaan rangka kerja dalam pembangunan perisian menawarkan pelbagai manfaat yang membantu menjadikan proses kerja lebih teratur dan meningkatkan kualiti hasil pembangunan. Berikut merupakan kelebihan penggunaan rangka kerja pengaturcaraan, iaitu:

a) Mempercepatkan proses pembangunan dan meningkatkan keselamatan dengan komponen dan fungsi yang sedia ada (built in).

b) Menyediakan standard dan struktur yang konsisten dalam pengekodan.

c) Memudahkan penyelenggaraan dan pengemaskinian sistem.

d) Membantu dalam pengurusan dan pengujian kod yang lebih teratur.

e) Menyokong pembangunan aplikasi yang bersifat kebolehskalaan (scalable) dan fleksibel.

f) Memperkasa kerjasama di antara ahli pasukan dengan penggunaan konvensyen, gaya dan cara pengekodan yang seragam.

### 5.6.7 AMALAN TERBAIK PENGATURCARAAN

Amalan terbaik pengaturcaraan merujuk kepada standard, panduan, prinsip, dan teknik yang membantu pengaturcara menulis kod yang bersih (clean), efisien, mudah difahami, dan mudah diselenggara. Ia bertujuan untuk memastikan perisian yang dibangunkan adalah berkualiti tinggi, bebas daripada ralat kritikal, dan mampu bertahan lama walaupun melalui proses pengemaskinian perisian (software updates).

**a) Kebolehbacaan Kod Pengaturcaraan**

Kebolehbacaan kod pengaturcaraan adalah aspek penting dalam pembangunan sistem bagi memastikan kod yang ditulis mudah untuk difahami serta tersusun dengan kemas. Kod-kod pengaturcaraan yang jelas memudahkan proses pengurusan, penyelenggaraan, serta melancarkan proses pengesanan, pengecaman, dan pembaikan ralat. Antara amalan terbaik untuk meningkatkan kebolehbacaan kod adalah seperti berikut:

i) Inden yang konsisten bagi memastikan struktur kod mudah dibaca dan difahami.

ii) Penetapan nama pemboleh ubah yang bermakna supaya jelas untuk dikenal pasti oleh pengaturcara lain.

iii) Penulisan komen yang ringkas dan padat pada setiap fungsi dan bahagian kod yang memerlukan penerangan lanjut.

iv) Menulis baris kod yang tidak terlalu panjang dengan memecahkan kod ke baris seterusnya supaya meningkatkan kebolehbacaan.

**b) Pematuhan Piawaian dan Konvensyen**

Pematuhan kepada piawaian dan konvensyen membantu memastikan kod yang dihasilkan adalah selaras, konsisten, dan mudah difahami oleh semua ahli pasukan. Dengan mengikuti piawaian dan konvensyen yang telah ditetapkan, kualiti kod dapat dipertingkatkan serta memudahkan proses penyelenggaraan.

i) Konvensyen penamaan yang konsisten seperti penggunaan gaya *lowercase*, *camelCase*, dan *PascalCase* penting untuk memastikan nama pemboleh ubah dan fungsi mudah dikenali dan difahami. Gaya lowercase biasanya digunakan untuk nama fail atau URL, camelCase pula untuk

pemboleh ubah dan fungsi, manakala PascalCase digunakan bagi menamakan kelas. Penggunaan konvensyen yang selaras membantu mengekalkan standard kod dan mengurangkan sebarang kekeliruan di kalangan pengaturcara.

ii) Pematuhan kepada piawaian pengaturcaraan khusus seperti *PHP Standard Recommendation* (PSR) dan *Python Enhancement Proposal* (PEP) adalah mustahak bagi memastikan kualiti kod yang dihasilkan menepati kepada standard industri, serta memudahkan integrasi dan kolaborasi di antara pasukan pembangun yang lain.

**c) Pelaksanaan Kawalan Versi**

i) Kawalan versi ialah proses mengesan dan mengurus perubahan kepada kod sumber menggunakan perisian kawalan versi. Contohnya seperti Git, Tortoise dan lain-lain.

ii) Kawalan versi akan membolehkan pengaturcara untuk bekerja secara kolaboratif dan memastikan setiap perubahan dicatat dengan mesej yang jelas.

iii) Penggunaan kawalan versi membolehkan aktiviti-aktiviti pengaturcaraan bagi fungsi-fungsi sistem dilakukan secara berasingan dan kemudiannya diuji terlebih dahulu sebelum ia digabungkan untuk proses pembangunan seterusnya.

iv) Kawalan versi membolehkan kod sumber dapat dikembalikan (rollback) kepada versi terdahulu sekiranya terdapat sebarang masalah yang timbul setelah kod tersebut diterbitkan.

