UNIVERSITAS GUNADARMA

PENULISAN ILMIAH

Perancangan Aplikasi Mobile Trading
Menggunakan Unified Modeling Language

Nama	: Wisnu Indarto
NPM	: 14101122
Jurusan	: Sistem Informasi
Pembimbing	: Dr. Ana Kurniawati

Ditulis guna melengkapi sebagian syarat
Untuk mencapai jenjang setara Sarjana Muda

Universitas Gunadarma

2010

PERNYATAAN ORIGINALITAS DAN PUBLIKASI

Saya yang bertanda tangan di bawah ini,

Nama	:	Wisnu Indarto
NPM	:	14101122
NIRM	:	
Judul PI	:	Perancangan Aplikasi Mobile Trading Menggunakan Unified Modeling Language
Tanggal Sidang	:	
Tanggal Lulus	:	

Menyatakan bahwa tulisan ini adalah merupakan hasil karya saya sendiri dan dapat dipublikasikan sepenuhnya oleh Universitas Gunadarma. Segala kutipan dalam bentuk apa pun telah mengikuti kaidah, etika yang berlaku. Mengenai isi dan tulisan adalah merupakan tanggung jawab Penulis, bukan Universitas Gunadarma.

Demikian pernyataan ini dibuat dengan sebenarnya dan dengan penuh kesadaran.

Depok, 1 November 2010

(Wisnu Indarto)

LEMBAR PENGESAHAN

Judul PI	:	Perancangan Aplikasi Mobile Trading Menggunakan Unified Modeling Language
Nama	:	Wisnu Indarto
NPM	:	14101122
NIRM	:	
Tanggal Sidang	:	
Tanggal Lulus	:	

Menyetujui

Pembimbing	Kasubag Bidang PI
(Dr. Ana Kurniawati)	(Sri Nawangsari, SE, MM)

Ketua Jurusan SI

(Dr. Setia Wirawan)

ABSTRAK

Wisnu Indarto.14101122
Perancangan Aplikasi Mobile Trading Menggunakan Unified Modeling Language
PI. Sistem Informasi, Direktorat Program Diploma Tiga Teknologi Informasi, Universitas Gunadarma, 2010
Kata Kunci : Aplikasi, Perancangan, UML, Mobile Trading

(xi + 61)

Penulisan Ilmiah ini bercerita tentang bagaimana aplikasi online trading dirancang untuk kemudahan *investor* menggunakan *Unified Modeling Language*.
Aplikasi yang dirancang ini akan digunakan untuk berbagai kebutuhan yang berkaitan dengan perdagangan saham. Untuk itu aplikasi perlu menyajikan segala informasi yang penting dan relevan kepada user selain fungsi utamanya untuk memfasilitasi transaksi perdagangan saham, sehingga aplikasi ini memiliki kemampuan untuk menampilkan informasi *Market Info*, *Messages* dan *Portfolio*

Daftar Pustaka (2003-2010)

KATA PENGANTAR

Segala puji dan syukur penulis panjatkan ke hadirat Allah SWT yang telah memberikan berkat, anugerah dan karunia yang melimpah sehingga penulis dapat menyelesaikan Penulisan Ilmiah ini.

Penulisan ilmiah ini disusun guna melengkapi sebagian syarat untuk memperoleh Gelar Ahli Madya pada Direktorat Program Diploma Tiga Teknologi Informasi Program Studi Sistem Informasi Universitas Gunadarma. Adapun judul Penulisan Ilmiah ini adalah "Perancangan Aplikasi Mobile Trading Menggunakan Unifide Modelling Language".

Walaupun banyak kesulitan yang penulis harus hadapi ketika menyusun penulisan ilmiah ini, namun berkat bantuan dan dorongan dari berbagai pihak akhirnya penulisan ilmiah ini dapat diselesaikan dengan baik. Untuk itu penulis tidak lupa mengucapkan terima kasih kepada:

- Ibu Prof. Dr. E. S. Margianti, SE, MM, selaku Rektor Universitas Gunadarma.
- Bapak Prof. Dr. A. Benny Mutiara, selaku Dekan Fakultas Teknologi Informasi.
- Bapak Dr. Setia Wirawan, selaku Ketua Jurusan Sistem Informasi.
- Ibu Dr. Ana Kurniawati, selaku Sekretaris Jurusan Sistem Informasi dan Pembimbing. Terimakasih telah bersedia meluangkan waktunya untuk membimbing penulis dalam menyelesaikan Penulisan Ilmiah ini
- Ibu Dyah Cita Irawati SSI, MM, Pembimbing Akademik, terimakasih atas bantuan, dorongan dan perhatiannya sehingga penulis dapat menyelesaikan Penulisan Ilmiah ini.
- Bapak dan Mamah Hadijanto, juga Mas Sigit, Mba Indri, Meity dan Ponti tercinta yang memberikan dukungan dan perhatian serta memberikan ekstra dorongan kepada penulis dengan tidak bosan-bosannya. I Love U.
- Bapak dan Mamah Karna, dan seluruh keluarga tercinta yang selalu mengingatkan dan sabar menunggu kelulusan penulis.

- Mami dan Mas Al yang menjadi dorongan utama terselesaikannya Penelitian Ilmiah ini. Mudah-mudahan Papi bisa lulus ya.
- Berbagai pihak yang tidak dapat disebutkan satu per satu. Terimakasih atas dukungan dan semangat yang kalian berikan

Akhir kata hanya kepada Allah jualah segalanya dikembalikan dan penulis sadar bahwa penulisan ilmiah ini masih jauh dari sempurna disebabkan karena berbagai keterbatasan yang penulis miliki. Untuk itu penulis mengharapkan kritik dan saran yang bersifat membangun untuk menjadi perbaikan di masa yang akan datang.

Depok, November 2010

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

1. PENDAHULUAN

1.1. Latar Belakang

Kemajuan teknologi bergerak sesuai dengan kemajuan zaman dan kemampuan serta tuntutan kebutuhan manusia. Dalam era 80-an misalnya kemajuan teknologi dibidang komputer masih belum merasuk dalam kegiatan sehari-hari kita. Dan setelah era tersebut, kita dapat melihat perkembangan dan perubahan yang sangat cepat.

Dengan kemajuan tersebut, computer dapat mendukung kegiatan disemua bidang sebagai alat bantu untuk mendukung evaluasi, analisis serta efisiensi dan efektifitas dalam proses pengambilan keputusan dan melakukan transaksi.

Pada tahun 2002, Bursa Efek Indonesia, yang saat itu masih bernama Bursa Efek Jakarta mulai mengaplikasi *sistem* perdagangan jarak jauh yang disebut dengan Remote Trading. Pada remote trading, setiap order transaksi di kantor broker (perusahaan efek) langsung dikirim ke *sistem* perdagangan Bursa Efek (*sistem* JATS), tanpa perlu memasukkan order dari lantai bursa (trading floor). Dengan demikian, order dapat dilakukan di kantor broker di mana saja sepanjang terhubung dengan *sistem* perdagangan bursa.

Dengan menggunakan remote trading, *investor* diharapkan mendapatkan beberapa manfaat, antara lain:

- Proses transaksi menjadi lebih cepat
- Konfirmasi menjadi lebih cepat
- Order *investor* diluar kota dapat langsung dieksekusi ke sistem perdagangan bursa

Banyak parameter yang diperlukan seorang *investor* dalam mengambil keputusan untuk melakukan transaksi. Dalam remote trading, biasanya seorang *investor* biasanya memonitor aplikasi Market Info untuk mengetahui dan menganalisa keadaan pasar maupun stok yang akan diperdagangkan. Selanjutnya, *investor* harus berkomunikasi dengan sales untuk melakukan transaksi. Kegiatan

tersebut masih cukup menyita waktu sehingga terkadang kondisi pasar sudah berubah saat transaksi dilakukan oleh Sales.

Dengan latar belakang masalah tersebut maka selanjutnya BEI mengatur penyelenggaraan fasilitas penyampaian pesanan secara langsung berbasis pada remote trading yang biasa disebut *online trading* atau *mobile trading* bila menggunakan handphone atau smartphone. Pada *online trading* maupun *mobile trading*, *investor* dapat langsung melakukan transaksi menggunakan PC maupun handphone melalui aplikasi yang dibuat oleh perusahaan broker.

Rancangan aplikasi *online trading* maupun *mobile trading* diserahkan pada broker yang ingin mengimplementasikan sistem tersebut. Untuk itu penulis mencoba membuat rancangan aplikasi *mobile trading* untuk mempermudah *investor* dalam melakukan transaksi.

1.2. Batasan Masalah

Pada penulisan ilmiah ini, penulis akan membahas rancangan aplikasi *Mobile Trading* menggunakan *Unified Modeling Language* (UML), terbatas pada perancangan *client* menggunakan Bisnis Model (*Use Case*), *Static Structure* (*Class Diagram*), *Dynamic Behaviour* (*Sequence Diagram*, *Activity Diagram*), dan *Data Structure*.

1.3. Tujuan Penulisan

Tujuan dari penulisan ini adalah untuk merancang aplikasi *Mobile Trading* menggunakan *Unified Modeling Language* (UML).

1.4. Metode Penulisan

Dalam penulisan ilmiah ini, penulis menggunakan metode penulisan studi pustaka yaitu Membaca panduan dari BEI, buku-buku dari perpustakaan, buku diktat kuliah, dan artikel-artikel yang berhubungan dengan penulisan ini.

1.5. Sistematika Penulisan

Untuk memperjelas arah dan mempermudah rangkaian pembahasan, maka diberikan gambaran garis besar mengenai pokok-pokok yang dibahas dalam urutan pembahasan secara sistematis.

Adapun pembahasan penulisan ilmiah ini dibagi dalam empat bab, yaitu:

1. Bab 1 (Pendahuluan), bab ini terdiri dari latar belakang penulisan, batasan masalah, tujuan penulisan, metode penulisan dan sistematika penulisan.
2. Bab 2 (Landasan Teori), bab ini berisikan teori-teori yang digunakan oleh penulis, baik itu teori dari perangkat rancangan UML, struktur navigasi maupun logika hasil pemikiran penulis.
3. Bab 3 (Pembahasan), bab ini membahas penggunaan UML dan struktur navigasi dalam merancang *Aplikasi Mobile Trading*.
4. Bab 4 (Penutup), pada bab terakhir ini berisi kesimpulan tentang rancangan *sistem* yang telah dibuat beserta saran–saran.

2. LANDASAN TEORI

2.1. Unified Modelling Language

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem.

Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan operation dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C.

Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan syntax/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML syntax mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (Object-Oriented Design), Jim Rumbaugh OMT (Object Modeling Technique), dan Ivar Jacobson OOSE (Object-Oriented Software Engineering).

2.1.1. Use Case Diagram

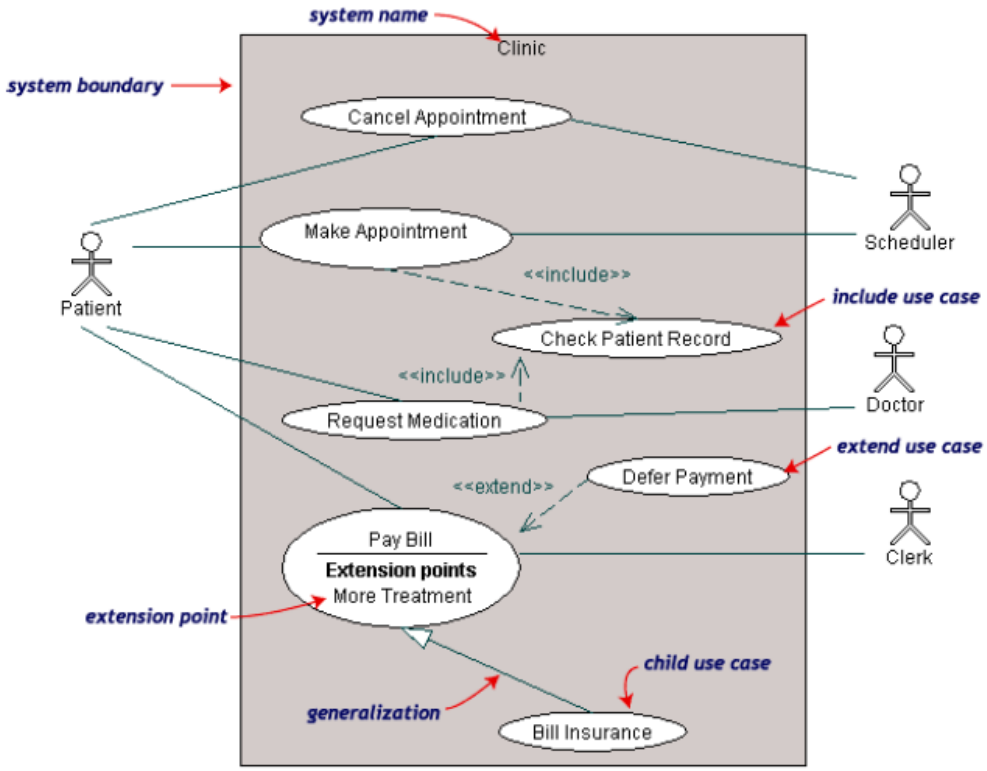
Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah use case merepresentasikan sebuah interaksi antara aktor dengan sistem. Use case merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-create sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

Use case diagram dapat sangat membantu bila kita sedang menyusun requirement sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang test case untuk semua feature yang ada pada sistem.

Sebuah use case dapat meng-*include* fungsionalitas use case lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa use case yang di-*include* akan dipanggil setiap kali use case yang meng-*include* dieksekusi secara normal.

Sebuah use case dapat di-*include* oleh lebih dari satu use case lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang common.

Sebuah use case juga dapat meng-*exted* use case lain dengan behaviour-nya sendiri. Sementara hubungan generalisasi antar use case menunjukkan bahwa use case yang satu merupakan spesialisasi dari yang lain.



