# BAB II

# TINJAUAN PUSTAKA

## 2.1 Penilitian Terkait

Penelitian yang dilakukan oleh (Arief Indrawan Putra, Anita Diana, 2020) tujuan penelitian ini mengembangkan sistem *e-commerce* berbasis web berbasis WordPress untuk bisnis parfum. Salah satu masalah dengan proses bisnis konvensional adalah kesalahan pencatatan data, waktu yang terbatas untuk transaksi, dan proses pemesanan yang tidak efisien. Tujuan dari penelitian ini adalah untuk mengatasi masalah ini. Dengan menggunakan model bisnis Canvas (BMC), sistem ini dapat mengubah model bisnis konvensional menjadi *e-commerce*, meningkatkan efisiensi transaksi, menyediakan laporan penjualan, dan membuat pelanggan lebih mudah memesan produk secara *online*[4].

Dalam penelitian yang juga dilakukan oleh (Grinaldi Wisnu Tri Prasetyo, Fajar Pradana, Bondan Sapta Prakoso, 2022) mengembangkan aplikasi "Point of Sale" WarunkQu yang membantu usaha kecil dan menengah (UMKM) melaporkan keuangan dan mencatat penjualan. Aplikasi ini dikembangkan menggunakan *framework Flutter* dan metode pengembangan *waterfall*. Sistem ini dimaksudkan untuk meningkatkan produktivitas proses manual yang sering memakan waktu lama dan rentan terhadap kesalahan. Hasil pengujian menunjukkan bahwa aplikasi bekerja dengan sangat baik dan sangat diterima oleh pengguna, memiliki tingkat

keberhasilan 100% berdasarkan pengujian kotak hitam dan nilai usabilitas rata-rata 91%[5].

Penelitian lain oleh(Andi, 2020) penelitian ini menghasilkan aplikasi Android berbasis *framework Flutter* yang memungkinkan pelanggan melakukan pemesanan sebelum tiba di lokasi, dengan tujuan mengurangi antrean. Aplikasi memungkinkan pelanggan melihat jumlah transaksi dan menggunakan kode pemesanan saat tiba di lokasi untuk mempercepat proses pelayanan. Pelanggan dan administrator dapat berkomunikasi dengan sistem melalui API Web *Service*. Hasil penelitian menunjukkan bahwa aplikasi bekerja dengan baik dan dapat mencapai tujuan, yaitu mengurangi antrean dan meningkatkan efisiensi layanan[6].

## 2.2 Landasan Teori

### 2.2.1. Android

Aplikasi adalah sebuah *software* (perangkat lunak) yang berfungsi sebagai *frontend* pada sebuah sistem dan berfungsi untuk membantu berbagai tugas manusia.

Menurut Sri Widianti, Aplikasi didefinisikan sebagai program berbentuk perangkat lunak yang berjalan pada sistem tertentu dan berfungsi untuk mengelolah berbagai macam data sehingga menjadi informasi yang bermanfaat bagi pengguna dan sistem yang terkait[7]A*ndroid Studio* adalah salah satu IDE untuk pengembangan aplikasi android, yang didasarkan pada Intellij IDE. Selain sebagai

code editor, *Android Studio* juga menawarkan banyak fitur yang dapat meningkatkan produktivitas seorang *developer* dalam mengembangkan aplikasi Android[8].

Gambar 2.1 Logo Android

### 2.2.2. *Framework*

Dart adalah bahasa pemrograman yang dikembangkan oleh Google yang memungkinkan pengembang membuat aplikasi *modern* yang mudah dipelajari dan efisien yang berjalan di berbagai *platform*, seperti web, *mobile*, desktop, dan server. Fitur-fiturnya yang canggih dan fleksibel mendukung produktivitas pengembang.

### 2.2.3. *MySQL*

*MySQL* adalah sistem manajemen basis data (DBMS) yang berfungsi sebagai sistem sosial. Selain itu, *MySQL* adalah aplikasi sumber dan fitur server *MySQL* yang cepat, kuat, dan mudah digunakan. Digabungkan untuk menunjukkan proses konversi *Database*, *MySQL* dapat bekerja dengan server atau sistem pemasaran apa pun[9].


Gambar 2.2 Logo *MySQL*

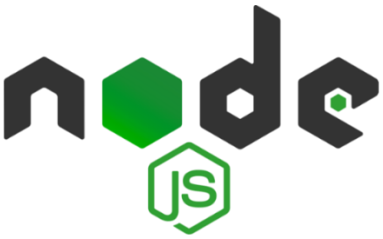
### 2.2.4. *Visual Studio Code*

Untuk mengembangkan aplikasi, *developer* sering menggunakan editor teks *Visual Studio Code* karena memiliki banyak fitur yang dapat membantu mereka mengembangkan aplikasi lebih cepat dengan beban yang lebih sedikit.