**d) Mempraktikkan Prinsip DRY**

i) DRY bermaksud *Don't Repeat Yourself*, iaitu prinsip yang menekankan kepada penggunaan semula kod-kod pengaturcaraan melalui kaedah modularisasi bagi mengelakkan sebarang penulisan kod secara berulang.

ii) Pembahagian kod pengaturcaraan kepada fungsi atau kelas yang boleh digunakan semula dalam fail-fail pengaturcaraan lain bertujuan untuk mengurangkan kerumitan dalam pengurusan kod apabila berlaku perubahan pada logik, struktur, atau sintaks.

**e) Mengelakkan *Over-Engineering***

i) *Over-engineering* merujuk kepada amalan pembangunan sistem menggunakan penyelesaian (solution) yang terlalu kompleks berbanding dengan keperluan yang telah diperolehi.

ii) Amalan *over-engineering* berpotensi untuk mempengaruhi kesejahteraan satu-satu projek pembangunan sistem, khususnya kepada pengurusan masa, kos dan sumber manusia, serta boleh menambahkan kerumitan kod-kod pengaturcaraan yang perlu dihasilkan.

### 5.6.8 AMALAN PENGATURCARAAN SELAMAT

Amalan pengaturcaraan selamat merujuk kepada pendekatan dan teknik pengaturcaraan yang bertujuan untuk melindungi aplikasi daripada kelemahan keselamatan (security vulnerability). Amalan ini membantu untuk mengurangkan risiko berlakunya serangan siber, contohnya seperti *SQL injection*, *Cross Site Scripting* (XSS), dan kecurian data, dengan memastikan kod-kod pengaturcaraan ditulis mengikut amalan keselamatan yang terbaik. Berikut adalah beberapa kategori amalan pengekodan selamat yang boleh diamalkan oleh pengaturcara:

**a) Validasi Input dan Output**

i) Memastikan setiap data yang diterima (input) dan setiap data yang dipaparkan (output) oleh sistem adalah sah, selamat, dan tidak mengandungi elemen berbahaya seperti skrip, kod SQL, atau karakter khas yang boleh dieksploitasi.

ii) Penggunaan validasi input melalui kaedah whitelist dalam pengaturcaraan adalah kritikal bagi memastikan hanya data-data yang dibenarkan sahaja boleh dijadikan sebagai input kepada sistem. Validasi input juga dapat dibuat dengan mengikut jenis data (data type) dan format yang telah ditentukan, contohnya seperti *integer*, *float*, *string*, ataupun format seperti e-mel, nombor telefon dan nombor kad pengenalan.

iii) Validasi output juga boleh dilakukan dengan mensanitasi data sebelum ia dipaparkan kepada pengguna di dalam sistem, contohnya seperti menggunakan fungsi htmlspecialchars() di dalam bahasa pengaturcaraan PHP bagi mengelakkan risiko seperti serangan XSS.

**b) Pengurusan Sesi**

i) Pengurusan sesi (session management) ialah proses mengurus identiti dan aktiviti pengguna semasa mereka melayari sistem. sambungan mereka dengan aplikasi, Proses ini termasuk pengwujudan, penyimpanan, dan penamatan sesi secara selamat, bagi menjamin keselamatan maklumat, kestabilan sistem, serta pengesahan identiti pengguna.

ii) Antara fungsi pengurusan sesi adalah seperti menyimpan data pengguna (contohnya ID pengguna dan tahap akses), mengekalkan status log masuk, mencegah akses tanpa kebenaran dan menyediakan mekanisme untuk log keluar secara automatik (timeout).

**c) Pengurusan Kata Laluan**

i) Pengurusan kata laluan (password management) ialah proses memastikan maklumat kata laluan pengguna disimpan, digunakan dan diuruskan secara selamat untuk mengelakkan akses tidak sah dan kebocoran data.

ii) Amalan terbaik dalam pengurusan kata laluan melibatkan juga (dan tidak terhad) kepada penyimpanan aksara dalam bentuk hash yang selamat dengan menggunakan algoritma yang kukuh, contohnya seperti bcrypt atau Argon2. Selain dari merangkumkan elemen penyimpanan aksara yang selamat, pelaksanaan polisi kata laluan yang kukuh juga dapat memperkasakan lagi tahap sekuriti sistem secara keseluruhan, contohnya seperti penetapan kata laluan yang tidak kurang dari 12 aksara dengan gabungan huruf besar, huruf kecil, nombor dan simbol khas.