Gambar 2.1 Use Case Diagram

2.1.2. Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan rancangan berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

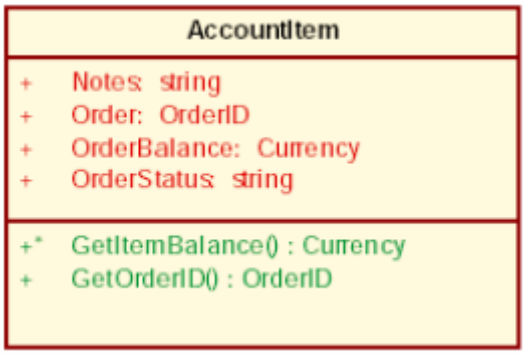
Class diagram menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti containment, pewarisan, asosiasi, dan lain-lain.

Class memiliki tiga area pokok :

1. Nama (dan stereotype)
2. Atribut
3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut :

1. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya
3. *Public*, dapat dipanggil oleh siapa saja



Gambar 2.2 Class Diagram

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Dengan demikian *interface* mendukung resolusi metoda pada saat run-time.

Gambar 2.3 Class Diagram, Interface

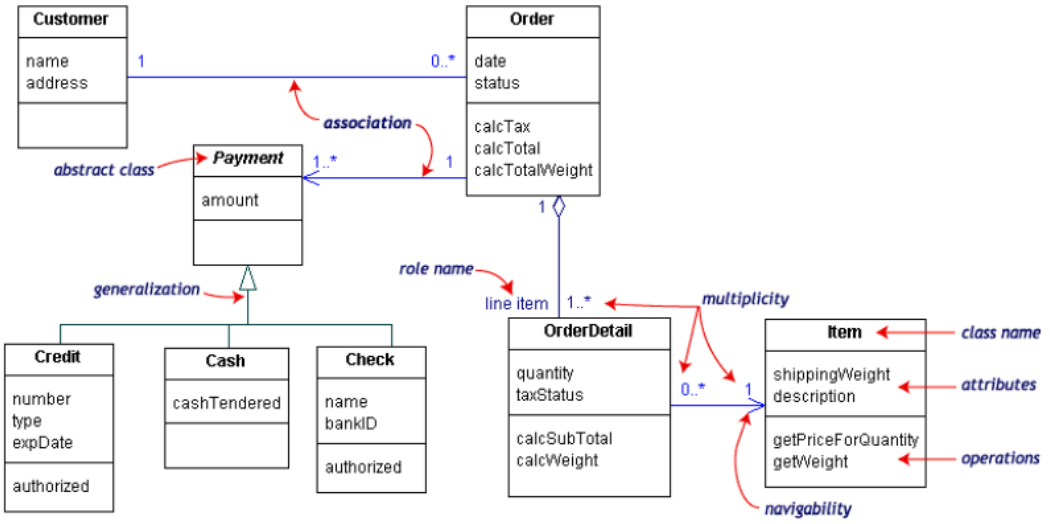
Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package*. Kita juga dapat membuat diagram yang terdiri atas *package*.

Gambar 2.4 Package

Hubungan Antar *Class*

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah navigability menunjukkan arah query antar *class*.
2. Agregasi, yaitu hubungan yang menyatakan bagian (“terdiri atas..”).
3. Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.

4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-passing dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence* diagram yang akan dijelaskan kemudian.



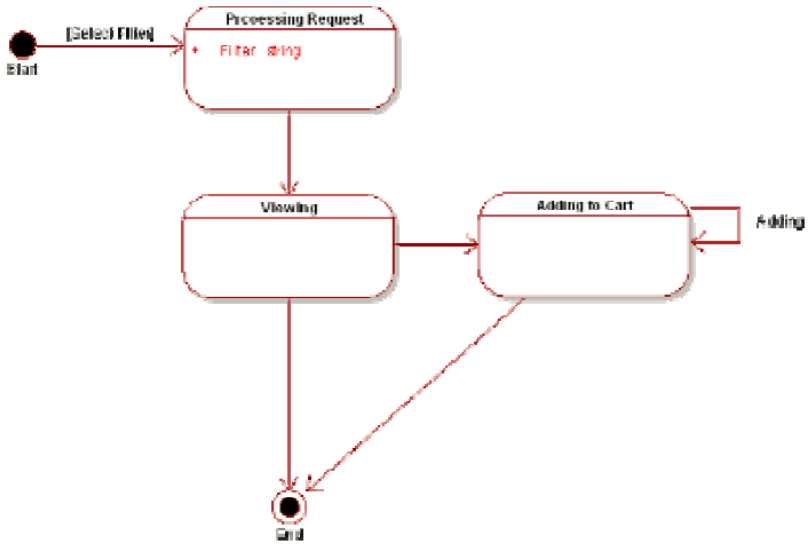
Gambar 2.5 Hubungan antar Class

2.1.3. Statechart Diagram

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu state ke state lainnya) suatu objek pada sistem sebagai akibat dari stimuli yang diterima. Pada umumnya statechart diagram menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu statechart diagram).

Dalam UML, state digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar state umumnya memiliki kondisi guard yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. Action yang dilakukan sebagai akibat dari *event* tertentu dituliskan dengan diawali garis miring.

Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.



Gambar 2.6 Statechart Diagram

2.1.4. Activity Diagram

Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, decision yang mungkin terjadi, dan bagaimana mereka berakhir. Activity diagram juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

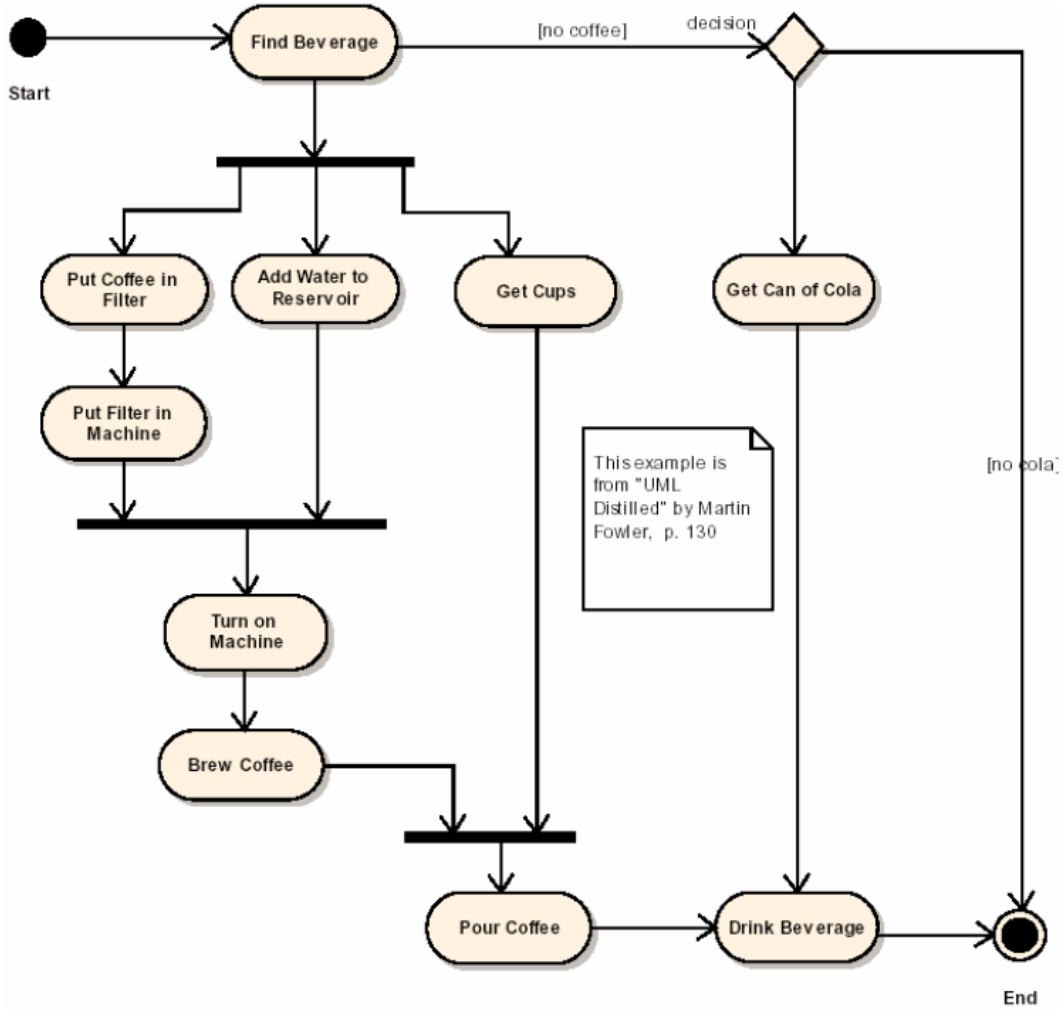
Activity diagram merupakan state diagram khusus, di mana sebagian besar state adalah action dan sebagian besar transisi di-*trigger* oleh selesainya state sebelumnya (internal processing). Oleh karena itu activity diagram tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu use case atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara use case menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti state, standar UML menggunakan segi empat dengan sudut membulat untuk menggambarkan aktivitas. Decision digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-

proses paralel (fork dan join) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal.

Activity diagram dapat dibagi menjadi beberapa object swimlane untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.



Gambar 2.7 Activity Diagram tanpa swimlane

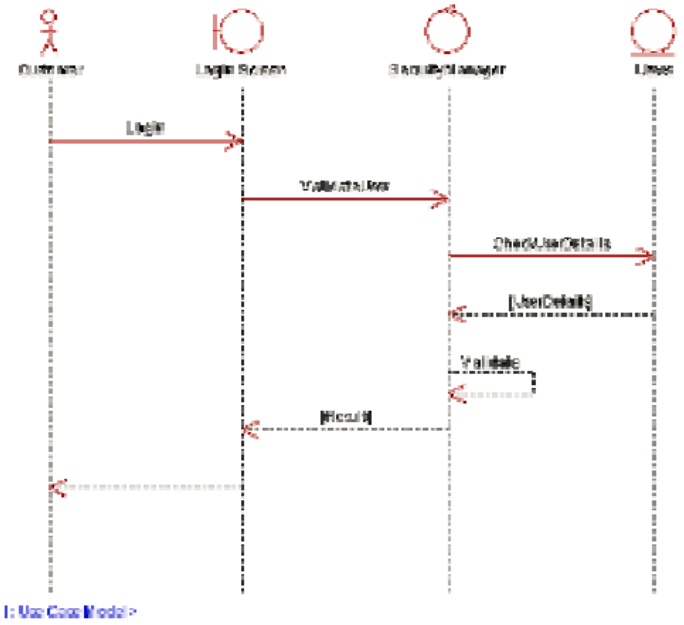
2.1.5. Sequence Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk *investor*, display, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence* diagram terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan output tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan output apa yang dihasilkan.

Masing-masing objek, termasuk aktor, memiliki lifeline vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase rancangan berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*. Activation bar menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*.

Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan icon khusus untuk objek boundary, controller dan persistent entity.



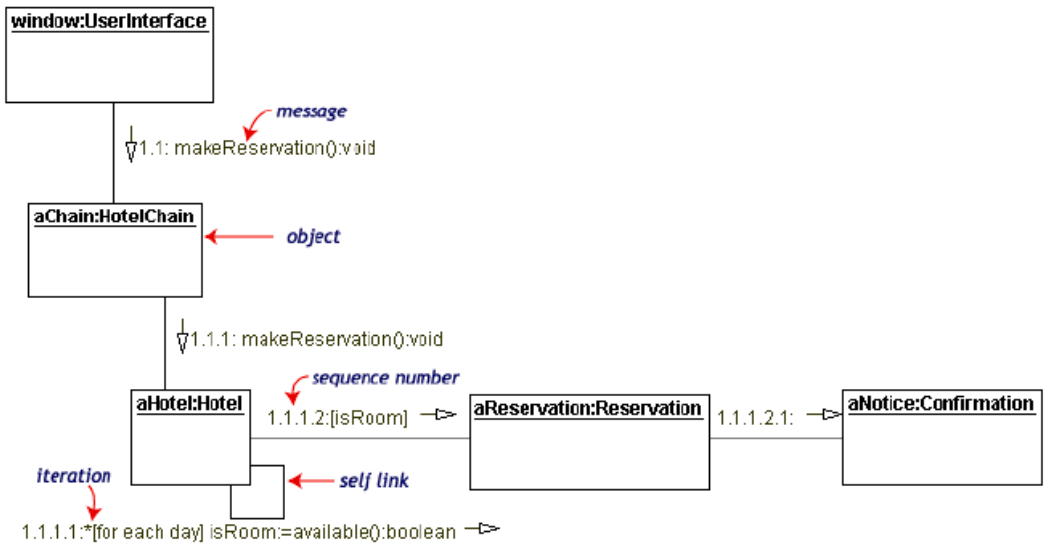
Gambar 2.8 Sequence Diagram

2.1.6. Collaboration Diagram

Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence* diagram, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*.

Setiap *message* memiliki *sequence* number, di mana *message* dari level tertinggi memiliki nomor 1.

*Message*s dari level yang sama memiliki prefiks yang sama.