Gambar 2.3 Logo *Visual Studio Code*

### 2.2.5. Node Js

Node.js adalah *runtime JavaScript* yang berbasis pada mesin V8 milik Google Chrome. Dimaksudkan untuk menjalankan kode *JavaScript* di sisi server, memungkinkan pengembang membangun aplikasi server yang cepat, ringan, dan efisien.



Gambar 2.4 Logo Node Js

### 2.2.6. *Flutter*

*Flutter* adalah metode utama untuk membuat aplikasi untuk sistem operasi Google Fuchsia. Selain itu, *Flutter* adalah *Software Development Kit* (SDK) untuk pengembangan aplikasi seluler sumber terbuka yang dikembangkan dan disponsori oleh Google. *Skia Graphics Engine* digunakan untuk membuat *Flutter*, yang ditulis dalam bahasa C++, Dart, dan C[10].


Gambar 2.5 Logo *Flutter*

### 2.2.7. *Laravel*

*Laravel* adalah *framework* PHP yang dirilis di bawah lisensi MIT dan dibangun dengan konsep *model view controller* (MVC). *Laravel* adalah pengembangan website berbasis MVP yang ditulis dalam PHP dan dimaksudkan untuk meningkatkan kualitas perangkat lunak dengan mengurangi biaya pengembangan awal dan pemeliharaan serta meningkatkan pengalaman bekerja dengan

aplikasi dengan sintaks yang ekspresif, jelas, dan menghemat waktu[11].


Gambar 2.6 Logo *Laravel*

### 2.2.8. *Database*

*Database* adalah kumpulan data yang terhubung dan dibuat sesuai kebutuhan, sehingga mudah untuk mengubah, mengambil, dan mencari data yang disimpan[12].

### 2.2.9. UML

Salah satu metode dalam teknik rekayasa perangkat lunak adalah *Unified Modeling Language* (UML). UML menggambarkan alur dan cara kerja sistem, serta fungsi, tujuan, dan mekanisme kontrol sistem[13].

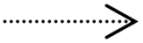
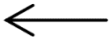
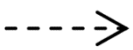
Di dalam UML terdapat 3 teknik perancangan sistem, model-model tersebebut yaitu : *Use case Diagram*, *Sequence Diagram*, dan *Class Diagram.*

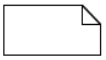

Gambar 2.7 Logo UML (*Unified Modeling Language*)

### 2.2.10. *Use case* Diagram

*Use case* Diagram adalah gambaran tentang fungsi yang diharapkan dari sebuah sistem dan menunjukkan interaksi antara aktor dan sistem. Dalam kasus penggunaan, ada aktor yang merupakan representasi dari entitas manusia atau sistem yang melakukan tugas untuk sistem[14]. Simbol *use case* diagram bisa dilihat pada tabel 2.1.

Tabel 2. 1 Simbol *Usecase* Diagram

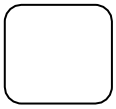
| No | Gambar | Nama | Keterangan |
|---|---|---|---|
| 1. |  | *Actor* | Menspesifikasikan himpunan peran yang pengguna mainkan ketika berinteraksi dengan *use case*. |
| 2. |  | *Dependency* | Hubungan di mana perubahan yang terjadi pada suatu elemen mandiri (*independent*) akan mempengaruhi elemen yang bergantung padanya elemen yang tidak mandiri (*independent*). |
| 3. |  | *Generalization* | Hubungan dimana objek anak (*descendent*) berbagi perilaku dan struktur data dari objek yang ada diatasnya objek induk (*ancertor*). |
| 4. |  | *Include* | Menspesifikasikan bahwa *use case* sumber secara eksplisit. |
| 5. |  | *Extend* | Menspesifikasikan bahwa *use case* target memperluas perilaku dari *use case* sumber pada suatu titik yang diberikan. |
| 6. |  | *Association* | Apa yang menghubungkan antara objek satu dengan objek lainnya. |
| 7. |  | *System* | Menspesifikasikan paket yang menampilkan sistem secara terbatas. |

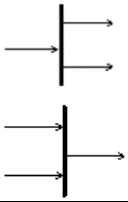
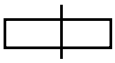
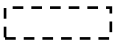
| No | Gambar | Nama | Keterangan |
|---|---|---|---|
| 8. |  | *Use case* | Deskripsi dari urutan aksi – aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu aktor. |
| 9. |  | *Collaboration* | Interaksi aturan – aturan dan elemen lain yang bekerja sama untuk menyediakan perilaku yang lebih besar dari jumlah dan elemen – elemennya (sinergi). |
| 10. |  | *Note* | Elemen fisik yang eksis saat aaplikasi dijalankan dan mencerminkan suatu sumber daya komputasi. |

### 2.2.11. *Activity* Diagram

*Activity* Diagram (Diagram Aktivitas) adalah Salah satu diagram UML yang paling populer, yang dapat digunakan untuk menunjukkan komunikasi data atau mengontrol urutan tindakan yang dilakukan[15]. Komponen simbol dalam *activity* diagram bisa dilihat pada tabel 2.2.

