BAB 4

HASIL DAN PEMBAHASAN

4.1 Analisa Kebutuhan Sistem

Pada tanggal 12 Mei 2018, 18 Juli 2018, dan 29 Agustus 2018 diadakan pertemuan antara *product owner, scrum master, tech lead,* dan penulis sebagai bagian *development team.* Dari pertemuan tersebut didapatkan beberapa *user requirement* yang digunakan sebagai pedoman untuk membuat aplikasi *tuner* untuk PT. Chorus Digital Indonesia dan mengembangkan fitur untuk aplikasi Chorus yang sudah ada. *User requirement* yang didapatkan kemudian digunakan oleh penulis untuk menentukan apa saja kebutuhan yang harus terpenuhi dari fitur-fitur yang dibuat oleh penulis melalui skripsi ini.

Dari hasil penelitian yang dilakukan oleh PT. Chorus Digital Indonesia yang dicantumkan pada lampiran di bawah, 4 dari 5 orang yang di *interview* mengatakan mereka membutuhkan fitur *tuner* dalam suatu aplikasi yang berhubungan dengan memainkan alat musik. Oleh karena itu pada *user requirement* yang didapatkan dari diskusi pada tanggal 18 Juli 2018 fitur *tuner* dimasukkan ke dalam salah satu fitur yang harus dikerjakan. Fitur *tuner* rencananya akan ditempatkan pada menu *settings* pada aplikasi Chorus dan akan memunculkan popup yang menanyakan apakah pengguna sudah melakukan *tuning* terhadap instrumennya setiap kali pengguna membuka aplikasi Chorus pada awal hari.

Dari *user requirement* yang didapatkan dari diskusi pada tanggal 28 Juli 2018 didapatkan PT. Chorus Digital Indonesia akan membuat fitur *metronome* untuk membantu pengguna dalam memainkan lagu sesuai dengan ketukan dan tempo yang tepat. Fitur *metronome* nantinya direncanakan akan memiliki fitur yang disediakan bagi pengguna untuk mengganti jenis ketukan yang digunakan dalam sebuah lagu dan mengganti tempo yang digunakan sesuai dengan lagu yang dimainkan. Fitur-fitur tersebut akan disematkan di dalam fitur *metronome* untuk mempermudah pengguna untuk mengganti tempo dan jenis ketukan sesuai dengan lagu yang dimainkannya karena setiap lagu memiliki tempo dan jenis ketukan yang berbeda antara satu dengan yang lainnya. Fitur *metronome* juga akan dilengkapi fitur untuk mendeteksi

tempo dalam sebuah lagu sesuai dengan input yang diberikan pengguna dengan cara memencet sebuah tombol secara berulang-ulang sesuai dengan tempo lagu untuk mempermudah pengguna mengetahui tempo dalam sebuah lagu.

Dari *user requirement* yang didapatkan dari diskusi pada tanggal 29 Agustus 2018 didapatkan PT. Chorus Digital Indonesia akan membuat satu aplikasi baru yang di dalamnya terdapat fitur *tuner*, fitur *metronome*, dan fitur *chord library* yang dinamakan aplikasi Chorus *Tuner*. Fitur *tuner* dan *chord library* direncanakan akan disertai fungsi untuk mengubah instrumen sehingga pengguna bisa melakukan *tuning* dan melihat *chord* dari jenis instrumen yang berbeda. Fitur *tuner* juga nantinya akan disematkan fungsi untuk mengganti tipe *tuning* dan pembuatan tipe *tuning* baru dengan pilihan not sesuai keinginan pengguna sehingga pengguna bisa mengganti tipe *tuning* instrumen yang dimainkannya sesuai dengan lagu yang ingin dimainkan. Fitur *chord library* direncanakan dapat diubah arah penggambarannya untuk membantu pemain gitar kidal dalam membaca diagram *chord* yang disajikan.

Karena pengembangan fitur dalam aplikasi ini dibuat menggunakan metodologi *scrum* maka penulis membuat fitur pada aplikasi ini sesuai dengan arahan yang tertera pada *product backlog.* Beberapa tabel berikut akan menampilkan apa saja yang ada di dalam *product backlog* yang telah disusun bersama *project manager* dan *tech lead* yang berasal dari *user requirements* yang tertera diatas. Ukuran yang digunakan pada *product backlog* berikut adalah *story points* dimana satu *story points* merepresentasikan dua jam pengerjaan fitur.