Gambar 2.9 Collaboration Diagram

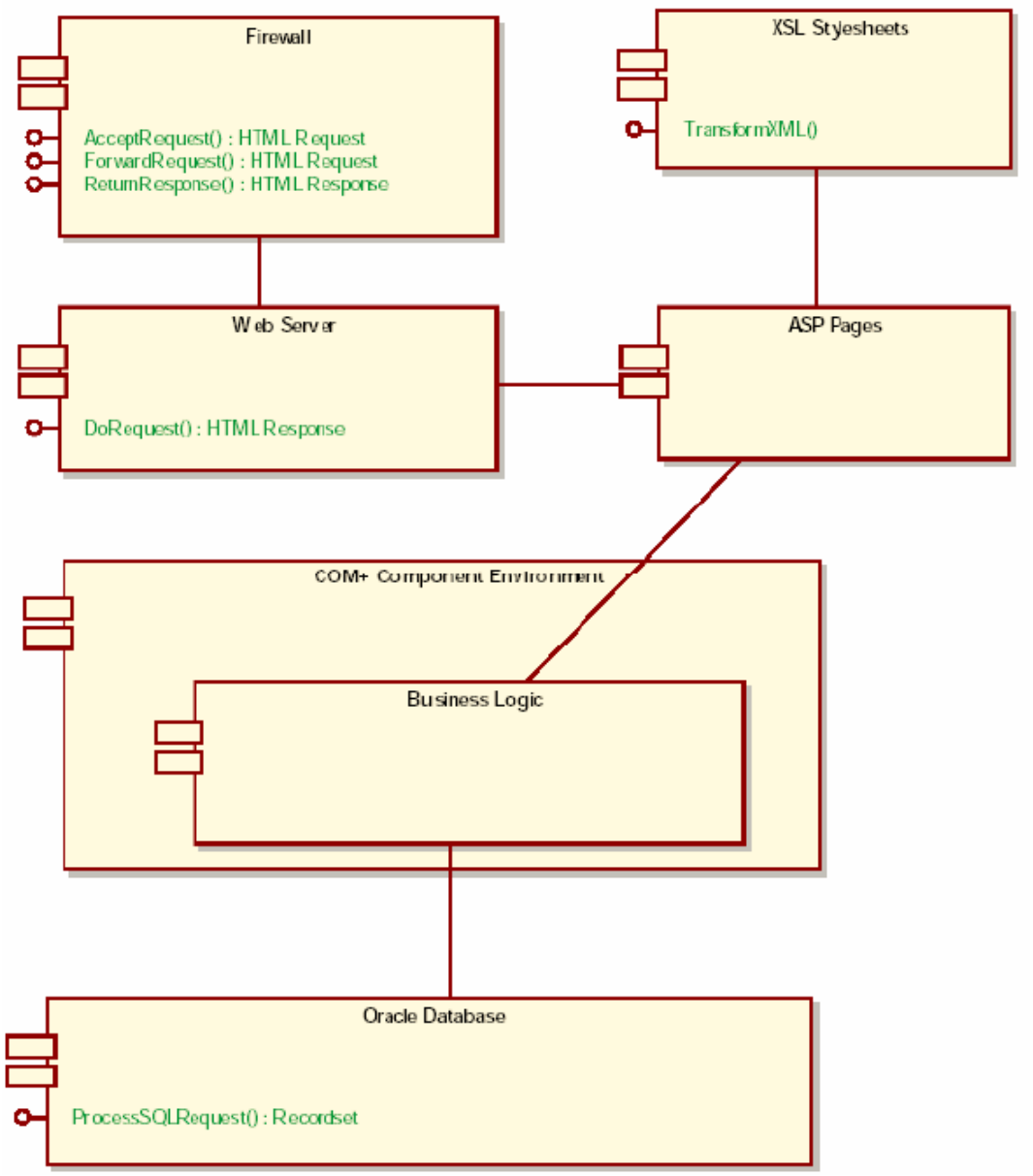
2.1.7. Component Diagram

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (dependency) di antaranya.

Komponen piranti lunak adalah modul berisi code, baik berisi source code maupun binary code, baik library maupun executable, baik yang muncul pada compile time, link time, maupun run time.

Umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil.

Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

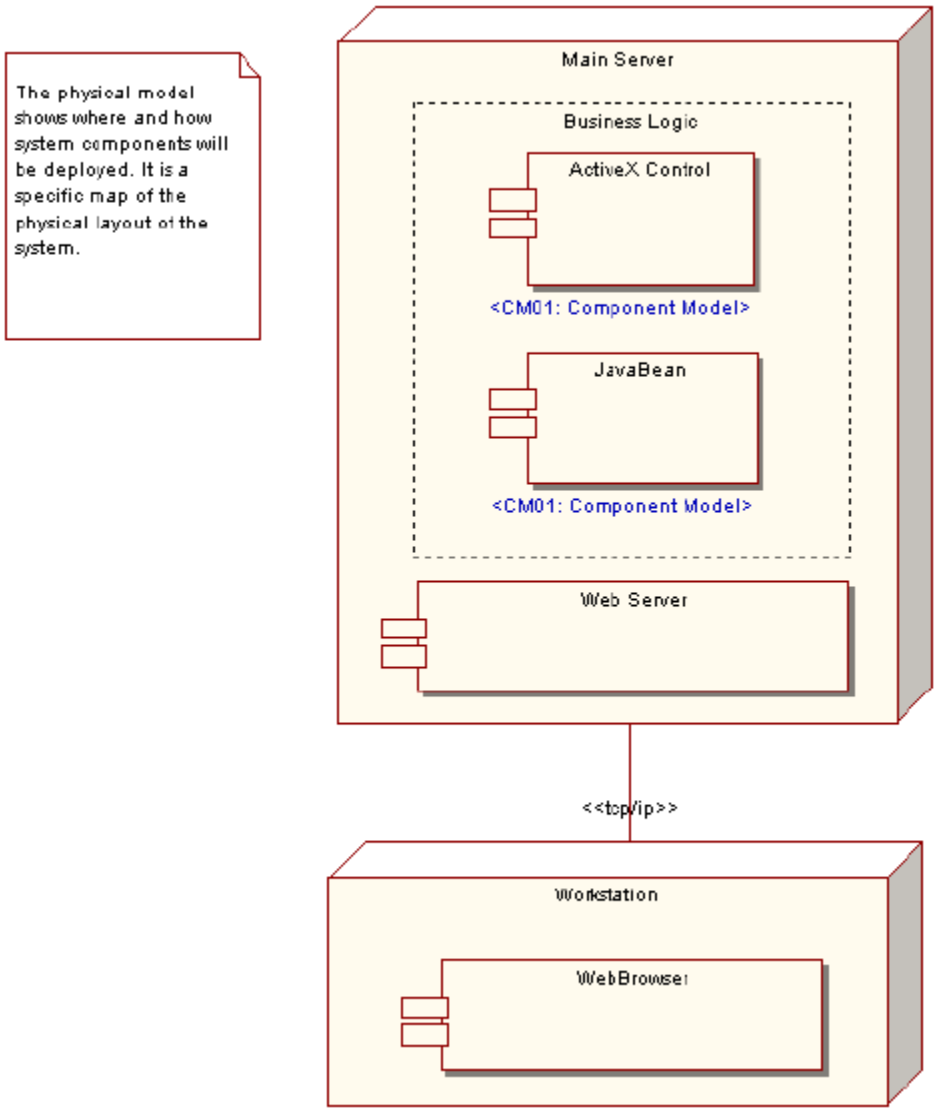


Gambar 2.10 Component Diagram

2.1.8. Deployment Diagram

Deployment/physical diagram menggambarkan detail bagaimana komponen di-deploy dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal

Sebuah node adalah server, workstation, atau piranti keras lain yang digunakan untuk men-deploy komponen dalam lingkungan sebenarnya. Hubungan antar node (misalnya TCP/IP) dan requirement dapat juga didefinisikan dalam diagram ini.



Gambar 2.11 Deployment Diagram

2.1.9. **Langkah-Langkah** Penggunaan **UML**

Berikut ini adalah tips pengembangan piranti lunak dengan menggunakan UML:

1. Buatlah daftar business process dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.
2. Petakan use case untuk tiap business process untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Kemudian perhalus use case diagram dan lengkapi dengan requirement, constraints dan catatan-catatan lain.
3. Buatlah deployment diagram secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Definisikan requirement lain (non-fungsional, security dan sebagainya) yang juga harus disediakan oleh sistem.
5. Berdasarkan use case diagram, mulailah membuat activity diagram.
6. Definisikan objek-objek level atas (*package* atau domain) dan buatlah *sequence* dan/atau collaboration diagram untuk tiap alir pekerjaan. Jika sebuah use case memiliki kemungkinan alir normal dan error, buatlah satu diagram untuk masing-masing alir.
7. Buatlah rancangan user *interface* model yang menyediakan antarmuka bagi *investor* untuk menjalankan skenario use case.
8. Berdasarkan model-model yang sudah ada, buatlah *class* diagram. Setiap *package* atau domain dipecah menjadi hirarki *class* lengkap dengan atribut dan metodanya. Akan lebih baik jika untuk setiap *class* dibuat unit test untuk menguji fungsionalitas *class* dan interaksi dengan *class* lain.
9. Setelah *class* diagram dibuat, kita dapat melihat kemungkinan pengelompokan *class* menjadi komponen-komponen. Karena itu buatlah component diagram pada tahap ini. Juga, definisikan tes integrasi untuk setiap komponen meyakinkan ia berinteraksi dengan baik.
10. Perhalus deployment diagram yang sudah dibuat. Detilkan kemampuan dan requirement piranti lunak, sistem operasi, jaringan, dan sebagainya. Petakan komponen ke dalam node.
11. Mulailah membangun sistem. Ada dua pendekatan yang dapat digunakan :

a. Pendekatan use case, dengan meng-assign setiap use case kepada tim pengembang tertentu untuk mengembangkan unit code yang lengkap dengan tes.
b. Pendekatan komponen, yaitu meng-assign setiap komponen kepada tim pengembang tertentu.

12. Lakukan uji modul dan uji integrasi serta perbaiki model berserta codenya. Model harus selalu sesuai dengan code yang aktual.
13. Piranti lunak siap dirilis.

2.2. Struktur Navigasi

Struktur navigasi adalah alur yang digunakan dalam aplikasi yang dibuat. Sebelum menyusun aplikasi multimedia kedalam sebuah software, kita harus menentukan terlebih dahulu alur apa yang akan digunakan dalam aplikasi yang dibuat.

Navigasi yang ada pada aplikasi mobile menunjukkan sesuatu yang penting dan menjadi kata kunci usabilitas aplikasi. Tersesat di dalam "sindrom hyperspace" pada navigasi searah memang harus dihindari. Oleh karena itu, penulis perlu menyampaikan suatu model mental dari struktur navigasi yang cepat dan membiarkan para pengguna untuk "menghafal peta aplikasi".

Struktur dasar logika yang jelas dan didukung oleh suatu peta aplikasi, seperti umpan balik untuk tetap pada posisi yang ada di dalam struktur ("di mana saya?"), informasi yang jelas tentang isi pada halaman yang ada ("apa yang bisa saya lakukan atau temukan di sini?"), dan item-item yang dapat dicapai pada langkah interaksi berikutnya ("kemana saya akan pergi?") adalah ramuan-ramuan paling penting untuk sebuah sistem dialog dan navigasi yang dirancang dengan baik. Bagaimanapun juga, kita harus selalu ingat bahwa mobile device menawarkan tambahan elemen navigasi "kembali" dan "*fast button*" yang mungkin dapat merusak struktur navigasi yang diharapkan atau menimbulkan ketidakefektifan.

Bentuk dasar dari struktur navigasi yang biasa digunakan dalam proses pembuatan aplikasi multimedia ada empat macam, yaitu struktur navigasi linier, hirarki, non linier dan campuran.

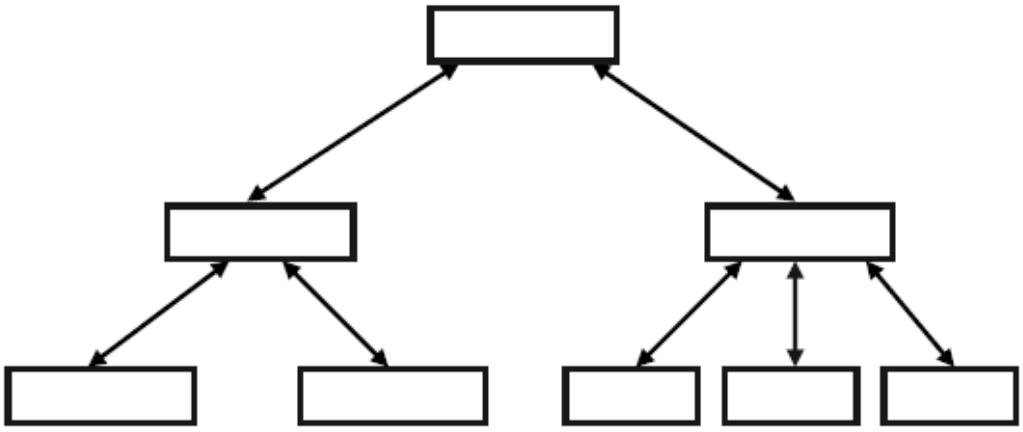
2.2.1. Struktur Navigasi Linier

Struktur navigasi linier merupakan struktur yang mempunyai satu rangkaian cerita berurutan. Struktur ini menampilkan satu demi satu tampilan layer secara berurutan menurut aturannya.

Gambar 2.12 Struktur Navigasi Linear

2.2.2. Struktur Navigasi Hirarki

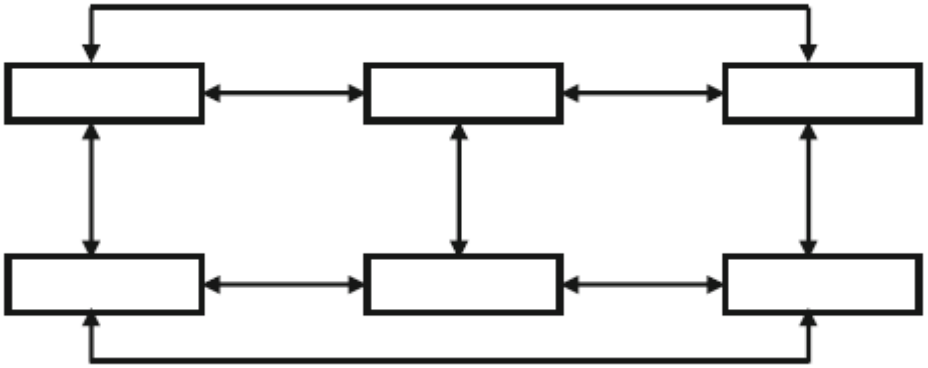
Struktur navigasi hirarki sering disebut struktur navigasi bercabang, yaitu merupakan suatu struktur yang mengandalkan percabangan untuk menampilkan data atau gambar pada layer dengan kriteria tertentu. Tampilan pada menu utama disebut master page (halaman utama satu), halaman tersebut mempunyai halaman percabangan yang disebut slave page (halaman pendukung) dan jika dipilih akan menjadi halaman kedua, begitu seterusnya.