**d) Kawalan Muat Naik dan Turun Fail**

Muat naik dan turun fail ialah proses membolehkan pengguna mengedar atau memperolehi semula fail melalui dalam talian. Pengendalian proses ini secara selamat adalah penting bagi mencegah eksploitasi, kebocoran maklumat, atau serangan terhadap pelayan. Antara amalan terbaik bagi pengurusan fail adalah dengan:

i) Melaksanakan proses validasi jenis fail (contohnya dalam format .pdf, .jpg, .docx) dan menetapkan had saiz maksimum fail yang dibenarkan,

ii) Menyimpan fail yang dimuat naik ke dalam direktori khas yang diasingkan daripada direktori sistem atau root aplikasi bagi mengurangkan risiko akses secara terus melalui pelayar dan mengelakkan serangan seperti *path traversal*, dan

iii) Menamakan semula fail secara rawak bagi mengelakkan penggodam merujuk nama fail yang terpapar untuk menjangka nama fail lain dan mengaksesnya secara terus.

**e) Kriptografi**

i) Kriptografi ialah teknik keselamatan yang digunakan untuk melindungi data dengan mengamalkan kaedah penyulitan kepada maklumat-maklumat sensitif agar tidak dapat difahami oleh pihak yang tidak dibenarkan. Ia memainkan peranan penting dalam memastikan kerahsiaan, integriti dan kelestarian data dalam sistem aplikasi tidak tergugat.

ii) Amalan terbaik bagi pelaksanaan kriptografi termasuklah menggunakan algoritma penyulitan moden contohnya seperti *Advanced Encryption Standard* (AES) untuk data rahsia, dan Rivest–Shamir–Adleman (RSA) untuk mekanisme berasaskan kunci awam. Penggunaan algoritma lapuk,

contohnya seperti *Message Digest Algorithm 5* (MD5) dan *Secure Hash Algorithm* 1 (SHA-1), haruslah dielakkan oleh kerana telah diketahui umum, mempunyai pelbagai kelemahan dari segi keselamatan, dan tidak lagi dianggap sebagai selamat.

**f) Log dan Audit**

Log dan audit ialah mekanisme yang digunakan untuk merekod, mengesan, dan menganalisis aktiviti sistem aplikasi, seperti interaksi pengguna dan perubahan data, termasuk aktiviti-aktiviti yang mencurigakan. Mekanisme ini adalah penting bagi tujuan pemantauan keselamatan, pengesanan ralat yang boleh menjejaskan sekuriti sistem, dan penyiasatan ke atas insiden keselamatan siber. Amalan terbaik bagi pengurusan log dan audit adalah seperti berikut:

i) merekod keseluruhan aktiviti utama sistem.

ii) menyimpan log tersebut di lokasi yang selamat.

iii). mengehadkan akses kepada log hanya kepada pegawai yang diberikan kebenaran.

iv) memantau log secara berkala dan menetapkan tempoh simpanan log.

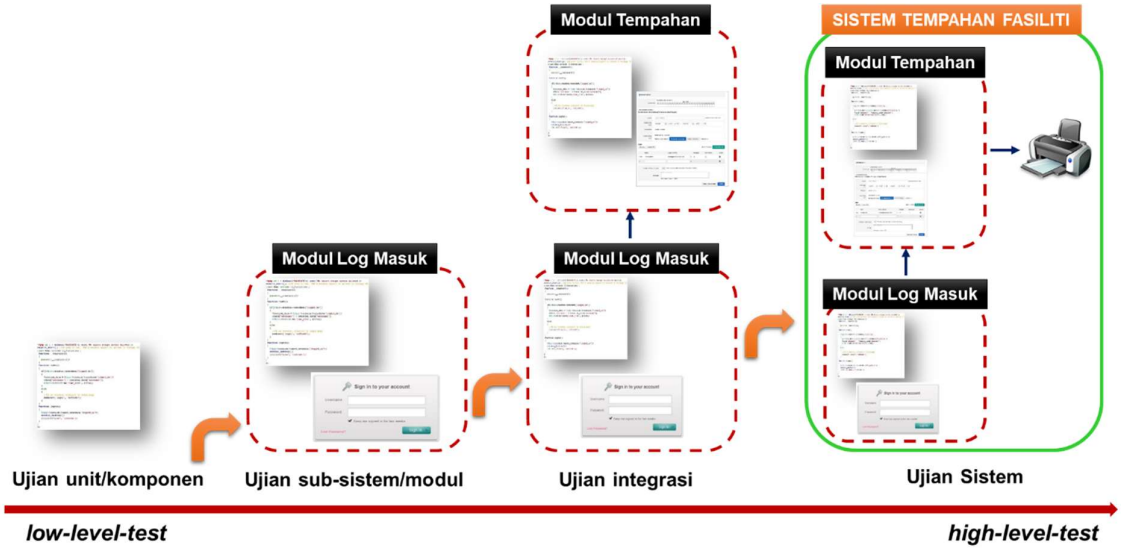
## 5.7 PENGUJIAN SISTEM [F4.3]