Tabel 4.1 *Product Backlog Sprint* Pertama

Description	Task	Story Point	Priority
Sebagai seorang pengguna saya bisa mengetahui status *tuning* instrumen saya melalui indikator status *tuning* dan informasi status *tuning* yang jelas	Melakukan *research* tetang cara membuat *tuner* di android studio	20	High
	Membuat code untuk mendeteksi frekuensi suara yang didapatkan melalui mikrofon	20	High
	Membuat code untuk melakukan update terhadap tampilan setelah frekuensi terdeteksi	8	High
	Membuat tampilan *tuner* yang menarik	2	Medium
Sebagai seorang pengguna saya bisa mengganti tipe *tuning* yang ingin saya gunakan untuk instrumen saya	Membuat algoritma untuk mengganti tipe *tuning* pada fitur *tuner* sesuai dengan tipe *tuning* yang dipilih pengguna	3	Low
	Membuat tampilan halaman penggantian tipe *tuning*	1	Low
Sebagai seorang pengguna saya bisa menemukan dan menekan menu untuk menuju ke halaman penggantian tipe *tuning*	Membuat pilihan menu untuk penggantian tipe *tuning*	1	Low
	Mengatur peletakkan pilihan menu penggantian tipe *tuning* pada satu layout baru bersama menu pembuatan tipe *tuning*		
	Membuat tombol untuk menampilkan halaman penggantian tipe *tuning*	1	Low
Sebagai seorang pengguna saya bisa membuat tipe *tuning*	Membuat algoritma untuk mengganti tipe *tuning* pada	3	Low

Description	Task	Story Point	Priority
baru dengan not yang sesuai dengan keinginan saya	fitur tuner sesuai dengan tipe *tuning* yang dibuat pengguna		
	Membuat tampilan halaman pembuatan tipe *tuning* baru	1	Low
Sebagai seorang pengguna saya bisa menemukan dan menekan menu untuk menuju ke halaman pembuatan tipe *tuning* baru	Membuat pilihan menu untuk pembuatan tipe *tuning* baru	1	Low
	Mengatur peletakkan pilihan menu pembuatan tipe *tuning* baru pada satu layout baru bersama menu penggantian tipe *tuning*		
	Membuat tombol untuk menampilkan halaman pembuatan tipe *tuning* baru	1	Low
Sebagai seorang pengguna saya bisa kembali ke halaman *tuner* ketika tidak jadi mengganti tipe *tuning*	Membuat tombol back pada action bar halaman penggantian tipe *tuning*, halaman pembuatan tipe *tuning* baru, dan halaman *tuner setting*	1	Low
Sebagai seorang pengguna saya bisa melihat dan bisa mengetahui nada yang benar dari senar yang sedang saya *tuning* sesuai dengan tipe *tuning* yang saya pilih	Membuat indikator penunjukkan nada pada layout fitur *tuner* dan indikator status *tuning* dari senar yang dimainkan oleh pengguna	1	Medium

Tabel 4.2 *Product Backlog Sprint* Kedua

Description	Task	Story Point	Priority
Sebagai seorang pengguna saya bisa melihat daftar *chord* yang ada di dalam aplikasi	Membuat tampilan daftar *chord* yang ada di dalam aplikasi	2	High
Sebagai seorang pengguna saya bisa melakukan penggantian *chord* pada fitur *chord library*	Membuat tampilan daftar *chord* yang ada di dalam aplikasi bisa di *scroll*	6	High
	Membuat tampilan daftar *chord* yang ada di dalam aplikasi bisa dipilih saat *stop scroll*		
	Membuat tampilan daftar *chord* yang ada di dalam aplikasi bisa dipilih dengan cara ditekan oleh pengguna		
Sebagai seorang pengguna saya bisa melihat daftar varian *chord* yang ada di dalam aplikasi	Membuat tampilan daftar varian *chord* yang ada di dalam aplikasi	2	High
Sebagai seorang pengguna saya bisa melakukan penggantian varian chord pada fitur *chord library*	Membuat tampilan daftar varian *chord* yang ada di dalam aplikasi bisa di *scroll*	6	High
	Membuat tampilan daftar varian *chord* yang ada di dalam aplikasi bisa dipilih saat *stop scroll*		
	Membuat tampilan daftar varian *chord* yang ada di dalam aplikasi bisa dipilih dengan cara ditekan oleh pengguna		