Gambar 2.13 Struktur Navigasi Hirarki

2.2.3. Struktur Navigasi Non Linier

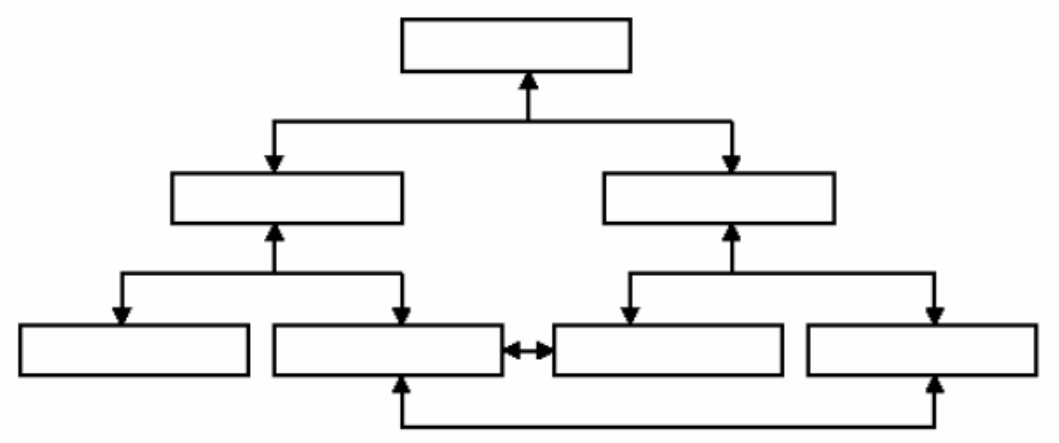
Struktur navigasi non linier (tidak terurut) merupakan pengembangan dari struktur navigasi linier, hanya saja pada struktur ini diperkenankan untuk membuat percabangan. Percabangan pada struktur non linier berbeda dengan percabangan pada struktur hirarki, pada struktur ini kedudukan semua page sama, sehingga tidak dikenal adanya master atau slave page.



Gambar 2.14 Struktur Navigasi Non Linear

2.2.4. Struktur Navigasi Campuran

Struktur navigasi campuran (composite) merupakan gabungan dari struktur sebelumnya dan disebut juga struktur navigasi bebas, maksudnya adalah jika suatu tampilan membutuhkan percabangan maka dibuat percabangan. Struktur ini paling banyak digunakan dalam pembuatan aplikasi multimedia.



Gambar 2.15 Stuktur Navigasi Campuran

3. PEMBAHASAN

3.1. Analisis Permasalahan

Aplikasi yang dirancang ini akan digunakan untuk berbagai kebutuhan yang berkaitan dengan perdagangan saham. Untuk itu aplikasi perlu menyajikan segala informasi yang penting dan relevan kepada user selain fungsi utamanya untuk memfasilitasi transaksi perdagangan saham.

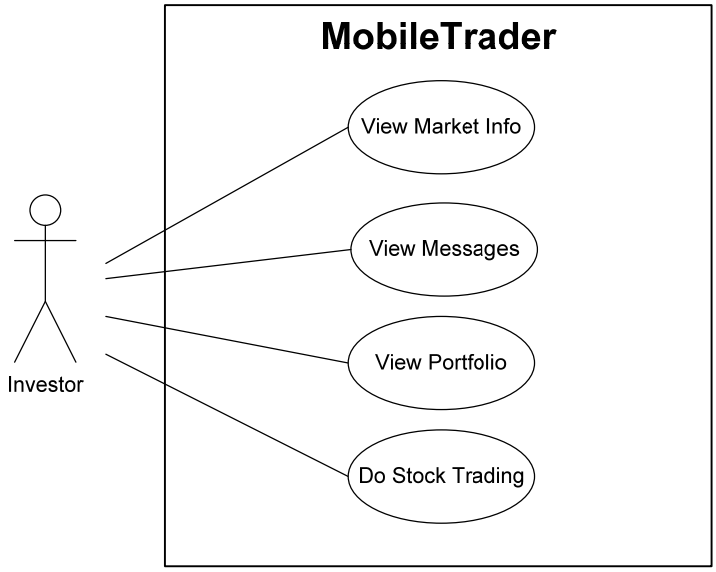
Gambaran penggunaan secara garis besar dapat dijelaskan sebagai berikut:

1. View Market Info. User akan menggunakan aplikasi untuk melihat informasi bursa, misalnya nilai terkini index harga saham gabungan (IHSG) dan index-index sektoral, transaksi saham-saham teraktif, saham-saham dengan gain terbesar atau terkecil, kalender aksi korporasi emiten, dan lain-lain.
2. View Messages. User akan menggunakan aplikasi untuk melihat pesan-pesan instan yang penting baik dari bursa seperti informasi suspensi atau buka suspensi saham atau informasi dari broker seperti status suspensi rekening, dan lain-lain.
3. View Portfolio. User akan menggunakan aplikasi untuk melihat daftar kekayaan dia di broker tempat ia terdaftar. Daftar kekayaan ini mencakup daftar saham-saham yang dimiliki beserta jumlahnya, saldo cash, dana yang tersedia untuk trading, dan lain-lain.
4. Do Stock Trading. User akan menggunakan aplikasi untuk melakukan transaksi perdagangan saham. Transaksi mencakup pengiriman order jual dan beli saham, perubahan order dan pembatalan order. User juga bisa melihat status order-ordernya dan melihat daftar order-order yang sudah match.

3.2. Perancangan Aplikasi

3.2.1. Bisnis Model

Bisnis model aplikasi *Mobile trading* digambarkan dalam Use Case Diagram berikut ini.



Gambar 3.1 Bisnis Model aplikasi *Mobile trading*

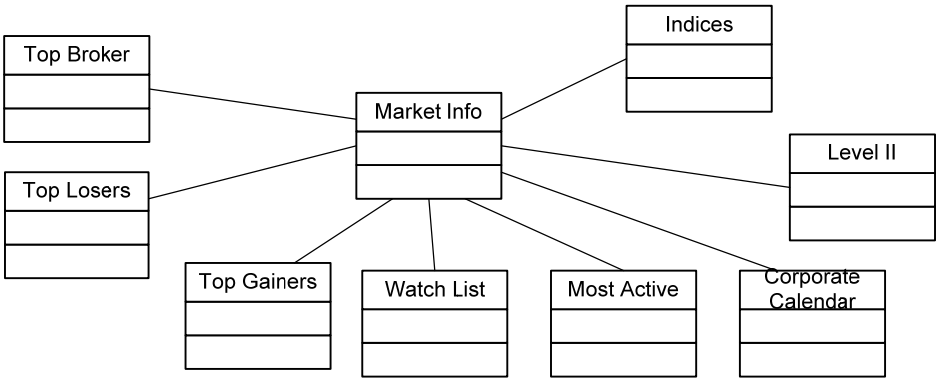
3.2.1.1. Actor

Actor yang berperan dalam aplikasi *Mobile trading* ini adalah *Investor*. *Investor* adalah nasabah perusahaan sekuritas yang tercatat memiliki rekening efek di perusahaan sekuritas tersebut. *Investor* ini tidak termasuk nasabah reksadana atau nasabah lain yang tidak memiliki rekening efek.

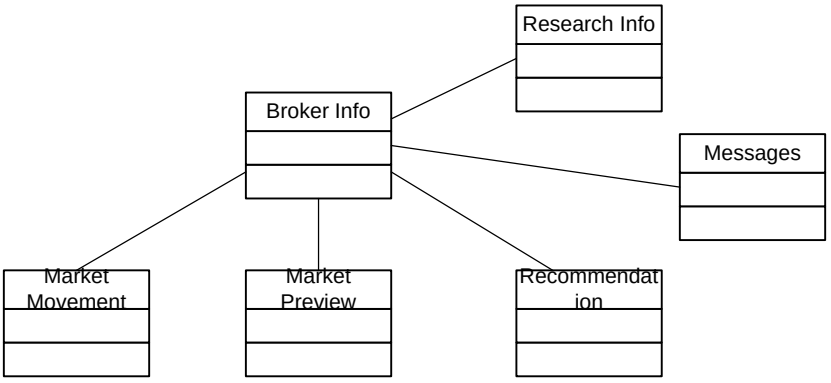
3.2.2. Static Structure

3.2.2.1. Class Diagram

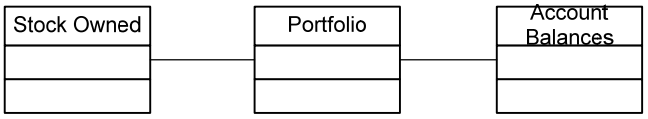
Struktur statik aplikasi digambarkan dalam *class* diagram berikut ini.



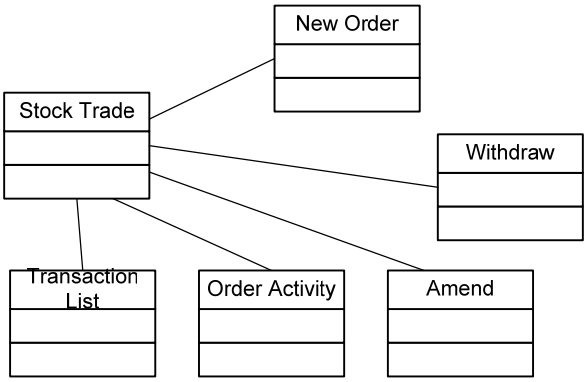
Gambar 3.2 *Class* Diagram Market Info



Gambar 3.3 *Class* Diagram Broker Info



Gambar 3.4 *Class* Diagram Portfolio



Gambar 3.5 *Class* Diagram Stock Trade

Berikut adalah penjelasan dari setiap *class* yang ada:

1. Market Info adalah *class* yang merepresentasikan informasi dari lantai bursa. Informasi yang disajikan dapat berupa data mentah atau data olahan tetapi kesemuanya merupakan data faktual bursa tanpa disertai data-data dari sumber lain. *Class* Market Info dapat dibagi menjadi beberapa kategori *class* data antara lain:
    a. Indices, adalah *class* yang merepresentasikan informasi indeks harga saham gabungan dan indeks-indeks sektoral bursa.
    b. Level II, adalah *class* yang merepresentasikan informasi antrian order dan jumlahnya berdasarkan kelompok Order Jual dan Order Beli. Order-order diurutkan berdasarkan harga, dimana Order Jual diurutkan mulai dari harga yang terkecil dan sebaliknya Order Beli diurutkan mulai dari harga yang terbesar.
    c. Corporate Calendar, adalah *class* yang merepresentasikan informasi kalender emiten menyangkut aksi korporasi yang dilakukannya, seperti RUPS, RUPSLB, Stock Dividen, Stock Split, Warrant Conversion, dan lain-lain.
    d. Most Active, adalah *class* yang merepresentasikan informasi saham-saham teraktif yang diperdagangkan di lantai bursa. Kriteria teraktifnya bisa dilihat dari frekuensi, volume, atau nilai transaksinya. Biasanya dipilih 10 sampai 20 saham teraktif.

e. Watch List, adalah *class* yang merepresentasikan informasi saham-saham pilihan yang dipantau oleh *investor*. Informasi yang ditampilkan biasanya hanya informasi yang *volatile* dan *current* seperti last price, change, change percent, best-bid, best-offer, best-bid volume dan best-offer volume. Sedangkan informasi-informasi yang jarang berubah seperti listed-shares tidak diakomodasi di *class* ini.

f. Top Gainers, adalah *class* yang merepresentasikan informasi saham-saham yang mengalami **kenaikan** harga terbesar antara harga terakhir (last price) dengan harga penutupan hari bursa sebelumnya (previous price). Selisih ini bisa diukur dari besarnya secara nilai maupun secara persentase. Biasanya dipilih 10 sampai 20 saham top gainers.

g. Top Losers, adalah *class* yang merepresentasikan informasi saham-saham yang mengalami **penurunan** harga terbesar antara harga terakhir (last price) dengan harga penutupan hari bursa sebelumnya (previous price). Selisih ini bisa diukur dari besarnya secara nilai maupun secara persentase. Biasanya dipilih 10 sampai 20 saham top losers.

h. Top Brokers, adalah *class* yang merepresentasikan informasi broker-broker teraktif yang melakukan perdagangan di lantai bursa. Kriteria teraktifnya bisa diukur dari frekuensi, volume, atau nilai transaksi yang dilakukannya. Biasanya dipilih 10 sampai 20 broker teraktif.

2. Broker Info, adalah *class* yang merepresentasikan informasi dari broker dimana si *investor* tercatat memiliki rekening. *Class* ini menyajikan informasi yang berkaitan dengan berita aktifitas bursa, aktifitas broker, rekomendasi dari tim riset, berita terkini dan pesan-pesan lain dari broker tersebut yang relevan dengan kebutuhan *investor*. *Class* Broker Info terdiri dari beberapa *class* yaitu:

a. Research Info, adalah *class* yang merepresentasikan informasi dari tim riset di broker. Research Info biasanya menyajikan informasi terkini tentang emiten yang dapat mempengaruhi trend pergerakan harga saham di bursa. Meskipun tidak menutup kemungkinan tim riset menyajikan informasi umum lainnya yang relevan seperti kondisi ekonomi makro, kebijakan pemerintah, pasar luar negeri dan lain-lain.
b. *Message*s, adalah *class* yang merepresentasikan informasi dari broker untuk si *investor*. *Class* ini mengakomodasi pesan-pesan yang ingin disampaikan oleh broker—biasanya dari sales, trader, atau tim riset—untuk si *investor* tersebut.
c. Recommendation, adalah *class* yang merepresentasikan informasi rekomendasi dari broker—biasanya dari tim riset—untuk saham-saham tertentu yang dipilih broker. Rekomendasi itu meliputi rekomendasi untuk melakukan aksi buy, sell, buy on weakness, sell on strength, hold, dan lain-lain.
d. Market Preview, adalah *class* yang merepresentasikan prediksi tentang aktifitas yang akan terjadi lantai bursa. Informasi ini disajikan sebelum market dibuka dan biasanya menyangkut prakiraan pergerakan indeks dan harga-harga saham tertentu secara umum serta aspek-aspek yang akan mempengaruhinya seperti trend bursa luar negeri, nilai tukar, suku bunga, harga komoditas, dan lain-lain.
e. Market movement, adalah *class* yang merepresentasikan informasi tentang aktifitas yang telah atau sedang terjadi di lantai bursa. Informasi ini disajikan pada saat akhir sesi I atau setelah market tutup dan biasanya menyangkut pergerakan indeks dan harga-harga saham tertentu secara umum serta aspek-aspek yang mempengaruhinya.