### 5.7.1 KETERANGAN

Pengujian Sistem merupakan aktiviti verifikasi yang dilakukan terhadap komponen atau sistem (*test object*) untuk memastikan ia dibangunkan berdasarkan kepada **spesifikasi keperluan dan reka bentuk sistem**. Pengujian sistem merangkumi pelbagai peringkat ujian sebelum sistem diuji secara komprehensif di dalam fasa pengujian penerimaan. Semasa pengujian ini dilaksanakan, ralat yang dikesan akan diperbetulkan dan unit/komponen/modul yang berkaitan akan diuji semula sehingga ralat berjaya diperbaiki.

Jenis-jenis pengujian yang dijalankan adalah **pengujian keperluan fungsian**, **pengujian keperluan bukan fungsian (kualiti)** serta **verifikasi terhadap ralat yang telah dibaiki**. Pengujian ini akan **dilaksanakan oleh Pasukan Pembangun Sistem**.

Peringkat-peringkat ujian yang dilaksanakan mengikut turutan adalah seperti rajah di bawah.




**Rajah 5.3 : Peringkat Pengujian Sistem**

Terdapat dua jenis pendekatan pengujian yang digunakan semasa melaksanakan pengujian ke atas sistem aplikasi iaitu *big bang testing* dan *incremental testing*. *Big bang testing* merupakan pengujian yang dilaksanakan terhadap sistem aplikasi yang telah lengkap dibangunkan, manakala *incremental testing* merupakan satu bentuk ujian modul di mana modul yang akan diuji digabungkan dengan modul yang sudah diuji.

Pendekatan bagi aktiviti pengujian ini dikenali sebagai *incremental testing* iaitu **pengujian dilakukan secara berperingkat** bermula daripada pengujian unit/komponen terkecil sistem aplikasi seperti fungsi, kelas, prosedur dan antara muka (**Ujian unit/komponen**); diikuti dengan ujian modul (**Ujian sub-sistem/modul**); seterusnya menguji dua atau lebih modul/sistem/elemen perkakasan yang disepadukan (**Ujian integrasi**); dan akhirnya semua modul yang terlibat diuji secara menyeluruh (**Ujian sistem**).

Pendekatan ini menerangkan bahawa **ujian tahap rendah (*low-level-test*)** perlu dilaksanakan terlebih dahulu bertujuan untuk mengesahkan bahawa ujian segmen kod sumber telah dilaksanakan dengan betul sebelum ke peringkat ujian berikutnya iaitu, **ujian tahap tinggi (*high-level test*)** yang bertujuan mengesahkan fungsi-fungsi utama sistem aplikasi.

Bagi setiap peringkat pengujian, terdapat empat (4) elemen yang perlu ditetapkan sebelum pelaksanaan ujian iaitu:

a) *Entry Criteria* boleh merujuk kepada dokumen, status/ aktiviti serta tahap pencapaian atau pengukuran yang menjadi pra-syarat untuk melaksanakan sesuatu peringkat pengujian.

b) Aktor merujuk kepada individu/kumpulan yang terlibat dengan sesuatu ujian.

c) Aktiviti ialah aktiviti yang perlu dijalankan semasa ujian.

d) *Exit Criteria* pula merujuk kepada dokumen, status/aktiviti serta tahap pencapaian atau pengukuran yang menjadi syarat untuk menamatkan sesuatu peringkat pengujian

### 5.7.2 OBJEKTIF

Menilai kualiti keseluruhan sistem selepas pembangunan bagi memastikan sistem aplikasi yang dibangunkan sedia untuk diuji di peringkat pengujian penerimaan pengguna.

