# LANDASAN TEORI

Teori yang digunakan sebagai landasan dalam penelitian ini meliputi pengertian *smartphone*, android, masjid, *user interface*, *user experience*, *java*, *application programming interface* (API), *database*, NoSQL, firebase, *JavaScript Object Notation* (JSON), *Extensible Markup Language* (XML), Unified Modeling Language (UML) dan *metode prototyping*.

## Absensi

Absensi adalah suatu pendataan kehadiran, bagian dari pelaporan aktivitas suatu institusi, suatu komponen institute itu sendiri yang berisi data-data kehadiran yang disusun dan diatur sedemikian rupa sehingga mudah untuk dicari dan dipergunakan apabila sewaktu-waktu di perlukan oleh pihak yang berkepentingan.

## *Flatform Mobile*

*User Interface* merupakan bagian dari sistem informasi yang perlu interaksi dari pengguna untuk membuat *input* dan *output*. Menjelaskan bahwa sebuah sistem informasi baru mempengaruhi banyak dari sistem informasi yang ada lainnya, dan analisis harus memastikan bahwa semua bekerja bersama-sama. Sistem juga harus berinteraksi dengan pengguna baik di dalam maupun di luar organisasi. *User* *Interface* yang lebih dari sekedar layar yaitu, itu adalah serangkai tampilan grafis yang dapat dimengerti oleh pengguna dalam menggunakan sistem, konseptual dan fisik. Dari penjelasan tersebut *User Interface* mempunyai peran yang penting dalam efektivitas suatu sistem informasi. Pembuatan *User Interface* bertujuan untuk menjadikan teknologi informasi tersebut mudah digunakan oleh pengguna [5].

## *Flutter*

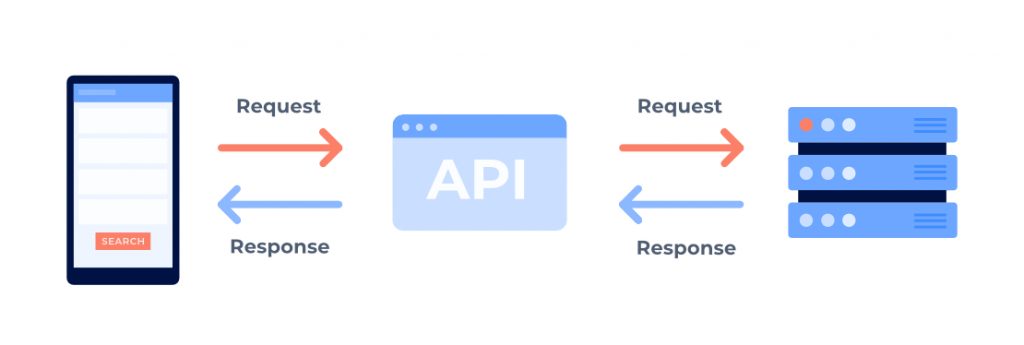
*User Experience* bukanlah tentang cara kerja dari suatu produk atau layanan yang ada. Tetapi bagaimana interaksi antara *user* dengan produk, seperti pengamalan pengguna dalam menggunakan produk, apakah mudah digunakan, sesederhana apa dalam mengoperasikan produk atau layanan hingga pengalaman untuk menemukan, menyerap dan memahami informasi yang tersedia. Hal ini penting karena ketika sebuah produk sedang dikembangkan, biasnya lebih memperhatikan apa yang dilakukannya atau dikembangkan. Sedangkan pengalaman pengguna *user experience* adalah sisi lain yang sering diabaikan, baik dalam segi persamaan hingga bagaimana cara kerjanya. Padahal *user experience* dapat membuat perbedaan antara produk yang sukses dan gagal [5].

## Maps

Java adalah suatu teknologi di dunia *software* komputer. Selain merupakan suatu bahasa pemrograman, Java juga merupakan suatu platform. Java merupakan teknologi dimana teknologi tersebut mencakup java sebagai bahasa pemrograman yang memiliki sintaks dan aturan pemrograman sendiri. Juga mencakup java sebagai platform dimana teknologi ini memiliki *virtual machine* dan *library* yang diperlukan untuk menulis dan menjalankan program yang ditulis dengan bahasa pemrograman java [7].

## API

API atau *Application Programming Interface* adalah sebuah *interface* yang dapat menghubungkan aplikasi satu dengan aplikasi lainnya. API berperan sebagai perantara antar berbagai aplikasi berbeda, baik dalam satu platform yang sama atau lintas platform. API sendiri bisa digunakan untuk komunikasi dengan berbagai [bahasa pemrograman](https://www.niagahoster.co.id/blog/bahasa-pemrograman/" \t "_blank) yang berbeda. Hal ini memudahkan *developer*, dimana *developer* tidak perlu menyediakan semua data sendiri karena cukup mengambil data yang dibutuhkan dari platform lain melalui API [8].