3. Portfolio, adalah *class* yang merepresentasikan informasi tentang kekayaan si *investor* di broker dimana ia terdaftar. Kekayaan itu

mencakup saham-saham yang dimiliki dan saldo rekening. Untuk lebih spesifiknya *class* ini dibagi menjadi dua sub-*class* yaitu:

a. Stock Owned, adalah *class* yang merepresentasikan daftar saham-saham yang dimiliki oleh si *investor* beserta jumlahnya.
b. Account Balances, adalah *class* yang merepresentasikan posisi cash, dana yang tersedia untuk trading (order power), dana yang tersedia untuk ditarik, dan nilai rekening. Order Power adalah *class* yang merepresentasikan daya beli si *investor* yang dihitung berdasarkan posisi cash dan saham-saham yang dia miliki.

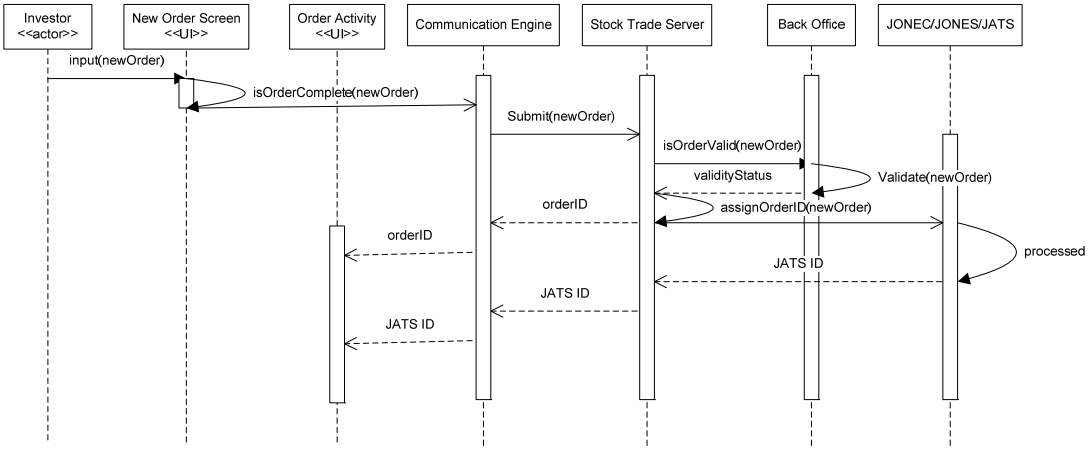
4. Stock Trade, adalah *class* yang merepresentasikan fungsi untuk melakukan perdagangan saham. *Class* ini terdiri dari beberapa *class* dengan fungsi yang lebih spesifik, yaitu:
   a. New Order, adalah *class* yang merepresentasikan fungsi untuk melakukan pengiriman order baru.
   b. Withdraw, adalah *class* yang merepresentasikan fungsi untuk melakukan pembatalan order yang sudah dikirimkan.
   c. Amend, adalah *class* yang merepresentasikan fungsi untuk melakukan perubahan pada order yang sudah dikirimkan.
   d. Order Activity, adalah *class* yang merepresentasikan informasi tentang status order-order yang sudah dikirimkan.
   e. Transaction List adalah *class* yang merepresentasikan informasi tentang order-order yang sudah match.

3.2.3. Dynamic Behaviour

Aspek dinamik dari aplikasi *Mobile trading* dapat digambarkan dalam diagram-diagram berikut ini.

3.2.3.1. Sequence Diagram

Proses pengiriman New Order dapat digambarkan dalam *sequence* diagram berikut.



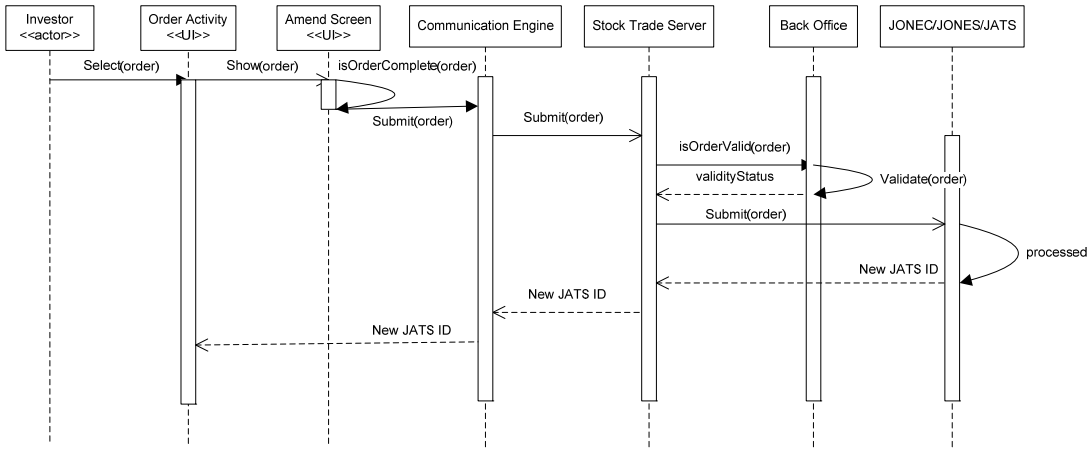
Gambar 3.6 New Order *sequence* diagram

Prosesnya dapat dijelaskan sebagai berikut:

1. *Investor* memasukan order baru (newOrder) melalui New Order Screen
2. New Order Screen mengecek kelengkapan data yang dimasukan *investor*. Jika data tidak lengkap maka New Order Screen akan memunculkan dialog bahwa data newOrder tidak lengkap sehingga user bisa melengkapi data yang belum dimasukan.
3. Jika data sudah lengkap maka New Order Screen akan mengirimkan newOrder ke Communication Engine.
4. Communication Engine akan meneruskan newOrder ke StockTrade Server.
5. StockTrade server akan memvalidasi newOrder dengan cara mengeceknya melalui Back Office.
6. Jika newOrder tidak valid—misalnya karena dana untuk trading tidak mencukupi—maka newOrder akan ditolak dan dikirimkan status “Rejected” ke Stock Trade Server yang kemudian akan diteruskan sampai ke Order Activity screen.
7. Jika newOrder valid maka newOrder akan dikirimkan ke JONEC/JONES/JATS.

8. Setelah diproses maka JONEC/JONES/JATS akan mengembalikan order status dan JATS ID ke StockTrade server yang kemudian akan meneruskannya sampai ke Order Activity screen.

Proses perubahan order yang sudah dikirimkan (amendment) dapat digambarkan dalam *sequence* diagram berikut ini.



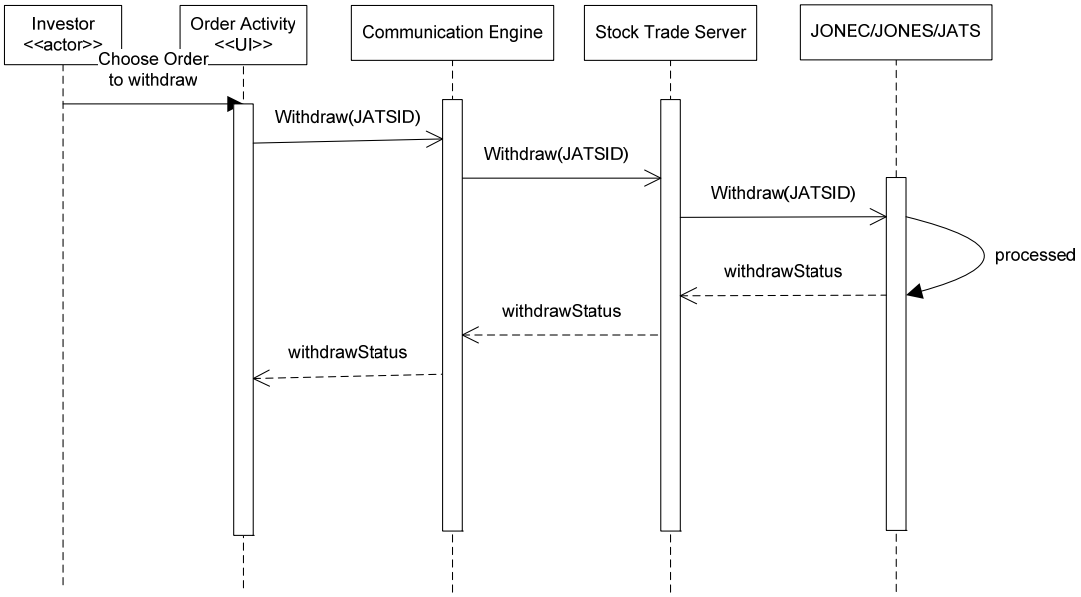
Gambar 3.7 Amend Order *sequence* diagram

Prosesnya dapat dijelaskan sebagai berikut:

1. Dari screen Order Activity, *Investor* menentukan order yang akan di-amend
2. *Investor* memilih menu *Amend* dari popup menu untuk order yang dipilih tersebut sehingga memunculkan Amend Screen.
3. Dari Amend Screen data-data baru akan dikirimkan ke Communication Engine.
4. Communication Engine akan meneruskannya ke StockTrade Server.
5. StockTrade Server akan memvalidasi order melalui Back Office.
6. Setelah mendapat status valid dari Back Office StockTrade Server meneruskannya ke JONEC/JONES/JATS.
7. Setelah diproses, JONEC/JONES/JATS mengembalikan status *amended* ke StockTrade Server.

8. StockTrade Server meneruskan status tersebut ke Communicatin Engine dan akhirnya sampai di Order Activity screen.

Proses pembatalan order yang sudah dikirimkan (withdraw) dapat digambarkan dalam *sequence* diagram berikut ini.



Gambar 3.8 Withdraw Order *sequence* diagram

Dalam proses withdraw order ini prosesnya lebih sederhana dari pengubahan order dimana tidak perlu lagi dilakukan verifikasi dan validasi oleh back-office. Prosesnya dapat dijelaskan sebagai berikut:

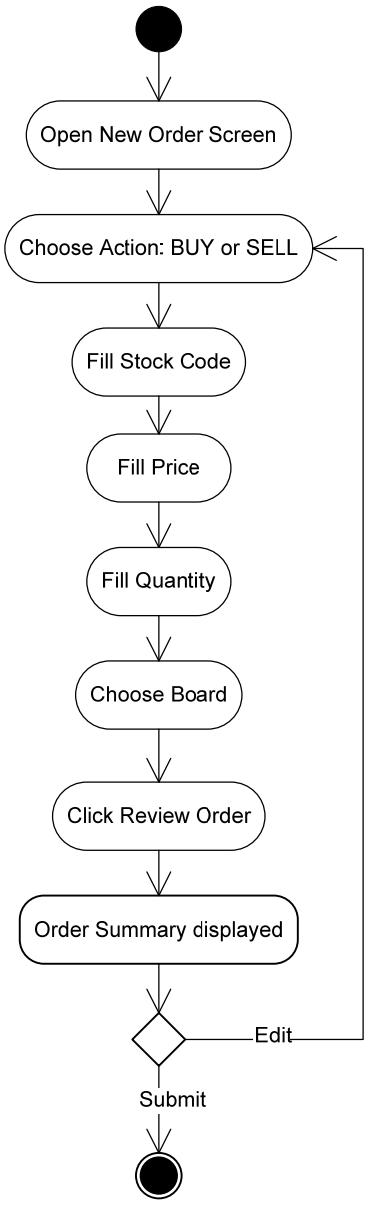
1. Dari Order Activity screen, *Investor* menentukan order yang akan di-withdraw.
2. Perintah withdraw dilakukan dengan memilih menu *Withdraw* dari popup menu dan melakukan konfirmasi dari dialog yang dimunculkan.
3. Perintah withdraw akan dikirimkan oleh OrderActivity screen ke Communication Engine.
4. Communication Engine meneruskannya ke StockTrade Server.
5. StockTrade Server meneruskannya ke JONEC/JONES/JATS.

6. Setelah diproses, JONEC/JONES/JATS mengembalikan status *withdrawn* ke StockTrade Server.
7. StockTrade Server meneruskan status withdraw ke Communicatin Engine dan akhirnya sampai di Order Activity screen.

3.2.3.2. Activity Diagram

Bentuk aktivitas yang dilakukan *investor* terhadap aplikasi *Mobile trading* dapat digambarkan dalam beberapa activity diagram berikut ini.

3.2.3.2.1. New Order



Gambar 3.9 Activity Diagram pembuatan New Order

Basic Flow

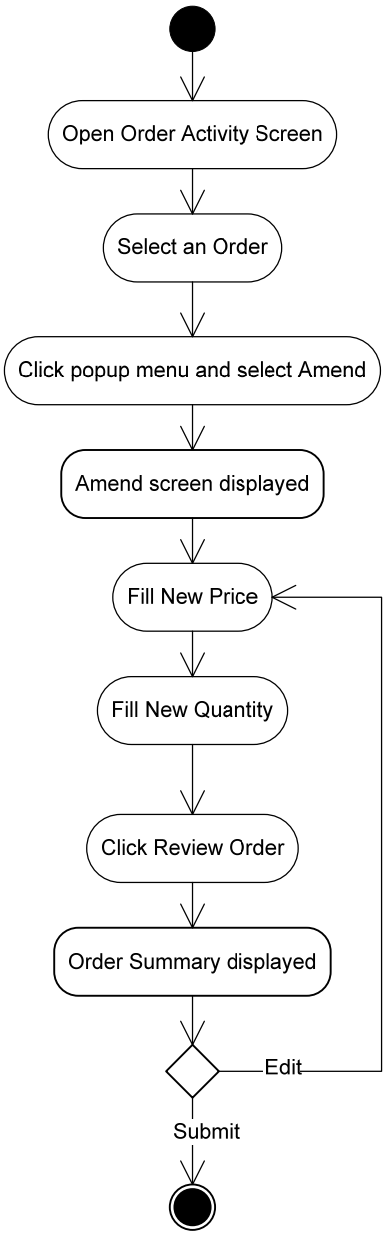
1. *Investor* membuka New Order screen
2. *Investor* memilih aksi: BUY atau SELL
3. *Investor* mengisi stock code
4. *Investor* mengisi price
5. *Investor* mengisi quantity

6. *Investor* memilih board
7. *Investor* meng-klik Review Order
8. Order Summary dialog ditampilkan.
9. *Investor* meng-klik submit

Alternate Flow

1. Pada basic flow point 8 jika *Investor* meng-klik edit maka Order Summary dialog akan ditutup dan New Order screen akan ditampilkan kembali. *Investor* bisa kembali ke basic flow point 3.
2. Pada semua point basic flow, *Investor* bisa meng-klik tombol ESCAPE untuk membatalkan proses dan menutup screen New Order.