Description	Task	Story Point	Priority
Sebagai seorang pengguna saya bisa melihat diagram *chord* yang tepat setelah mengganti *chord* atau varian *chord*	Melakukan *research* tetang cara membuat diagram *chord* menggunakan *custom view* di android studio	3	High
	Membuat tampilan diagram *chord* menggunakan *custom view*	4	High
	Membuat fungsi untuk mengganti tampilan diagram *chord* sesuai dengan *chord* atau varian *chord* yang dipilih oleh pengguna	4	High
Sebagai seorang pengguna saya bisa melakukan penggantian mode dari *left handed mode* ke *right handed mode* dan sebaliknya	Membuat fungsi untuk mengganti tampilan diagram *chord* ke mode left handed atau sebaliknya	2	Low
	Membuat shared preferences untuk menyimpan status aktif left handed mode terakhir		Low
Sebagai seorang pengguna saya bisa melihat status aktif *left handed mode* melalui sebuah indikator	Membuat tampilan indikator status aktif left handed mode	2	Low
	Membuat fungsi untuk mengganti tampilan indikator status aktif left handed mode		
Sebagai seorang pengguna saya bisa melihat jari mana yang harus saya gunakan untuk menekan senar pada fret gitar dengan jelas sesuai dengan nomor pada indikator diagram	Membuat tampilan penjelasan nomor jari	2	High

Description	Task	Story Point	Priority
chord			
Sebagai seorang pengguna saya bisa mengganti jenis instrumen yang ingin saya *tuning* atau yang ingin saya tampilkan diagram *chord* nya melalui menu *setting*	Membuat tampilan pemilihan jenis instrumen pada menu setting	3	Medium
	Membuat fungsi untuk mengganti instrumen ketika pengguna memilih salah satu jenis instrumen		
	Membuat shared preferences untuk menyimpan instrumen yang terakhir kali digunakan oleh pengguna		
Sebagai seorang pengguna saya bisa mengakses menu *setting* melalui halaman *tuner* atau *chord library*	Membuat tombol untuk mengakses menu setting pada action bar halaman *tuner* dan *chord library*	1	Medium
Sebagai seorang pengguna saya bisa kembali ke halaman sebelumnya ketika tidak jadi mengganti *instrumen*	Membuat tombol back pada action bar halaman setting	1	Medium

Tabel 4.3 *Product Backlog Sprint* Ketiga

Description	Task	Story Point	Priority
Sebagai seorang pengguna saya bisa melihat tampilan fitur *metronome* yang menarik dan bisa menggunakannya dengan baik	Melakukan *research* tetang cara membuat *metronome* di android studio	10	High
	Membuat fungsi untuk melakukan penghitungan tempo dan ketukan	10	High
Sebagai seorang pengguna saya bisa mengatur tempo yang ingin saya gunakan untuk fitur *metronome*	Membuat tampilan slider untuk menaikkan atau menurunkan tempo	2	High
	Membuat fungsi untuk mengganti tempo pada fitur *metronome* sesuai dengan nilai tempo yang ditentukan oleh pengguna		
Sebagai seorang pengguna saya bisa mendapatkan tempo yang tepat setelah memencet tombol *set tempo* beberapa kali	Membuat fungsi untuk mendeteksi kecepatan pengguna memencet tombol ketika pengguna memencet tombol berkali-kali	4	Medium
	Membuat fungsi untuk melakukan update terhadap indikator nilai tempo sesuai dengan tempo yang terdeteksi	2	Medium
Sebagai seorang pengguna saya bisa menemukan dan menekan tombol untuk mengatur tempo	Membuat tombol untuk mengatur tempo pada halaman fitur *metronome*	1	Medium
	Membuat tombol untuk mengatur tempo dapat ditekan		
Sebagai seorang pengguna saya bisa memilih jenis	Membuat menu untuk memilih jenis ketukan yang	1	Medium