Gambar 2.1 Cara Kerja API

Sumber: (https://www.niagahoster.co.id/blog/api-adalah/)

## SQL

NoSQL merupakan makna dari “*Not Only SQL*” merupakan *database* jenis non-relasional yang menjadi alternatif dari *database* SQL yang merupakan *database* relasional. Perbedaan dari kedua jenis database tersebut terdapat di bentuk skema, SQL memiliki skema yang kaku sedangkan *database* NoSQL memiliki bentuk skema yang lebih fleksibel dan mudah diubah tanpa menggangu sistem yang sedang berjalan [10].

## *Unified Modeling Language* (UML)

*Unified Modelling Language* adalah sebuah teknik pengembangan sistem yang menggunakan bahasa grafis sebagai alat untuk pendokumentasian dan melakukan spesifikasi pada sistem. UML pertama kali di populerkan oleh Grady Booch dan James Rumbaugh pada tahun 1994 untuk mengkombinasikan dua metodologi terkenal yaitu Booch dan OMT, kemudian Ivar Jacobson yang menciptakan *Object Oriented Software Engineering* (OOSE) ikut bergabung. Standar UML dikelolah oleh *Object Management Group* (OMG) [4].

1. *Use Case* Diagram

*Use Case* Diagrammerupakan pemodelan untuk kelakukan (*behavior*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interkasi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Berikut adalah simbol-simbol yang ada pada diagram *Use Case* [4].

Tabel 2.2 Simbol *Use Case* Diagram

|  |  |  |
| --- | --- | --- |
| **Simbol** | **Pengertian** | **Keterangan** |
|  | *Use case* | Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antara unit atau aktor, biasanya dinyatakan dengan menggunakan kata kerja di awal frase nama *use case* |
|  | Aktor | Orang, proses, atau lain yang berinteraksi dengan sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang lain, tapi aktor belum tentu merupaka orang; biasanya dinyatakan menggunakan kata benda di awal frase nama aktor |
|  | Asosiasi | Komunikasi antara aktor dan *use case* yang berpartisipasi pada *use case* atau *use case* memiliki interaksi dengan aktor |
| <<extend>> | Ekstensi | Relasi *use case* tambahan ke sebuah *use case* dimana *use case* yang ditambahkan dapat berdiri sendiri walaupun tanpa *use case* tambahan itu; mirip dengan prinsip *inheritance* pada pemprograman berorientasi objek; biasanya *use case* tambahan memiliki nama depan yang sama dengan *use case* yang ditambahkan. |

Sumber: (Rosa A.S dan M. Shalahuddin, 2016)

1. *Activity* Diagram

*Activity* Diagram merupakan diagram yang menggambarkan *workflow* (alirankerja) atau aktivitas dari sebuah sistem atau proses bisnis atau menu yang ada padaperangkat lunak. Berikut adalah simbol-simbol yang ada pada *activity* diagram [4].

Tabel 2.3 Simbol *Activity* Diagram

|  |  |  |
| --- | --- | --- |
| **Simbol** | **Pengertian** | **Keterangan** |
|  | Status awal | Status awal aktivitas sistem, sebuah diagram aktivitas  memiliki sebuah status awal |
| Aktivitas | Aktivitas | Aktivitas yang dilakukan sistem, aktivitas biasanya di awali dengan kata kerja |
|  | Percabangan | Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu |
|  | Penggabungan | Asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu |
|  | *Swimlane* | Memisahkan organsasi bisnis yang bertanggung |
| State |  | jawab terhadap aktivitas yang terjadi sistem pada waktu tertentu. State dapat berubah jika ada event tertentu yang memicu perubahan tersebut |

Sumber: (Rosa A.S dan M. Shalahuddin, 2016)

1. *Sequence* Diagram

*Sequence* Diagram merupakan diagram yang menggambarkan kelakuan objek pada *Use Case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Berikut adalah simbol-simbol yang ada pada *sequence* diagram [4].