3.2.3.2.2. Amend Order



Gambar 3.10 Activity Diagram perubahan order (amendment)

Basic Flow

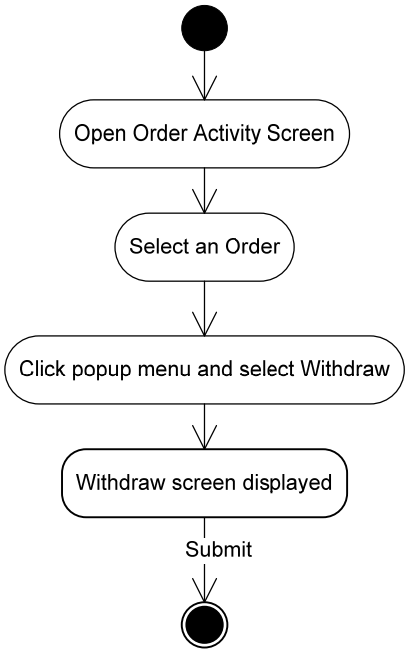
1. *Investor* membuka Order Activity screen
2. *Investor* memilih sebuah order dari daftar order yang ditampilkan.
3. *Investor* membuka menu pop-up dan memilih menu Amend
4. Amend screen ditampilkan

5. *Investor* mengisi New Price
6. *Investor* mengisi New Quantity
7. *Investor* mengklik tombol Review Order
8. Order Summary dialog ditampilkan
9. *Investor* meng-klik tombol Submit

Alternate Flow

1. Pada basic flow point 8 jika *Investor* meng-klik edit maka Order Summary dialog akan ditutup dan Amend Order screen akan ditampilkan kembali. *Investor* bisa kembali ke basic flow point 5.
2. Pada semua point basic flow, *investor* bisa meng-klik tombol ESCAPE untuk membatalkan proses dan menutup screen Amend Order

3.2.3.2.3. Withdraw Order



Gambar 3.11 Activity Diagram pembatalan order (withdrawal)

Basic Flow

1. *Investor* membuka Order Activity screen
2. *Investor* memilih sebuah order dari daftar order yang ditampilkan.
3. *Investor* membuka menu pop-up dan memilih menu Withdraw

4. Withdraw screen ditampilkan
5. *Investor* mengklik tombol Submit

Alternate Flow

1. Pada semua point basic flow, *investor* bisa meng-klik tombol ESCAPE untuk membatalkan proses dan menutup screen Withdraw Order

3.2.4. Data Structure

Struktur data aplikasi *Mobile trading* direpresentasikan dalam kelas-kelas data berikut ini.

1. *Class* **Indices**

Attributes:

- *Items: IndicesItem[]*, daftar item indeks-indeks.

Methods:

- *getItems():IndicesItem[]*, mengembalikan nilai yang dikandung attribute *items*.

2. *Class* **IndicesItem**

Attributes:

a. *code:String*, kode indeks
b. *lastValue:float*, nilai indeks terkini
c. *previousValue:float*, nilai indeks hari bursa sebelumnya
d. *change:float*, perubahan nilai indeks dihitung dari nilai atribut: *lastValue – previousValue*
e. *changePct:float*, perubahan nilai indeks dalam persen dihitung dari nilai atribut: (*change / previousValue*)*100%

Methods:

f. *getCode():String*, mengembalikan nilai yang dikandung atribut *code*
g. *getLastValue():float*, mengembalikan nilai yang dikandung atribut *lastValue*
h. *getPreviousValue():float*, mengembalikan nilai yang dikandung atribut *previousValue*

i. *getChange():float*, mengembalikan nilai yang dikandung atribut *change*
j. *getChangePct():float*, mengembalikan nilai yang dikandung atribut *changePct*

3. *Class* **MostActive**

Attributes:

a. *items: MostActiveItem[]*, daftar item saham-saham teraktif

Methods:

b. *getItems():MostActiveItem[]*, mengembalikan nilai yang dikandung attribute *items*.

4. *Class* **MostActiveItem**

Attributes:

a. *code:String, kode saham*
b. *volume:long, total volume transaksi (dalam lot)*
c. *value:double, total nilai transaksi (dalam rupiah)*
d. *frequency:long, total jumlah/cacah transaksi*

Methods:

e. *getCode():String*, mengembalikan nilai yang dikandung atribut *code*
f. *getVolume():long*, mengembalikan nilai yang dikandung atribut *volume*
g. *getValue():double*, mengembalikan nilai yang dikandung atribut *value*
h. *getFrequency():long*, mengembalikan nilai yang dikandung atribut *frequency*

5. *Class* **Level2**

Attributes:

a. *code:string*, kode saham
b. *bidQueue*:OrderQueueItem[], daftar antrian beli
c. *offerQueue*:OrderQueueItem[], daftar antrian jual
d. *tradeSummary*:TradeSummaryItem[], daftar ringkasan transaksi

e. *bidVol:long*, total volume antrian beli
f. *offerVol:long*, total volume antrian jual

Methods:

g. *getCode():String*, mengembalikan nilai yang dikandung attribute *code*.
h. *getBidQueue():OrderQueueItem[]*, mengembalikan nilai yang dikandung attribute *bidQueue*.
i. *getOfferQueue():OrderQueueItem[]*, mengembalikan nilai yang dikandung attribute *offerQueue*.
j. *getTradeSummary():TradeSummaryItem[]*, mengembalikan nilai yang dikandung attribute *tradeSummary*.
k. *getBidVol():long*, mengembalikan nilai yang dikandung attribute *bidVol*.
l. *getOfferVol():long*, mengembalikan nilai yang dikandung attribute *offerVol*.

6. *Class* **OrderQueueItem**

Attributes:

a. *price:float*, harga dalam antrian
b. *volume:long*, volume dalam antrian pada harga tersebut

Methods:

c. *getPrice():float*, mengembalikan nilai yang dikandung attribute *price*.
d. *getVolume():long*, mengembalikan nilai yang dikandung attribute *volume*.

7. *Class* **TradeSummaryItem**

Attributes:

a. *price:float*, harga transaksi
b. *volume:long*, volume transaksi pada harga tersebut
c. *frequency:long*, frekuensi transaksi pada harga tersebut

Methods:

d. *getPrice():float*, mengembalikan nilai yang dikandung attribute *price*.
e. *getVolume():long*, mengembalikan nilai yang dikandung attribute *volume*.
f. *getFrequency():long*, mengembalikan nilai yang dikandung attribute *frequency*.

8. *Class* **SimpleStockSummaryItem**

Attributes:

a. *code:String*, kode saham
b. *lastPrice:float*, harga terkini di bursa
c. *previousPrice:float*, harga penutupan hari bursa sebelumnya
d. *change:float*, perubahan harga yang dihitung dari nilai atribut: *lastPrice – previousPrice*
e. *changePct:float*, perubahan harga dalam persen yang dihitung dari nilai atribut: (*change / previousValue*)**100%*

Methods:

f. *getCode():String*, mengembalikan nilai yang dikandung attribute *code*.
g. *getLastPrice():float*, mengembalikan nilai yang dikandung attribute *lastPrice*.
h. *getPreviousPrice():float*, mengembalikan nilai yang dikandung attribute *previousPrice*.
i. *getChange():float*, mengembalikan nilai yang dikandung attribute *change*.
j. *getChangePct():float*, mengembalikan nilai yang dikandung attribute *changePct*.

9. *Class* **TopGainers**

Attributes:

a. *items:SimpleStockSummaryItem[]*, daftar saham-saham dengan kenaikan harga terbesar

Methods:

b. *getItems():SimpleStockSummaryItem[]*, mengembalikan nilai yang dikandung attribute *items*.

10. *Class* **TopLosers**

Attributes:

a. *items:SimpleStockSummaryItem[]*, daftar saham-saham dengan penurunan harga terbesar

Methods:

b. *getItems():SimpleStockSummaryItem[]*, mengembalikan nilai yang dikandung attribute *items*.

11. *Class* **WatchList**

Attributes:

a. *items:Vector<SimpleStockSummaryItem>*, daftar saham-saham yang dipantau user; dibuat dalam bentuk vektor sehingga daftarnya bersifat dinamis, bisa ditambah dan dikurangi.

Methods:

b. *getItems():Vector<SimpleStockSummaryItem>*, mengembalikan nilai yang dikandung attribute *items*.

12. *Class* **BrokerSummaryItem**

Attributes:

a. *code:string*, kode broker
b. *volume:long*, total volume transaksi (dalam lot)
c. *value:double*, total nilai transaksi (dalam rupiah)
d. *frequency:long*, total frekuensi transaksi

Methods:

e. *getCode():String*, mengembalikan nilai yang dikandung attribute *code*
f. *getVolume():long*, mengembalikan nilai yang dikandung attribute *volume*
g. *getValue():double*, mengembalikan nilai yang dikandung attribute *value*

h. *getFrequency():long*, mengembalikan nilai yang dikandung attribute *frequency*

13. *Class* **TopBroker**

Attributes:

a. *items:BrokerSummaryItem[]*, daftar broker-broker teraktif

Methods:

b. *getItems():BrokerSummaryItem[]*, mengembalikan nilai yang dikandung attribute *items*.

14. *Class* CorpCalendar

Attributes:

a. *code:String*, kode emiten
b. *action:String*, kode aksi korporasi
c. *date:Date*, tanggal aksi korporasi
d. *details:CorpCalendarDetails[]*, detail aksi korporasi

Methods:

e. *getCode():String*, mengembalikan nilai yang dikandung attribute *code*.
f. *getAction():String*, mengembalikan nilai yang dikandung attribute *action*.
g. *getDate():Date*, mengembalikan nilai yang dikandung attribute *date*.
h. *getDetails():CorpCalendarDetails[]*, mengembalikan nilai yang dikandung attribute *details*.

15. *Class* **CorpCalendarDetails**

Attributes:

a. *tag:String*, kode info
b. *value:String*, nilai info

Methods:

c. *getTag():String*, mengembalikan nilai yang dikandung attribute *tag*.

d. *getValue():String*, mengembalikan nilai yang dikandung attribute *value*.

16. *Class* **MarketMovement**

Attributes:

a. *date:Date*, tanggal data
b. *title:String*, judul ulasan
c. content:String, isi ulasan
d. *source:String*, sumber ulasan

Methods:

e. *getDate():Date*, mengembalikan nilai yang dikandung atribut *date*.
f. *getTitle():String*, mengembalikan nilai yang dikandung atribut *title.*
g. *getContent():String*, mengembalikan nilai yang dikandung atribut *content.*
h. *getSource():String*, mengembalikan nilai yang dikandung atribut *source.*

17. *Class* **MarketPreview**

Attributes:

a. *date:Date*, tanggal data
b. *title:String*, judul ulasan
c. *content:String*, isi ulasan
d. *source:String*, sumber ulasan

Methods:

e. *getDate():Date*, mengembalikan nilai yang dikandung atribut *date*.
f. *getTitle():String*, mengembalikan nilai yang dikandung atribut *title*
g. *getContent():String*, mengembalikan nilai yang dikandung atribut *content.*

h. *getSource():String*, mengembalikan nilai yang dikandung atribut *source.*

18. *Class* **RecommendationItem**

Attributes:

a. *code:String*, kode saham
b. *recom:String*, rekomendasi: “Buy”, “Sell”, “Buy On Weakness”, “Sell On Strength”, “Hold”, “No Recommendation”.
c. *remarks:String*, catatan dari tim riset

Methods:

d. *getCode():String*, mengembalikan nilai yang dikandung atribut *code*.
e. *getRecom():String*, mengembalikan nilai yang dikandung atribut *recom*.
f. *getRemarks():String*, mengembalikan nilai yang dikandung atribut *remarks.*

19. *Class* **Recommendation**

Attributes:

a. *date:Date*, tanggal rekomendasi
b. *items:RecommendationItem[]*, daftar item rekomendasi.

Methods:

c. *getDate():Date*, mengembalikan nilai yang dikandung atribut *date*.
d. *getItems():RecommendationItem[]*, mengembalikan nilai yang dikandung atribut *items*.

20. *Class* ***Message*Item**

Attributes:

a. *date:Date*, tanggal data
b. *title:String*, judul ulasan
c. *content:String*, isi ulasan
d. *source:String*, sumber ulasan

Methods:

e. *getDate():Date*, mengembalikan nilai yang dikandung atribut *date*.
f. *getTitle():String*, mengembalikan nilai yang dikandung atribut *title*
g. *getContent():String*, mengembalikan nilai yang dikandung atribut *content.*
h. *getSource():String*, mengembalikan nilai yang dikandung atribut *source.*

21. *Class* ***Message*s**

Attributes:

a. *date:Date*, tanggal rekomendasi
b. *items:MessageItem[]*, daftar item rekomendasi.

Methods:

c. *getDate():Date*, mengembalikan nilai yang dikandung *date*.
d. *getItems():RecommendationItem[]*, mengembalikan nilai yang dikandung *items*.

22. *Class* **StockOwned**

Attributes:

a. *investorID: String*, id *investor*
b. *dateTime: Date*, tanggal dan waktu data
c. *items:Vector<StockOwnedItem>*, daftar saham yang dimiliki *investor*

Methods:

d. *getInvestorID():String*, mengembalikan nilai yang dikandung atribut *investorID.*
e. *getDateTime():Date*, mengembalikan nilai yang dikandung *dateTime*.
f. *getItems: Vector<StockOwnedItem>*, mengembalikan nilai yang dikandung *items*.