Description	Task	Story Point	Priority
ketukan yang ingin saya gunakan untuk fitur *metronome*	akan terbuka ketika memencet tombol untuk membukanya		
	Membuat tombol untuk membuka menu untuk memilih jenis ketukan	1	Medium
	Membuat fungsi untuk mengganti jenis ketukan pada fitur *metronome* sesuai dengan jenis ketukan yang ditentukan oleh pengguna	2	Medium
Sebagai seorang pengguna saya bisa mengetahui jenis ketukan mana yang sedang saya gunakan saat ini untuk fitur *metronome*	Membuat indikator penanda jenis ketukan yang sedang digunakan pada menu untuk memilih jenis ketukan	2	Low
Sebagai seorang pengguna saya bisa menyalakan atau mematikan *metronome*	Membuat fungsi untuk menyalakan *metronome* sesuai dengan tempo dan jenis ketukan yang dipilih saat *metronome* dalam kondisi mati	4	High
	Membuat fungsi untuk mematikan *metronome* saat *metronome* dalam kondisi menyala		
Sebagai seorang pengguna saya bisa menemukan dan menekan tombol untuk menyalakan atau mematikan *metronome*	Membuat tombol untuk menyalakan atau mematikan *metronome* pada halaman fitur *metronome*	0.5	High
	Mengganti simbol pada tombol menjadi stop pada saat	0.5	High

Description	Task	Story Point	Priority
	metronome menyala		
	Mengganti simbol pada tombol menjadi play pada saat *metronome* mati		

Tabel 4.4 Waktu Pengerjaan *Product Backlog Sprint* Pertama

Description	Task	Story Point	Consumed Point
Sebagai seorang pengguna saya bisa mengetahui status *tuning* instrumen saya melalui indikator status *tuning* dan informasi status *tuning* yang jelas	Melakukan *research* tetang cara membuat *tuner* di android studio	20	34
	Membuat code untuk mendeteksi frekuensi suara yang didapatkan melalui mikrofon	20	38
	Membuat code untuk melakukan update terhadap tampilan setelah frekuensi terdeteksi	8	2
	Membuat tampilan *tuner* yang menarik	2	12
Sebagai seorang pengguna saya bisa mengganti tipe *tuning* yang ingin saya gunakan untuk instrumen saya	Membuat algoritma untuk mengganti tipe *tuning* pada fitur *tuner* sesuai dengan tipe *tuning* yang dipilih pengguna	3	1
	Membuat tampilan halaman penggantian tipe *tuning*	1	3
Sebagai seorang pengguna saya bisa menemukan dan menekan menu untuk menuju	Membuat pilihan menu untuk penggantian tipe *tuning*	1	1
	Mengatur peletakkan pilihan		

Description	Task	Story Point	Consumed Point
ke halaman penggantian tipe *tuning*	menu penggantian tipe *tuning* pada satu layout baru bersama menu pembuatan tipe *tuning*		
	Membuat tombol untuk menampilkan halaman penggantian tipe *tuning*	1	1
Sebagai seorang pengguna saya bisa membuat tipe *tuning* baru dengan not yang sesuai dengan keinginan saya	Membuat algoritma untuk mengganti tipe *tuning* pada fitur *tuner* sesuai dengan tipe *tuning* yang dibuat pengguna	3	8
	Membuat tampilan halaman pembuatan tipe *tuning* baru	1	6
Sebagai seorang pengguna saya bisa menemukan dan menekan menu untuk menuju ke halaman pembuatan tipe *tuning* baru	Membuat pilihan menu untuk pembuatan tipe *tuning* baru	1	1
	Mengatur peletakkan pilihan menu pembuatan tipe *tuning* baru pada satu layout baru bersama menu penggantian tipe *tuning*		
	Membuat tombol untuk menampilkan halaman pembuatan tipe *tuning* baru	1	1
Sebagai seorang pengguna saya bisa kembali ke halaman *tuner* ketika tidak jadi mengganti tipe *tuning*	Membuat tombol back pada action bar halaman penggantian tipe *tuning*, halaman pembuatan tipe *tuning* baru, dan halaman *tuner setting*	1	1
Sebagai seorang pengguna saya bisa melihat dan bisa mengetahui nada yang benar	Membuat indikator penunjukkan nada pada layout fitur *tuner* dan indikator status	1	6