Tabel 2.4 Simbol *Sequence Diagram*

|  |  |
| --- | --- |
| **Simbol** | **Deskripsi** |
| Aktor  Nama actor   |  | | --- | | Nama aktor |   Atau  Tanpa waktu aktif | Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi actor belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal frase nama aktor |
| Garis hidup / lifeline | Menyatakan kehidupan suatu objek |
| Objek  Nama objek: nama kelas | Menyatakan objek yang berinteraksi pesan |
| Waktu aktif | Menyatakan objek dalam keadaan aktif dan berinteraksi, semua yang terhubung dengan waktu aktif ini adalah sebuah tahapan yang dilakukan di dalamnya, misalnya  1: *login*() 2: cek status *login*()    3: *open*()    Maka cek status *login*() dan *open*() dilakukan di dalam metode *login*() Aktor tidak memiliki waktu aktif |
| Pesan tipe create  <<create>> | Menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat |
| Pesan tipe call  1: nama\_metode() | Menyatakan suatu objek memanggil operasi/metode yang ada pada objek lain atau dirinya sendiri,  1: nama\_metode() |

Sumber: (Rosa A.S dan M. Shalahuddin, 2016)

### *Class Diagram*

*Class* Diagram merupakan diagram yang menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi [4].

Tabel 2.5 Simbol *Class* Diagram

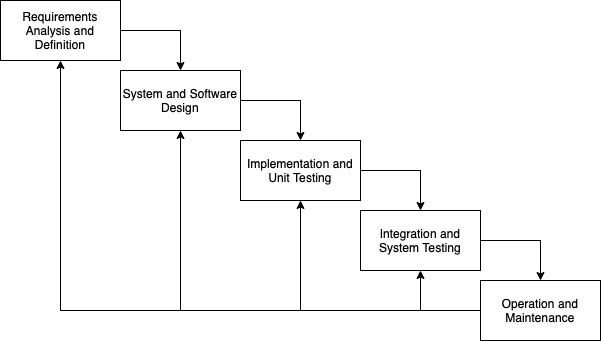
|  |  |  |
| --- | --- | --- |
| **Simbol** | **Pengertian** | **Keterangan** |
| |  | | --- | | Nama\_kelas | | +atribut | | +operasi() | | Kelas | Kelas pada struktur sistem |
|  | Antarmuka | Sama dengan konsep *interface* dalam pemprograman  berorientasi objek |
|  | Asosiasi | Relasi antarmuka dengan makna umum, asosiasi biasanya juga disertai dengan *multiplicity* |
|  | Asosiasi berarah | Relasi antarkelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan *multiplicity* |
|  | Generalisasi | Relasi antarkelas dengan makna generalisasi-spesialisasi (umum khusus) |
|  | Kebergantungan | Kebergantungan antar kelas |
|  | Agregasi | Relasi antarmuka dengan makna semua bagian *(whole-part)* |

Sumber: (Rosa A.S dan M. Shalahuddin, 2016)

## Metode *Waterfall*

Metode *Prototyping* adalah salah satu metode siklus hidup sistem yang didasarkan pada konsep model bekerja (working model). Tujuannya adalah mengembangkan model menjadi sistem final. Artinya sistem akan dikembangkan lebih cepat dari pada metode tradisional dan biayanya menjadi lebih rendah [13].

### Tahapan *Waterfall*



Gambar 2.2 Tahapan *waterfall*

(Sumber : https://medium.com/dot-intern/sdlc-metode-prototype-8f50322b14bf)

Dalam metode pengembangan *waterfall* terdapat tahapan-tahapan yang dilakukan yaitu sebagai berikut.[13]

1. Pengumpulan kebutuhan

Pelanggan dan pengembang bersama-sama mendefinisikan format dan kebutuhan keseluruhan perangkat lunak, mengidentifikasikan semua kebutuhan, dan garis besar sistem yang akan dibuat.

1. Membangun *prototyping*

Membangun *prototyping* dengan membuat perancangan sementara yang berpusat pada penyajian kepada pelanggan (misalnya dengan membuat *input* dan contoh *output*-nya).

1. Evaluasi *protoptyping*

Evaluasi ini dilakukan oleh pelanggan apakah *prototyping* yang sudah dibangun sudah sesuai dengan keinginan pelanggan. Jika sudah sesuai maka langkah keempat akan diambil. Jika tidak, maka *prototyping* diperbaiki dengan mengulang langkah 1, 2 , dan 3.

1. Pengkodean sistem

Dalam tahap ini *prototyping* yang sudah disepakati diterjemahkan ke dalam bahasa pemrograman yang sesuai.

1. Menguji system

Setelah sistem sudah menjadi suatu perangkat lunak yang siap pakai, harus dites dahulu sebelum digunakan. Pengujian ini dilakukan *dengan White Box, Black Box, Basis Path*, pengujian arsitektur dan lain-lain.

1. Evaluasi Sistem

Pelanggan mengevaluasi apakah sistem yang sudah jadi sudah sesuai dengan yang diharapkan . Jika sudah, maka langkah ketujuh dilakukan, jika belum maka mengulangi langkah 4 dan 5.

1. Menggunakan system

Perangkat lunak yang telah diuji dan diterima pelanggan siap untuk digunakan.