23. *Class* **StockOwnedItem**

Attributes:

a. *code:string*, kode saham
b. *quantity:long*, jumlah saham
c. *avgPrice:float*, harga rata-rata perolehan

Methods:

d. *getCode():String*, mengembalikan nilai yang dikandung atribut *code*.
e. *getQuantity():long*, mengembalikan nilai yang dikandung atribut *quantity*.
f. *getAvgPrice():String*, mengembalikan nilai yang dikandung atribut *avgPrice.*

24. *Class* **AccountBalances**

Attributes:

a. *investorID: String*, id *investor*
b. *dateTime: Date*, tanggal dan waktu data
c. *cash:double*, posisi saldo cash
d. *availForTrading:double*, posisi dana yang tersedia untuk trading
e. *availForWithdrawal:double*, posisi dana yang tersedia untuk penarikan
f. *accValue:double*, posisi nilai rekening yang dihitung dari cash dan saham yang dimiliki

Methods:

g. *getInvestorID(): String*, mengembalikan nilai yang dikandung atribut *investorID*
h. *getDateTime(): Date*, mengembalikan nilai yang dikandung atribut *dateTime*
i. *getCash():double*, mengembalikan nilai yang dikandung atribut *cash*
j. *getAvailForTrading():double*, mengembalikan nilai yang dikandung atribut *availForTrading*
k. *getAvailForWithdrawal():double*, mengembalikan nilai yang dikandung atribut availForWithdrawal

l. *getAccValue():double*, mengembalikan nilai yang dikandung atribut *accValue*

25. *Class* **Order**

Attributes:

a. *dateTime:Date*, tanggal dan waktu data
b. *orderID:string*, nomor order dari server broker
c. *jatsID:string*, nomor order dari server JATS
d. code:string, kode saham
e. *side:Side*, posisi aksi: BUY atau SELL
f. *status:OrderStatus*, order status: OPEN, FULL_MATCH, PARTIAL_MATCH, AMENDED, WITHDRAWN
g. *price:float*, harga transaksi
h. *quantity:long*, jumlah saham yang ditransaksikan
i. *board:Board*, papan transaksi: RG (regular), TN (pasar tunai)
j. *matchQty:long*, jumlah saham yang sudah match.
k. *remainQty:long*, jumlah saham sisa yang belum match

Methods:

l. *getDateTime():Date*, mengembalikan nilai yang dikandung atribut *dateTime.*
m. *getOrderID():String*, mengembalikan nilai yang dikandung atribut *orderID.*
n. *getJatsID():String*, mengembalikan nilai yang dikandung atribut *jatsID.*
o. *getCode():String*, mengembalikan nilai yang dikandung atribut *code.*
p. *getSide():Side,* mengembalikan nilai yang dikandung atribut *side.*
q. *getStatus():OrderStatus*, mengembalikan nilai yang dikandung atribut *status.*
r. *getPrice():float*, mengembalikan nilai yang dikandung atribut *price.*

s. *getQuantity():long*, mengembalikan nilai yang dikandung atribut *quanitity.*
t. *getBoard():Board*, mengembalikan nilai yang dikandung atribut *board.*
u. *getMatchQty():long*, mengembalikan nilai yang dikandung atribut *matchQty*.
v. *getRemainQty():long*, mengembalikan nilai yang dikandung atribut *remainQty.*

26. *Class* **OrderActivity**

Attributes:

a. *date:Date*, tanggal order activity
b. *orders:Vector<Order>*, daftar order

Methods:

c. *getDate():Date*, mengembalikan nilai yang dikandung atribut *date*
d. *getOrders():Vector<Order>*, mengembalikan nilai yang dikandung atribut *orders*

27. *Class* **Transaction**

Attributes:

a. *dateTime:Date*, tanggal dan waktu data
b. *orderID:string*, nomor order dari server broker
c. *jatsID:string*, nomor order dari server JATS
d. *transactionID:string*, nomor transaksi dari server JATS
e. *code:string*, kode saham
f. *side:Side*, posisi aksi: BUY atau SELL
g. *price:float*, harga transaksi
h. *matchQty:long*, jumlah saham yang sudah match
i. *board:Board*, papan transaksi: RG (regular), TN (pasar tunai)

Methods:

j. *getDateTime():Date*, mengembalikan nilai yang dikandung atribut *dateTime.*

k. *getOrderID():String*, mengembalikan nilai yang dikandung atribut *orderID.*
l. *getJatsID():String*, mengembalikan nilai yang dikandung atribut *jatsID.*
m. *getTransactionID():String*, mengembalikan nilai yang dikandung atribut *transactionID.*
n. *getCode():String*, mengembalikan nilai yang dikandung atribut *code.*
o. *getSide():Side,* mengembalikan nilai yang dikandung atribut *side.*
p. *getPrice():float*, mengembalikan nilai yang dikandung atribut *price.*
q. *getBoard():Board*, mengembalikan nilai yang dikandung atribut *board.*
r. *getMatchQty():long*, mengembalikan nilai yang dikandung atribut *matchQty.*

28. *Class* **TransactionList**

Attributes:

a. *date:Date*, tanggal order activity
b. *transactions:Vector<Transaction>*, daftar transaksi

Methods:

c. *getDate():Date*, mengembalikan nilai yang dikandung atribut *date*
d. *getTransactions():Vector<Transaction>*, mengembalikan nilai yang dikandung atribut transactions

29. *Class* **Alert**

Attributes:

a. *dateTime:Date*, tanggal dan waktu alert
b. *title:String*, judul alert
c. *content:String*, isi alert
d. *source:String*, sumber alert

e. *priority:Priority*, prioritas alert: IMPORTANT, ORDINARY, LESS IMPORTANT

Methods:

f. *getDateTime():Date*, mengembalikan nilai yang dikandung atribut *dateTime*

g. *getTitle():String*, mengembalikan nilai yang dikandung atribut *title*

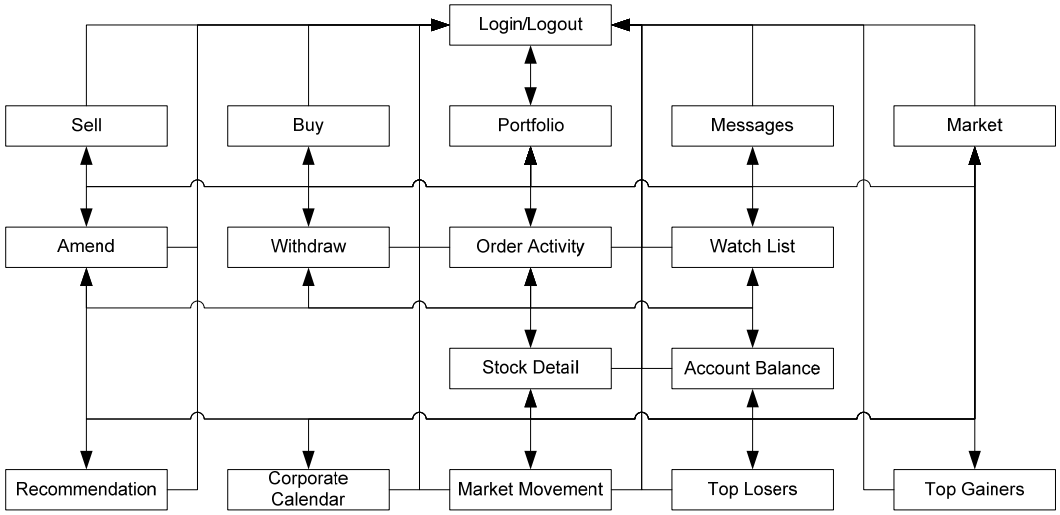
h. *getContent():String*, mengembalikan nilai yang dikandung atribut *content*

i. *getSource():String*, mengembalikan nilai yang dikandung atribut *source*

j. *getPriority():Priority*, mengembalikan nilai yang dikandung atribut *priority*

3.3. Struktur Navigasi

Struktur Navigasi yang digunakan dalam perancangan ini adalah struktur navigasi campuran.

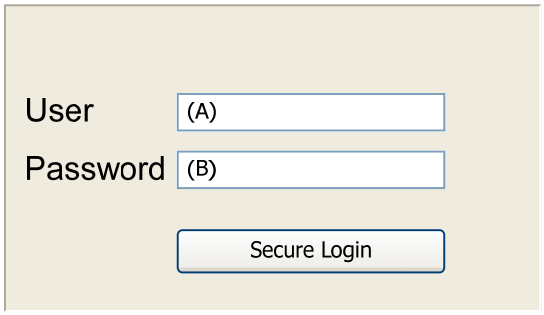


Gambar 3.12 Struktur Navigasi Mobile Trading

3.4. Perancangan Halaman

Perancangan halaman aplikasi ini terdiri dari screen-screen dan dialog-dialog yang akan mewakili use-case model yang telah dirancang sebelumnya. Halaman yang dirancang akan ditunjukan pada gambar-gambar dibawah ini.

3.4.1. Login



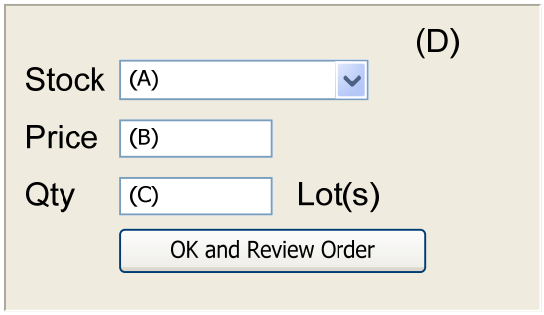
Gambar 3.13 Halaman Login

Keterangan gambar:

(A) *UserID*: Menunjukkan kode pengguna/investor. Tipe kontrol Editbox. Dapat diedit.

(B) *Password*: Menunjukkan kode *password* yang digunakan oleh investor. Tipe kontrol Editbox. Dapat diedit.

3.4.2. New Order



Gambar 3.14 Halaman New Order

Keterangan gambar:

Halaman ini akan muncul setelah *investor* memilih menu order jual atau order beli.

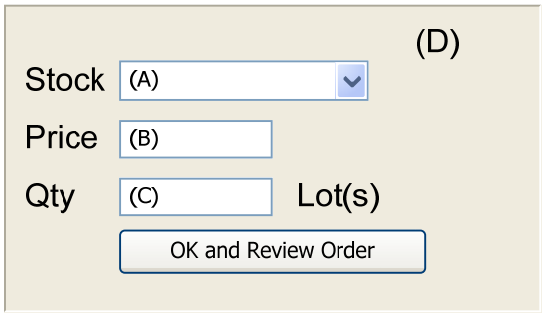
(C) *Stock Code*: Menunjukkan Stock Code yang ingin di-order. Misalnya: BUMI, ANTM, TLKM. Tipe kontrol Lookup-combo. Dapat diedit.

(D) *Price*: Menunjukkan harga stock dalam rupiah. Tipe kontrol Editbox. Dapat diedit.

(E) *Quantity*: Menunjukkan jumlah stock yang di-order. Tipe kontrol Editbox. Dapat diedit.

(F) *Position*: Posisi order, yaitu Buy (beli) atau Sell (jual). Tipe kontrol Editbox. Tidak dapat diedit.

3.4.3. Amend Order



Gambar 3.15 Halaman Amend Order

Keterangan gambar:

Halaman ini akan muncul setelah *investor* memilih menu amend pada *open order* yang terdapat di halaman portfolio.

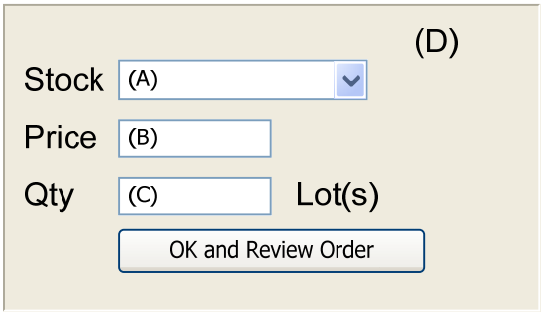
(A) *Stock Code*: Menunjukkan Stock Code yang ingin di-order. Misalnya: BUMI, ANTM, TLKM. Tipe kontrol Lookup-combo. Tidak dapat diedit.

(B) *Price*: Menunjukkan harga stock dalam rupiah. Tipe kontrol Editbox. Dapat diedit.

(C) *Quantity*: Menunjukkan jumlah stock yang di-order. Tipe kontrol Editbox. Dapat diedit.

(D) *Position*: Posisi order, yaitu Buy (beli) atau Sell (jual). Tipe kontrol Editbox. Tidak dapat diedit.

3.4.4. Withdraw Order



Gambar 3.16 Halaman Withdraw Order

Keterangan gambar:

Halaman ini akan muncul setelah *investor* memilih menu withdraw pada *open order* yang terdapat di halaman portfolio.

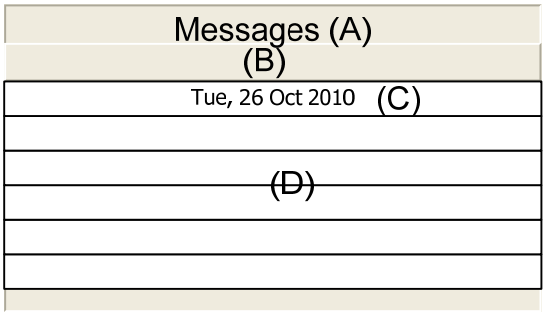
(A) *Stock Code*: Menunjukkan Stock Code yang ingin di-order. Misalnya: BUMI, ANTM, TLKM. Tipe kontrol Lookup-combo. Tidak dapat diedit.

(B) *Price*: Menunjukkan harga stock dalam rupiah. Tipe kontrol Editbox. Tidak dapat diedit.

(C) *Quantity*: Menunjukkan jumlah stock yang di-order. Tipe kontrol Editbox. Tidak dapat diedit.

(D) *Position*: Posisi order, yaitu Buy (beli) atau Sell (jual). Tipe kontrol Editbox. Tidak dapat diedit.

3.4.5. Messages



Gambar 3.17 Halaman Messages

Keterangan gambar:

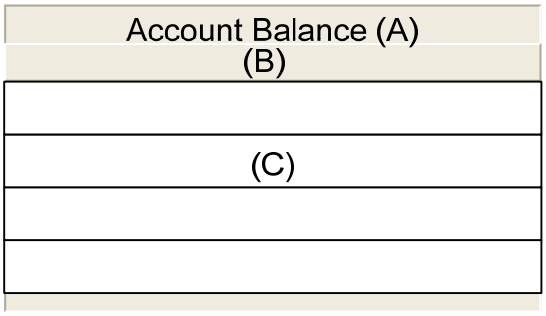
(A) Judul Halaman: Menunjukkan nama halaman.