### 5.7.3 LANGKAH-LANGKAH

**Langkah 1 : Laksana Ujian Unit/Komponen**

a) Tetapkan elemen *Entry Criteria*, aktor, aktiviti dan *Exit Criteria* bagi Ujian Unit/ Komponen seperti jadual di bawah:

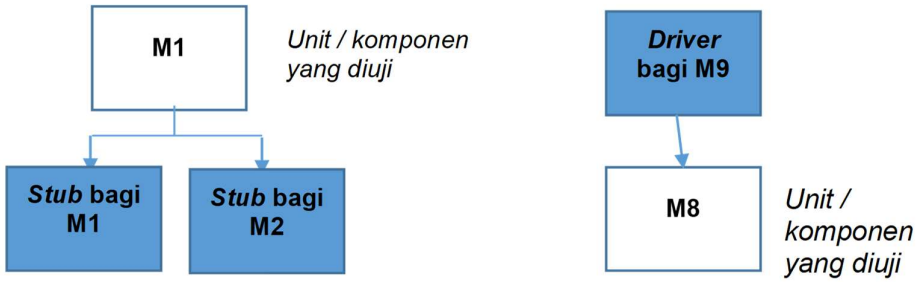
**Jadual 5.2 : *Entry Criteria* dan *Exit Criteria* bagi Ujian Unit/Komponen**

| | |
|---|---|
| ***Entry Criteria*** | i. Dokumen SRS dan SDS telah disahkan<br>ii. Kod sumber bagi unit/komponen telah selesai dibangunkan/ telah diperbaiki bagi tujuan *re-test*<br>iii. Unit/komponen untuk diuji serta *stubs*/*drivers* yang diperlukan telah disediakan dalam persekitaran pembangunan |
| **Aktor** | Pasukan Pembangun Sistem |
| **Aktiviti** | i. Pasukan Pembangun Sistem melaksanakan ujian unit/komponen<br>ii. Pembetulan kod sumber dilakukan bagi ujian yang gagal.<br>iii. Pengujian semula dilaksanakan bagi ujian yang gagal |

| | |
|---|---|
| ***Exit Criteria*** | i. Senarai semak hasil ujian menunjukkan unit/komponen berjaya melaksanakan fungsian atau bukan fungsian seperti yang telah ditetapkan<br>ii. Ralat telah berjaya dibaiki |

b) Kenalpasti unit/komponen adalah bahagian terkecil di dalam sistem aplikasi yang boleh diuji (*testable*) seperti fungsi, kelas, prosedur dan antara muka.

c) Sediakan *stubs dan drivers* yang diperlukan di dalam ujian unit/komponen bagi menggantikan unit/ komponen/modul yang perlu semasa berinteraksi dengan unit/komponen yang diuji. *Stubs* adalah simulasi (*dummy module*) bagi menggantikan unit/komponen selepas (*subordinate/ lower level*) unit/komponen yang diuji. *Drivers* pula adalah simulasi bagi menggantikan unit/komponen sebelum (*upper level*) unit/komponen yang diuji. Rajah di bawah menunjukkan gambaran penggunaan *stubs* dan *drivers* di dalam ujian unit/komponen.




**Rajah 5.4 : Ujian Komponen - Stubs dan Drivers**

d) Laksana ujian unit/komponen bagi memastikan kod program yang dibangunkan memenuhi fungsi unit/komponen dan mengenal pasti ralat di dalam unit/komponen berkenaan. Ujian ini juga hendaklah merangkumi ujian fungsian dan ujian bukan fungsian.

e) Rujuk asas ujian (*test basis*) yang berkaitan iaitu spesifikasi keperluan unit/komponen, dokumen reka bentuk terperinci seperti SRS atau SDS dan kod sumber program semasa melaksanakan ujian bagi memastikan ianya memenuhi keperluan dan reka bentuk unit/ komponen yang telah ditetapkan.

f) Guna kombinasi teknik pengujian yang bersesuaian seperti teknik *white-box testing* dan juga *black box testing*. *White-box testing* merupakan teknik pengujian yang terperinci yang dilakukan terhadap logik dalam dan struktur kod. Teknik ini memerlukan penguji mempunyai pengetahuan penuh tentang kod sumber. Manakala *black box te*sting pula merupakan teknik pengujian berdasarkan kepada spesifikasi sistem aplikasi. Pengujian ini dilaksanakan dengan memasukan *input* dan *output* akan diperiksa sama ada memenuhi fungsi yang telah ditetapkan atau tidak.

g) Uji pengendalian ralat (*error handling*) sekiranya input yang tidak sah diberikan.

h) Baiki ralat yang ditemui dan uji semula unit/komponen berkenaan. Pengujian akan dilaksanakan sehingga unit/ komponen memenuhi *Exit Criteria* yang telah ditetapkan untuk ke peringkat ujian seterusnya iaitu Ujian Sub-Sistem/Modul.

