BAB II
LANDASAN TEORI

A. SiTata

SiTata merupakan aplikasi *mobile* yang dikembangkan oleh Kantor Pertanahan Kota Banjarbaru menggunakan *framework flutter* untuk platfrom *android*. Aplikasi ini ditujukan sebagai sarana penyampaian informasi tentang pertanahan khususnya untuk wilayah kota Banjarbaru. Dalam aplikasi SiTata berisi layanan informasi persyaratan berkas-berkas yang perlukan dalam kepengurusan pertanahan, layanan konsultasi online maupun offline, serta informasi peta Rencana Tata Ruang Wilayah.

B. *Flutter*

Flutter merupakan *Software Development Kit* yang digunakan dalam pengembangan aplikasi *mobile. Flutter* dikembangkan oleh *Google* sebagai *framework* untuk pengembangan aplikasi yang mampu berjalan pada sistem operasi *Android* dan *IOS*. *Flutter* dikembangkan dalam bahasa C, C++, *Dart* dan *Skia*. Pada *Flutter,* kode yang ditulis akan di-*compile* menjadi kode *native* (*Android NDK, LLVM, AOT-Compiled*) tanpa menggunakan *interpreter* sehingga proses *compile* berjalan lebih cepat [10].

C. Roya

Menurut J. Satrio, roya diartikan sebagai penghapusan catatan beban [3]. Dalam Pasal 22 ayat (1) Undang-Undang Nomor 4 Tahun 1996 tentang Hak Tanggungan Atas Tanah Beserta Benda-Benda yang Berkaitan Dengan Tanah menyebutkan bahwa roya sama dengan pencoretan pencatatan terhadap Akta Pemberian Hak Tanggungan yang berisi beberapa objek hak tanggungan yang dijaminkan. Maka dalam akta tersebut, dicantumkan pula perjanjian roya untuk sebagian objek hak tanggungan (roya parsial) yang telah dilunasi pembayaran hutangnya [11].

Keberadaan roya berkaitan dengan hak tanggungan. Sebab, dihapusnya hak tanggungan karena pelunasan hutang perlu diikuti dengan penghapusan beban hak tanggungan dari pencatatan di buku tanah tentnag hak atas tanah yang menjadi objek hak tanggungan [4].

D. *Bug*

Secara umum *Bug* diartikan sebagai kesalahan yang terjadi pada sebuah perangkat lunak [5]. Menurut Matthew dan Yuan, *bug* merupakan kondisi di mana sesuatu tidak seharusnya terjadi atau dilakukan oleh perangkat lunak atau kondisi di mana suatu perangkat lunak melakukan sesuatu yang seharusnya tidak dilakukan [12]. Semakin besar skala pengembangan suatu perangkat lunak, semakin besar pula peluang terjadinya suatu *bug* [13]. *Bug* terdiri dari beberapa macam, yaitu:

1. *Bug* Aritmatika, yaitu *bug* yang terjadi akibat kesalahan perhitungan di dalam kode perangkat lunak.
2. *Bug* Logika, yaitu *bug* yang diakibatkan kesalahan logika program seperti kesalahan dalam logika perulangan yang menjadi tak terbatas.
3. *Bug syntax,* yaitu *bug* yang terjadi karena kesalahan penulisan kode program.
4. *Bug* sumber daya, yaitu *bug* yang terjadi karena perangkat keras yang digunakan saat aplikasi dijalankan tidak mengimbangi kebutuhan sehingga terjadi *buffering.*
5. *Interfacing Bug,* yaitu *bug* yang terjadi pada antar muka yang ada, seperti pada API, perangkat keras, atau protokol yang digunakan.
6. *Multi-threading programming bug*, yaitu *bug* yang terjadi pada perangkat lunak yang dijalankan secara *multi-threading*. Salah satu contohnya adalah *deadlock.*
7. *Teamworking bug,* merupakan jenis *bug* yang muncul saat suatu perangkat lunak dikembangkan oleh banyak individu. Semakin banyak individu yang mengerjakan suatu perangkat lunak, semakin besar pula kemungkinan *bug* jenis ini muncul.

E. *Debugging*

Debugging merupakan tahapan pencarian kesalahan atau kerusakan di dalam perangkat lunak yang dikembangkan untuk meningkatkan kinerja perangkat lunak. Dalam proses *debugging,* kinerja perangkat lunak akan dievaluasi apakah sudah berjalan sesuai kebutuhan [6]. *Debugging* menjadi salah satu tahapan penting dalam pengembangan suatu perangkat lunak. Beberapa manfaat yang dapat proses *debugging* adalah [14]:

1. *Error* dapat dideteksi lebih awal sehingga proses perbaikan dapat dilakukan lebih awal.
2. Menghindari desain program yang tidak sesuai.
3. Memberikan informasi dari struktur data program.
4. Membantu program untuk tidak menambahkan informasi yang tidak berguna.
5. Mengurangi ancaman peretasan akibat celah dari *bug* atau error.
6. Menghidari proses *testing* yang rumit sehingga waktu yang digunakan oleh developer saat pengembangan dapat dikurangi.

Proses *debugging* dimulai dengan mereproduksi *bug* untuk mengetahui *bug* apa saja yang perlu dituntaskan. Selanjtunya dilakukan identifikasi *bug* untuk mengetahui jenis *bug* apa yang ditemukan. Mengidentifikasi bug dapat dilakukan dengan melihat pesan *error* yang muncul pada *log* di terminal IDE. Setelah jenis *bug* diketahui, dilanjutkan dengan mencari lokasi di mana *bug* terjadi. Lokasi *bug* dapat berada di kode tombol, navigasi program, atau kode fungsi yang dipanggil. Setelah lokasi *bug* di ketahui, *bug* perlu dianalisis penyebabnya dan mencari tahu apakah *bug* yang dimaksud diakibatkan atau mengakibatkan *bug* lainnya. Kemudian dilakukan pembuktian dari analisis *bug.* Apabila analisis sudah dipastikan, maka dilakukan *debugging* pada semua *error*. Tahap terakhir dari proses ini adalah memperbaiki dan memvalidasi kode program untuk memastikan *error* tidak lagi terjadi [14].