Description	Task	Story Point	Consumed Point
dari senar yang sedang saya *tuning* sesuai dengan tipe *tuning* yang saya pilih	*tuning* dari senar yang dimainkan oleh pengguna		
Total:		**64**	**115 (28 hari)**

Tabel 4.5 Waktu Pengerjaan *Product Backlog Sprint* Kedua

Description	Task	Story Point	Consumed Point
Sebagai seorang pengguna saya bisa melihat daftar *chord* yang ada di dalam aplikasi	Membuat tampilan daftar *chord* yang ada di dalam aplikasi	2	1
Sebagai seorang pengguna saya bisa melakukan penggantian *chord* pada fitur *chord library*	Membuat tampilan daftar *chord* yang ada di dalam aplikasi bisa di *scroll*	6	3
	Membuat tampilan daftar *chord* yang ada di dalam aplikasi bisa dipilih saat *stop scroll*		
	Membuat tampilan daftar *chord* yang ada di dalam aplikasi bisa dipilih dengan cara ditekan oleh pengguna		
Sebagai seorang pengguna saya bisa melihat daftar varian *chord* yang ada di dalam aplikasi	Membuat tampilan daftar varian *chord* yang ada di dalam aplikasi	2	1
Sebagai seorang pengguna saya bisa melakukan penggantian varian chord pada	Membuat tampilan daftar varian *chord* yang ada di dalam aplikasi bisa di *scroll*	6	3

Description	Task	Story Point	Consumed Point
fitur *chord library*	Membuat tampilan daftar varian *chord* yang ada di dalam aplikasi bisa dipilih saat *stop scroll*		
	Membuat tampilan daftar varian *chord* yang ada di dalam aplikasi bisa dipilih dengan cara ditekan oleh pengguna		
Sebagai seorang pengguna saya bisa melihat diagram *chord* yang tepat setelah mengganti *chord* atau varian *chord*	Melakukan *research* tetang cara membuat diagram *chord* menggunakan *custom view* di android studio	3	2
	Membuat tampilan diagram *chord* menggunakan *custom view*	4	2
	Membuat fungsi untuk mengganti tampilan diagram *chord* sesuai dengan *chord* atau varian *chord* yang dipilih oleh pengguna	4	8
Sebagai seorang pengguna saya bisa melakukan penggantian mode dari *left handed mode* ke *right handed mode* dan sebaliknya	Membuat fungsi untuk mengganti tampilan diagram *chord* ke mode left handed atau sebaliknya	2	1
	Membuat shared preferences untuk menyimpan status aktif left handed mode terakhir		
Sebagai seorang pengguna saya bisa melihat status aktif	Membuat tampilan indikator status aktif left handed mode	2	1

Description	Task	Story Point	Consumed Point
left handed mode melalui sebuah indikator	Membuat fungsi untuk mengganti tampilan indikator status aktif left handed mode		
Sebagai seorang pengguna saya bisa melihat jari mana yang harus saya gunakan untuk menekan senar pada fret gitar dengan jelas sesuai dengan nomor pada indikator diagram *chord*	Membuat tampilan penjelasan nomor jari	2	1
Sebagai seorang pengguna saya bisa mengganti jenis instrumen yang ingin saya *tuning* atau yang ingin saya tampilkan diagram *chord* nya melalui menu *setting*	Membuat tampilan pemilihan jenis instrumen pada menu setting	3	2
	Membuat fungsi untuk mengganti instrumen ketika pengguna memilih salah satu jenis instrumen		
	Membuat shared preferences untuk menyimpan instrumen yang terakhir kali digunakan oleh pengguna		
Sebagai seorang pengguna saya bisa mengakses menu *setting* melalui halaman *tuner* atau *chord library*	Membuat tombol untuk mengakses menu setting pada action bar halaman *tuner* dan *chord library*	1	1
Sebagai seorang pengguna saya bisa kembali ke halaman sebelumnya ketika tidak jadi mengganti *instrumen*	Membuat tombol back pada action bar halaman setting	1	1
Total:		**40**	**27 (7 hari)**