**Langkah 2 : Laksana Ujian Sub-Sistem/Modul**

a) Tetapkan elemen *Entry Criteria*, aktor, aktiviti dan *Exit Criteria* bagi Ujian Sub-Sistem/Modul seperti jadual di bawah:

**Jadual 5.3 : *Entry Criteria* dan *Exit Criteria* bagi Ujian Sub-Sistem/Modul**

| | |
|---|---|
| ***Entry Criteria*** | i. Ujian unit/komponen telah dilaksanakan dengan sempurna<br>ii. *Defects/bugs* yang dilaporkan dalam ujian unit/komponen telah diperbaiki dan diuji semula<br>iii. *Test script* dan *test data* bagi ujian sub-sistem/modul telah disediakan oleh Pasukan Pembangun Sistem |
| **Aktor** | Pasukan Pembangun Sistem |
| **Aktiviti** | i. Pasukan Pembangun Sistem melaksanakan ujian sub-sistem/modul<br>ii. Pembetulan dilakukan bagi ujian sub-sistem/modul yang gagal<br>iii. Pengujian semula dilaksanakan bagi ujian yang gagal |
| ***Exit Criteria*** | i. *Test script* telah diuji<br>ii. Senarai semak hasil ujian menunjukkan sub-sistem/modul berjaya melaksanakan fungsian atau bukan fungsian seperti yang telah ditetapkan<br>iii. Ralat telah berjaya dibaiki |

b) Sediakan *test data*, bangunkan *test script* berdasarkan pertimbangan di bawah bagi memastikan ujian sub-sistem/modul dilaksanakan dengan berkesan:

i) *Module Interface Test*: bertujuan untuk menguji maklumat yang masuk dan keluar daripada modul;

ii) *Local data structures*: Struktur data tempatan diperiksa untuk memastikan data yang disimpan secara sementara dapat mengekalkan integritinya semasa pelaksanaan algoritma.

iii) *Boundary conditions*: bertujuan untuk memastikan modul beroperasi dengan betul di sempadan yang ditetapkan (*to limit or restrict processing*).

iv) *Independent paths:* bertujuan untuk menguji semua *independent paths* yang melalui struktur kawalan bagi memastikan bahawa semua kenyataan dalam modul telah dilaksanakan sekurang-kurangnya sekali.

v) *Error handling paths*: untuk memastikan ralat ditangani dengan betul dan *error handling paths* yang dikenalpasti dapat digunakan selepas melepasi beberapa siri ujian.

c) Laksanakan ujian sub-sistem/modul untuk menguji interaksi di antara unit/komponen dalam modul bagi memastikan ianya dapat berfungsi secara kolektif bagi fungsi, kelas, prosedur dan antara muka yang terlibat di dalam sesebuah modul.

d) Uji setiap modul yang terdapat dalam sistem aplikasi secara berasingan sebelum ianya diuji secara bersepadu di peringkat ujian integrasi.

**Langkah 3 : Laksana Ujian Integrasi**

a) Tetapkan elemen *Entry Criteria*, aktor, aktiviti dan *Exit Criteria* bagi Ujian Integrasi seperti jadual di bawah:

**Jadual 5.4 : *Entry Criteria* dan *Exit Criteria* bagi Ujian Integrasi**

| | |
|---|---|
| ***Entry Criteria*** | i. Ujian sub-sistem/modul telah dilaksanakan dengan sempurna<br>ii. *Defects/bugs* yang dilaporkan dalam ujian sub-sistem/modul telah diperbaiki dan diuji semula<br>iii. Sistem/antara muka sistem luar telah sedia diintegrasi dengan sistem yang diuji (*SUT*).<br>iv. *Test script* dan *test data* bagi ujian integrasi telah disediakan oleh Pasukan Pembangun Sistem<br>v. Penguji Integrasi telah diberikan penerangan |
| **Aktor** | i. Pasukan Pembangun Sistem<br>ii. Penguji Integrasi |
| **Aktiviti** | i. Penguji integrasi melaksanakan ujian integrasi<br>ii. Penguji integrasi merekodkan hasil ujian<br>iii. Pembetulan dilakukan bagi ujian integrasi yang gagal<br>iv. Pengujian semula dilaksanakan bagi ujian yang gagal |
| ***Exit Criteria*** | i. *Test script* telah diuji |

| | |
|---|---|
| | ii. Senarai semak hasil ujian ujian menunjukkan integrasi berjaya melaksanakan fungsian atau bukan fungsian seperti yang telah ditetapkan<br>iii. Ralat telah berjaya dibaiki |

b) Sediakan *test data*, bangunkan *test script* bagi memastikan ujian sub-sistem/modul dilaksanakan dengan berkesan. Objek yang akan dinilai semasa ujian integrasi adalah sub-sistem, pangkalan data, infrastruktur, antara muka serta konfigurasi sistem. Bagi tujuan ini, Pasukan Pembangun Sistem hendaklah mempunyai kefahaman yang jelas terhadap reka bentuk dan keperluan integrasi sebelum ujian ini dilaksanakan.