Tabel 2. 2 Simbol *Activity* Diagram

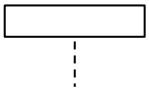
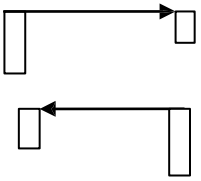
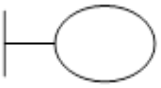
| No | Gambar | Nama | Keterangan |
|---|---|---|---|
| 1. |  | *Activity* | Memperlihatkan bagaimana masing - masing kelas antarmuka saling berinteraksi satu sama lain. |
| 2. |  | *Action* | State dari sistem yang mencerminkan eksekusi suatu aksi. |
| 3. |  | *Initial Node* | Bagaimana objek dibentuk atau diawali. |
| 4. |  | *Final Node* | Bagaimana objek dibentuk dan dihancurkan. |

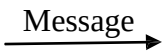
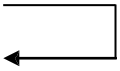
| No | Gambar | Nama | Keterangan |
|---|---|---|---|
| 5. |  | *Fork Node* | Satu aliran yang pada tahap tertentu berubah menjadi beberapa aliran. |
| 6. |  | *Decision* | Pilihan untuk mengambil keputusan |
| 7. |  | *Fork/Join* | Digunakan untuk menunjukkan kegiatan yang dilakukan secara paralel atau untuk menggabungkan dua kegiatan paralel menjadi satu |
| 8. |  | *Rake* | Menunjukkan adanya dekomposisi |
| 9. |  | *Time* | Tanda waktu |
| 10. |  | *Send* | Tanda pengiriman |

### 2.2.12. *Sequence* Diagram

Salah satu jenis diagram UML yang dapat menjelaskan urutan waktu pemrosesan sistem adalah *sequence* diagram[16]. komponen simbol *sequence* bisa dilihat pada tabel 2.3.

Tabel 2. 3 Simbol *Sequence* Diagram

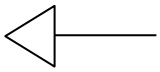
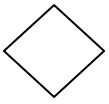
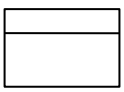
| No | Gambar | Nama | Keterangan |
|---|---|---|---|
| 1. |  | *LifeLine* | Objek *entity*, antar muka yang saling berinteraksi. |
| 2. |  | *Message* | Spesifikasi dari komunikasi antar objek yang memuat informasi – informasi tentang aktifitas yang terjadi. |
| 3 |  | *Actor* | Menggambarkan orang yang sedang berinteraksi dengan sistem |
| 4 |  | *Boundary Class* | Menggambarkan penggambaran dari *form* |

| No | Gambar | Nama | Keterangan |
|---|---|---|---|
| 5 |  | *Entity Class* | Mengambarkan hubungan kegiatan yang akan dilakukan |
| 6. |  | *Control Class* | Menggambarkan penghubung antara *Boundary* dengan tabel |
| 7 |  | *Activation* | Sebagai sebuah objek yang akan melakukan sebuah aksi |
| 8 |   | *Message* | Mengindikasikan komunikasi antara objek dengan objek |
| 9 |  | *Self Message* | Menginndikasikan komunikasi kembali kedalam sebuah objek itu sendiri |

### 2.2.13. *Class* Diagram

*Class* Diagram adalah struktur dan penjelasan dari *class*, *package*, dan objek serta hubungannya satu sama lain seperti *containment, inheritance*, dan *associations*[15]. Untuk simbol dalam *class* diagram bisa dilihat pada tabel 2.4.

Tabel 2. 4 *Class* Diagram

| No | Gambar | Nama | Keterangan |
|---|---|---|---|
| 1. |  | *Generalization* | Hubungan di mana objek anak (*descendent*) berbagi perilaku dan struktur data dari objek yang ada di |
| 2. |  | *Nary Association* | Upaya untuk menghindari asosiasi dengan lebih dari 2 objek. |
| 3. |  | *Class* | Himpunan dari objek - objek yang berbagi atribut serta |

| No | Gambar | Nama | Keterangan |
|---|---|---|---|
| 4. |  | *Collaboration* | Deskripsi dari urutan aksi - aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur |
| 5. |  | *Dependency* | Operasi yang benar - benar dilakukan oleh suatu objek. |
| 6. |  | *Dependency* | Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri (*independent*) akan mempegaruhi elemen yang |
| 7. |  | *Association* | Apa yang menghubungkan antara objek satu dengan objek lainnya. |