Tabel 4.6 Waktu Pengerjaan *Product Backlog Sprint* Ketiga

Description	Task	Story Point	Consumed Point
Sebagai seorang pengguna saya bisa melihat tampilan fitur *metronome* yang menarik dan bisa menggunakannya dengan baik	Melakukan *research* tetang cara membuat *metronome* di android studio	10	4
	Membuat fungsi untuk melakukan penghitungan tempo dan ketukan	10	1
Sebagai seorang pengguna saya bisa mengatur tempo yang ingin saya gunakan untuk fitur *metronome*	Membuat tampilan slider untuk menaikkan atau menurunkan tempo	2	0.5
	Membuat fungsi untuk mengganti tempo pada fitur *metronome* sesuai dengan nilai tempo yang ditentukan oleh pengguna		
Sebagai seorang pengguna saya bisa mendapatkan tempo yang tepat setelah memencet tombol *set tempo* beberapa kali	Membuat fungsi untuk mendeteksi kecepatan pengguna memencet tombol ketika pengguna memencet tombol berkali-kali	4	0.5
	Membuat fungsi untuk melakukan update terhadap indikator nilai tempo sesuai dengan tempo yang terdeteksi	2	0.5
Sebagai seorang pengguna saya bisa menemukan dan menekan tombol untuk mengatur tempo	Membuat tombol untuk mengatur tempo pada halaman fitur *metronome*	1	0.5
	Membuat tombol untuk mengatur tempo dapat ditekan		

Description	Task	Story Point	Consumed Point
Sebagai seorang pengguna saya bisa memilih jenis ketukan yang ingin saya gunakan untuk fitur *metronome*	Membuat menu untuk memilih jenis ketukan yang akan terbuka ketika memencet tombol untuk membukanya	1	0.5
	Membuat tombol untuk membuka menu untuk memilih jenis ketukan	1	0.5
	Membuat fungsi untuk mengganti jenis ketukan pada fitur *metronome* sesuai dengan jenis ketukan yang ditentukan oleh pengguna	2	1
Sebagai seorang pengguna saya bisa mengetahui jenis ketukan mana yang sedang saya gunakan saat ini untuk fitur *metronome*	Membuat indikator penanda jenis ketukan yang sedang digunakan pada menu untuk memilih jenis ketukan	2	1
Sebagai seorang pengguna saya bisa menyalakan atau mematikan *metronome*	Membuat fungsi untuk menyalakan *metronome* sesuai dengan tempo dan jenis ketukan yang dipilih saat *metronome* dalam kondisi mati	4	1
	Membuat fungsi untuk mematikan *metronome* saat *metronome* dalam kondisi menyala		
Sebagai seorang pengguna saya bisa menemukan dan menekan tombol untuk	Membuat tombol untuk menyalakan atau mematikan *metronome* pada halaman	0.5	0.5

Description	Task	Story Point	Consumed Point
menyalakan atau mematikan *metronome*	fitur *metronome*		
	Mengganti simbol pada tombol menjadi stop pada saat *metronome* menyala	0.5	0.5
	Mengganti simbol pada tombol menjadi play pada saat *metronome* mati		
Total:		**40**	**12 (3 hari)**

4.2 Rancangan Sistem

4.2.1 *Use Case Diagram*

Gambar 4.1 *Use Case* Diagram Aplikasi Chorus Tuner

4.2.2 *Use Case Description*

Tabel 4.7 *Use Case Description* untuk Memilih Instrumen yang Ingin di *Tuning* dan Ditampilkan *Chord* nya