c) Laksanakan ujian integrasi yang merangkumi ujian fungsian dan ujian bukan fungsian bagi menguji perkara berikut:

i) interaksi di antara sub-sistem/modul dalam sistem aplikasi;

ii) interaksi dalaman di antara sistem operasi, fail sistem dan API perkakasan sistem; DAN

iii) integrasi di antara sistem yang dibangunkan dengan sistem luaran/antara muka luaran (sekiranya ada).

d) Rujuk asas ujian (test basis) yang berkaitan iaitu reka bentuk dan arkitektur sistem serta dokumen **D02 Spesifikasi Keperluan Bisnes**.

e) Laksanakan ujian integrasi dalam beberapa sesi mengikut keperluan integrasi yang wujud dalam sistem aplikasi sehingga memenuhi *Exit Criteria* yang telah ditetapkan untuk ke peringkat ujian seterusnya iaitu Ujian Sistem.

**Langkah 4 : Laksana Ujian Sistem**

a) Tetapkan elemen *Entry Criteria*, aktor, aktiviti dan *Exit Criteria* bagi Ujian Sistem seperti jadual di bawah:

**Jadual 5.5 : Entry Criteria dan Exit Criteria bag Ujian Sistem**

| | |
|---|---|
| ***Entry Criteria*** | 1) Ujian integrasi telah dilaksanakan dengan sempurna<br>2) *Defects/bugs* yang dilaporkan dalam ujian integrasi telah diperbaiki. Tiada lagi *defects/bugs* bagi:<br>a) *Priority High* atau *Medium;* dan<br>b) *Severity Blocking, Critical*, dan *Major*<br>*3) SUT telah disediakan di dalam persekitaran pengujian.* |

| | |
|---|---|
| | 4) *Test script* dan *test data* bagi ujian sistem telah disediakan oleh Pasukan Pembangun Sistem |
| **Aktor** | Pasukan Pembangun Sistem |
| **Aktiviti** | 1. Pasukan Pembangun Sistem melaksanakan ujian sistem<br>2. Pasukan Pembangun Sistem merekodkan hasil ujian<br>3. Pembetulan dilakukan bagi ujian yang gagal<br>4. Pengujian semula dilaksanakan bagi ujian yang gagal |
| ***Exit Criteria*** | a) *Test script* telah diuji<br>b) Hasil ujian menunjukkan sistem berjaya melaksanakan fungsian atau bukan fungsian seperti yang telah ditetapkan.<br>c) Sistem telah berjaya memenuhi keperluan bisnes dan keperluan fungsian<br>d) Ralat telah berjaya dibaiki |

b) Sediakan *test data*, bangunkan *test script* bagi memastikan ujian sistem dilaksanakan dengan berkesan.

c) Laksanakan ujian sistem untuk menentukan keseluruhan sistem aplikasi telah memenuhi spesifikasi yang ditetapkan sebelum ke fasa pengujian penerimaan. Walaubagaimanapun, ujian ini sebaik-baiknya dilaksanakan di dalam persekitaran yang hampir sama dengan persekitaran sebenar (*staging/pre-production).*

d) Rujuk dokumen spesifikasi keperluan, proses bisnes, *Use Case*, laporan analisis risiko, interaksi sistem dengan sistem operasi dan *system resources* semasa melaksanakan ujian sistem.

e) Laksanakan ujian sistem sehingga memenuhi *Exit Criteria* yang telah ditetapkan.