(B) *Menu Bar*: Menunjukkan menu-menu yang tersedia.

(C) *Date*: Menunjukkan tanggal message diterima dalam format “ddd, dd mm yyyy”. Tipe Kontrol Grid Header.

(D) *Message Contents*: Berisi keterangan pesan dan waktu diterimanya pesan. Tipe Kontrol Grid Content.

3.4.6. Account Balances



Gambar 3.18 Halaman Account Balance

Keterangan gambar:

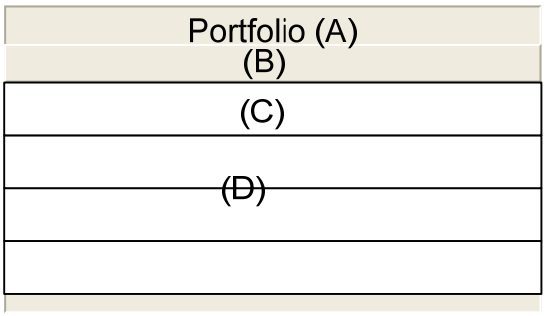
(A) Judul Halaman: Menunjukkan nama halaman.

(B) *Menu Bar*: Menunjukkan menu-menu yang tersedia.

(C) *Balance*: Menunjukkan dana yang dimiliki nasabah. Masing-masing *Row* berisi:

a. *Cash Balance*: Yaitu jumlah uang yang dimiliki.
b. *Available Funds for Trading*: Batas dana yang diperbolehkan untuk melakukan jual beli.
c. *Available for Withdrawal*: Nilai uang dari stock yang open order.
d. *Account Value*: Nilai uang dan stock yang dimiliki oleh *investor*.

3.4.7. Portfolio

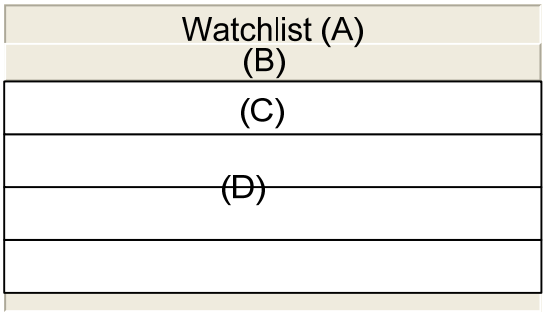


Gambar 3.19 Halaman Portfolio

Keterangan gambar:

(A) Judul Halaman: Menunjukkan nama halaman.
(B) *Menu Bar*: Menunjukkan menu-menu yang tersedia.
(C) *Header*: Berisi:
   a. *Stock Code*.
   b. *Quantity* dalam Lot.
   c. *Price*: Harga pembukaan dalam rupiah.
   d. *Last*: Harga saat ini dalam rupiah.
   e. *Change*: Perubahan harga dalam rupiah.
   f. *Gain*: Keuntungan dalam persen.
(D) *Content*: Menunjukkan Stock yang dimiliki *investor*.

3.4.8. Watch List



Gambar 3.20 Halaman Watchlist

Keterangan gambar:

Halaman ini menampilkan Stock di pasar yang ingin dimonitor oleh *investor*.

(A) Judul Halaman: Menunjukkan nama halaman.

(B) *Menu Bar*: Menunjukkan menu-menu yang tersedia.

(C) *Header*: Berisi:

a. *Stock Code*.

b. *Price*: Harga pembukaan dalam rupiah.

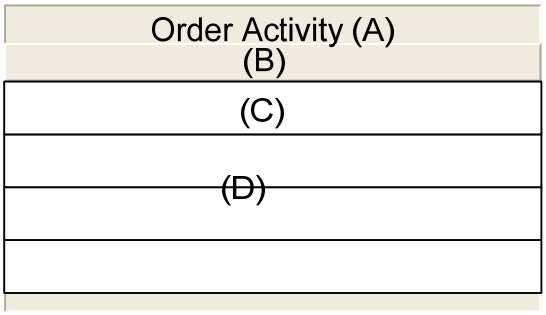
c. *Last*: Harga saat ini dalam rupiah.

d. *Change*: Perubahan harga dalam rupiah.

e. *Change(%)*:Perubahan harga dalam persen.

(D) *Content*: Menunjukkan Stock yang dimonitor di pasar.

3.4.9. Order Activity



Gambar 3.21 Halaman Order Activity

Keterangan gambar:

(A) Judul Halaman: Menunjukkan nama halaman.

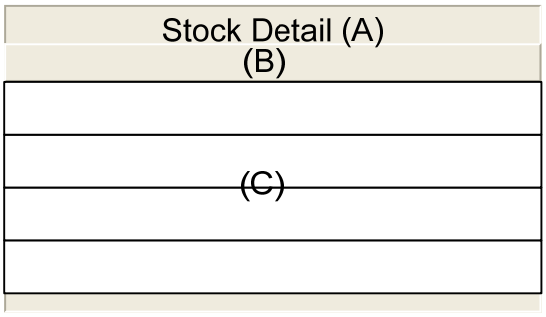
(B) *Menu Bar*: Menunjukkan menu-menu yang tersedia.

(C) *Header*: Berisi:

a. *Time*: Waktu Order dilakukan
b. *OrderID*: Identifikasi order dari BEI
c. *Stock Code*.
d. *Price*: Harga dalam rupiah.
e. *Quantity*: Jumlah dalam Lot.
f. *Status*: Status Order (*Open, Partially Match, Fully Match, Amended, Withdrawed*).

(D) *Content*: Menunjukkan isi *Order Activity*.

3.4.10. Stock Detail



Gambar 3.22 Halaman Stock Detail

Keterangan gambar:

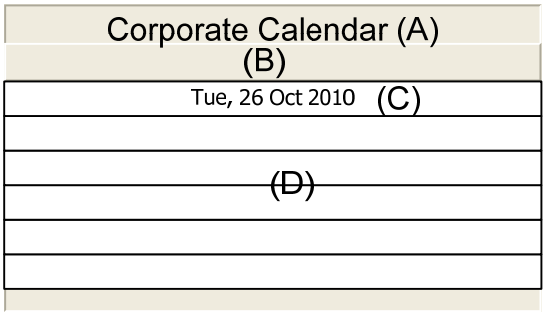
(A) Judul Halaman: Menunjukkan nama halaman.

(B) *Menu Bar*: Menunjukkan menu-menu yang tersedia.

(C) *Stock Detail*: Berisi:

a. *Last*: Harga stock terakhir dalam rupiah.
b. *Prev*: Harga stock sebelum terakhir dalam rupiah.
c. *Open*: Harga stock pembukaan dalam rupiah.
d. *High*: Harga stock tertinggi dalam rupiah.
e. *Average*: Harga stock rata-rata dalam rupiah.

3.4.11. Corporate Calendar



Gambar 3.23 Halaman Corporate Calendar

Keterangan gambar:

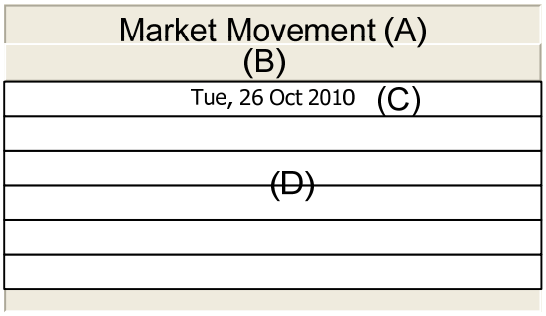
(A) Judul Halaman: Menunjukkan nama halaman.

(B) *Menu Bar*: Menunjukkan menu-menu yang tersedia.

(C) *Date*: Menunjukkan tanggal message diterima dalam format “ddd, dd mm yyyy”. Tipe Kontrol Grid Header.

(D) *Message Contents*: Berisi keterangan Corporate Calendar. Tipe Kontrol Grid Content.

3.4.12. Market Movement



Gambar 3.24 Halaman Market Movement

Keterangan gambar:

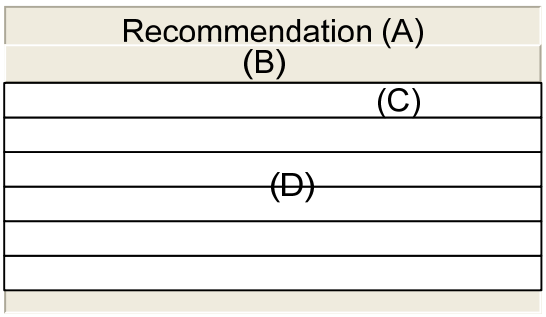
(A) Judul Halaman: Menunjukkan nama halaman.

(B) *Menu Bar*: Menunjukkan menu-menu yang tersedia.

(C) *Date*: Menunjukkan tanggal message diterima dalam format “ddd, dd mm yyyy”. Tipe Kontrol Grid Header.

(D) *Contents*: Berisi keterangan Market Movement. Tipe Kontrol Grid Content.

3.4.13. Recommendation



Gambar 3.25 Halaman Recommendation

Keterangan gambar:

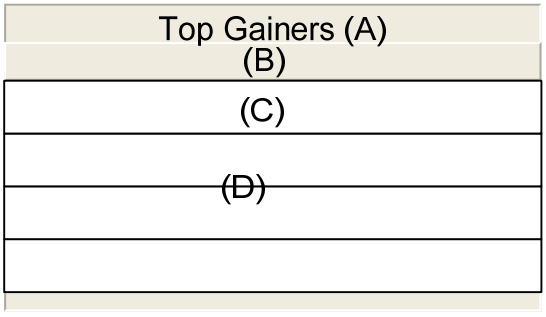
(A) Judul Halaman: Menunjukkan nama halaman.

(B) *Menu Bar*: Menunjukkan menu-menu yang tersedia.

(C) Nama Perusahaan: Menunjukkan nama perusahaan terbuka yang sahamnya dilempar ke pasar. Tipe Kontrol Grid Header.

(D) *Contents*: Berisi keterangan Rekomendasi. Tipe Kontrol Grid Content.

3.4.14. Top Gainers



Gambar 3.26 Halaman Top Gainers

Keterangan gambar:

(A) Judul Halaman: Menunjukkan nama halaman.

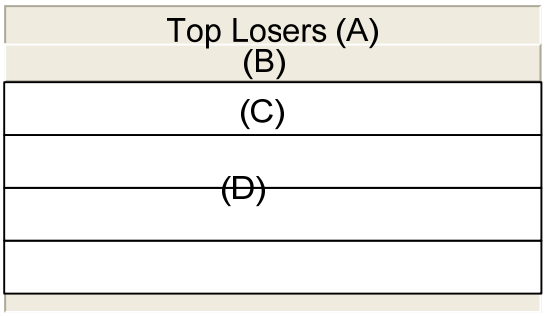
(B) *Menu Bar*: Menunjukkan menu-menu yang tersedia.

(C) *Header*: Berisi:

a. *Stock Code*.
b. *Last*: Harga terakhir dalam rupiah.
c. *Change*: Perubahan harga terakhir dibandingkan harga pembukaan dalam rupiah.
d. *Change(%)*: Perubahan harga terakhir dibandingkan harga pembukaan dalam persen.

(D) *Content*: Menunjukkan isi *Top Gainers*.

3.4.15. Top Losers



Gambar 3.27 Halaman Top Losers

Keterangan gambar:

(A) Judul Halaman: Menunjukkan nama halaman.

(B) *Menu Bar*: Menunjukkan menu-menu yang tersedia.

(C) *Header*: Berisi:

a. *Stock Code*.
b. *Last*: Harga terakhir dalam rupiah.
c. *Change*: Perubahan harga terakhir dibandingkan harga pembukaan dalam rupiah.
d. *Change(%)*: Perubahan harga terakhir dibandingkan harga pembukaan dalam persen.

(D) *Content*: Menunjukkan isi *Top Losers*.

3.5. Tabel

Dalam aplikasi ini hanya dua tabel yang digunakan, yaitu tabel Stock dan tabel Watchlist.

Tabel 3.1 Stock

Nama	Type	Panjang	Keterangan
StockCode	String	4	FK
Stock Name	String	30	

Tabel 3.2 Watchlist

Nama	Type	Panjang	Keterangan
StockCode	String	4	FK

4. PENUTUP

4.1. Kesimpulan

Perancangan aplikasi ini hanya menggambarkan proses yang terjadi dalam perangkat *mobile* sebagai aplikasi *client*. Hampir semua data disimpan dan diproses di dalam memori untuk kemudian dikirimkan ke server sehingga aplikasi hanya membutuhkan dua buah tabel, yaitu untuk menyimpan seluruh nama stock yang akan diupdate setiap hari dan menyimpan nama stock yang diinginkan pengguna untuk diamati.

Bentuk tampilan halaman dibuat sesederhana mungkin namun tetap dapat menampilkan informasi yang dibutuhkan sehingga diharapkan dapat memudahkan pengguna untuk mengamati isi halaman tersebut.

4.2. Saran

Bagaimanapun juga perancangan aplikasi ini akan banyak mengalami kendala karena keterbatasan waktu dan kemampuan penulis. Oleh karena itu seperti yang penulis telah sebutkan sebelumnya, perancangan aplikasi ini hanya menggambarkan proses yang terjadi dalam perangkat *mobile* dan dapat dikembangkan hingga ke perancangan protokol komunikasi, struktur jaringan, *load balancing* dan perancangan *server*.

DAFTAR PUSTAKA

[1] Sri Dharwiyanti, Pengantar Unified Modeling Language (UML), IlmuKomputer.Com, 2003.

[2] Chonoles, Michael Jesse, James A. Schardt, UML 2 for Dummies, Wiley Publishing, 2003

[3] Indonesia Stock Exchange, Sejarah, http://www.idx.co.id/MainMenu/TentangBEI/History/tabid/61/lang/id-ID/language/id-ID/Default.aspx, 2007

[4] Object Management Group, UML® Resource Page, http://www.omg.org, 2010.

[5] Randy Miller, Practical UML: A Hands-On Introduction for Developers, Embarcadero Technologies, Inc http://edn.embarcadero.com/together/modeling/uml, 2010.

[6] BEI, Panduan Penyelenggaraan Fasilitas Penyampaian Pesanan Secara Langsung bagi Nasabah, BEI, 2010.

[7] Janner Simarmata, Rekayasa Web, Penerbit Andi, 2010.