**Langkah 5 : Sediakan Laporan Ujian Sistem**

a) Sediakan Laporan Ujian Sistem sebagai pengesahan aktiviti Pengujian Sistem telah dilaksanakan sepenuhnya. Laporan Ujian Sistem menentukan tahap kesediaan sistem dan merupakan *Entry Criteria* kepada Ujian Penerimaan Pengguna. Format Laporan Ujian Sistem adalah seperti D11 Laporan Ujian Sistem.

b) Laporan Ujian Sistem hendaklah **mempunyai sekurang-kurangnya elemen** berikut:

**Jadual 5.6 : Format Laporan Ujian Sistem**

<table>
<tr><th>Bil.</th><th>Item</th><th>Keterangan</th></tr>
<tr><td>1</td><td>Pengenalan</td><td>Penerangan berkaitan tujuan pelaporan dan skop pelaporan. Skop pelaporan boleh berdasarkan sistem atau modul (sekiranya sistem adalah berskala besar dan mempunyai kompleksiti yang tinggi).</td></tr>
<tr><td>2</td><td>Aktiviti Pengujian</td><td>Serahan<br>Menerangkan bilangan modul/submodul yang terlibat, status lulus dan gagal, serta keterangan yang berkaitan dengannya.<br><br>
<table>
<tr><th>Bil</th><th>Modul & Sub Modul</th><th>Status Pengujian (Lulus/Gagal/KIV)</th><th>Keterangan</th></tr>
<tr><td rowspan="3">1</td><td>Modul Personel</td><td>Lulus</td><td>Selesai</td></tr>
<tr><td>Modul Gaji</td><td>KIV</td><td>2 modul masih KIV untuk pengujian</td></tr>
<tr><td>Modul Pelaporan</td><td>Lulus</td><td>Selesai</td></tr>
</table>
<br>Rumusan Serahan<br>Ringkasan kepada aktiviti pengujian sistem yang telah dilakukan berdasarkan sistem atau modul yang diuji.<br><br>Sekiranya rumusan adalah diperingkat sistem, maka bilangan yang digunakan adalah bilangan modul. Bagi peringkat modul, bilangan yang digunakan adalah bilangan submodul.<br><br>Contoh:<br>
<table>
<tr><th>Bil</th><th>Modul</th><th>Bil Lulus</th><th>Bil Gagal/ KIV</th><th>Keterangan</th></tr>
<tr><td>1</td><td>Modul Personel</td><td>5</td><td>0</td><td>Selesai</td></tr>
<tr><td>2</td><td>Modul Gaji</td><td>5</td><td>2</td><td>2 submodul KIV kerana terdapat isu compliance</td></tr>
<tr><td>3</td><td>Modul Pelaporan</td><td>5</td><td>1</td><td>Submodul Business Intelligence tidak diuji untuk unstructured data</td></tr>
<tr><td>4</td><td>Jumlah</td><td>15</td><td>3</td><td></td></tr>
<tr><td>5</td><td>Peratus Lulus/ Gagal/ KIV</td><td>83.33</td><td>16.67</td><td></td></tr>
</table>
</td></tr>
</table>

| | | |
|---|---|---|
| **3** | **Dokumen Sokongan** | Menyatakan dokumen sokongan yang dirujuk berkaitan dengan Laporan Ujian Sistem yang boleh digunakan oleh pemilik sistem untuk mengesahkan pelaksanaan Ujian Sistem.<br><br>Contoh:<br>Senarai nama penguji, modul/sub-modul yang terlibat dan tarikh selesai pengujian. |

## RUJUKAN


i) Software Engineering: A Practitioner's Approach Eighth Edition, Roger S. Pressman, Ph.D, Bruce R. Maxim, Ph.D, Mc Graw Hill Education.

ii) Software Quality Assurance From theory to implementation, G. Daniel.

iii) Certified Tester Foundation Level Syllabus Version 2018.

iv) Certified Tester Advanced Level Syllabus Test Analyst Version 2012

v) Oracle Database SQL Language Reference, 11g Release 2 (11.2).

vi) MySQL 5.7 Reference Manual.

vii) MongoDB 8.0 Reference Manual.

viii) General Software Development Standard and Guidelines Version 3.5 - Science Infusion Software Engineering Process Group (SISEPG).

ix) Java Language Coding Standard - Sun Microsystem.

x) Sun Microsystems. (Year). Java Language Coding Standard. Diperoleh daripada https://www.oracle.com/java/technologies/javase/codeconventions-contents.html

xi) PHP Framework Interop Group. (2019). PHP Standard Recommendation (PSR). Diperoleh daripada https://www.php-fig.org/psr/

xii) Python Software Foundation. (2001). PEP 8 – Style Guide for Python Code. Diperoleh daripada https://www.python.org/dev/peps/pep-0008/

xiii) OWASP Foundation. (2023). OWASP Secure Coding Practices - Quick Reference Guide. Diperoleh daripada https://owasp.org/