Nama:	Memilih instrumen yang ingin di *tuning* dan ditampilkan *chord* nya
Deskripsi Singkat:	Pengguna dapat memilih instrumen yang ingin di *tuning* dan ditampilkan *chord* nya pada fitur *chord library*
***Precondition*:**	Pengguna harus berada pada halaman *tuner* atau *chord library*
***Postcondition*:**	Instrumen pada halaman *tuner* dan *chord library* akan terganti sesuai dengan instrumen yang dipilih oleh pengguna
***Error Situation*:**	Ketika pengguna memilih instrumen yang sama dengan instrumen yang sebelumnya digunakan
Status sistem saat terjadi *error*:	Instrumen yang digunakan pada sistem tidak terganti
***Actor(s)*:**	Pengguna
***Trigger*:**	Pengguna menekan tombol *setting*
***Standard process*:**	(1) Pengguna memencet tombol setting (2) *System* akan menampilkan tampilan halaman penggantian instrumen (3) Pengguna memilih salah satu instrumen (4) *System* melakukan penggantian instrumen pada halaman *tuner* dan *chord library* (5) *System* mengembalikan tampilan ke halaman *tuner* atau *chord library*
***Alternative process*:**	-

Tabel 4.8 *Use Case Description* untuk Memilih Tipe *Tuning*

Nama:	Memilih tipe *tuning*
Deskripsi Singkat:	Pengguna dapat memilih tipe *tuning* dari tipe *tuning* yang sudah ada untuk mengganti tipe *tuning* sesuai dengan kebutuhan pengguna
***Precondition*:**	Pengguna harus berada di halaman pemilihan tipe *tuning*
***Postcondition*:**	Tipe *tuning* pada halaman *tuner* akan terganti sesuai dengan tipe *tuning* yang dipilih oleh pengguna
***Error Situation*:**	Ketika pengguna memilih tipe *tuning* yang sama dengan tipe *tuning* yang sebelumnya digunakan
Status sistem saat terjadi *error*:	Tipe *tuning* yang digunakan pada sistem tidak terganti
***Actor(s)*:**	Pengguna
***Trigger*:**	Pengguna menekan tombol pemilihan tipe *tuning* pada halaman penggantian tipe *tuning*
***Standard process*:**	(1) Pengguna menekan tombol pemilihan tipe *tuning* (2) *System* akan menampilkan tampilan halaman penggantian tipe *tuning* (3) Pengguna memilih salah satu tipe *tuning* (4) *System* melakukan penggantian tipe *tuning* pada halaman *tuner* (5) *System* mengembalikan tampilan ke halaman *tuner*
***Alternative process*:**	-

Tabel 4.9 *Use Case Description* untuk Membuat Tipe *Tuning* Baru

Nama:	Membuat tipe *tuning* baru
Deskripsi Singkat:	Pengguna dapat membuat tipe *tuning* baru untuk menyesuaikan *tuning* instrumen sesuai dengan kebutuhan pengguna
***Precondition*:**	Pengguna harus berada di halaman pemilihan tipe *tuning*
***Postcondition*:**	Tipe *tuning* pada halaman *tuner* akan terganti sesuai dengan tipe *tuning* yang dibuat oleh pengguna

***Error Situation*:**	Ketika pengguna membuat tipe *tuning* yang sama dengan tipe *tuning* yang sudah ada pada sistem
Status sistem saat terjadi *error*:	Tipe *tuning* yang digunakan pada sistem tetap disesuaikan dengan tipe tuning yang dibuat oleh pengguna
***Actor(s)*:**	Pengguna
***Trigger*:**	Pengguna menekan tombol pembuatan tipe *tuning* baru pada halaman penggantian tipe *tuning*
***Standard process*:**	(1) Pengguna menekan tombol pembuatan tipe *tuning* baru (2) *System* akan menampilkan tampilan halaman pembuatan tipe *tuning* baru (3) Pengguna memilih not untuk tipe *tuning* yang dibuat (4) Pengguna menekan tombol *confirm* (5) *System* akan membuat tipe *tuning* baru sesuai not yang dipilih pengguna dan mengganti tipe *tuning* pada halaman *tuner* (6) *System* mengembalikan tampilan ke halaman *tuner*
***Alternative process*:**	-

Tabel 4.10 *Use Case Description* untuk Melakukan *Tuning* terhadap Instrumen yang Dipilih Oleh Pengguna

Nama:	Melakukan *tuning* terhadap instrumen yang dipilih oleh pengguna
Deskripsi Singkat:	Pengguna dapat melakukan *tuning* terhadap instrumen yang ingin dimainkan olehnya
***Precondition*:**	Pengguna harus berada di halaman *tuner*
***Postcondition*:**	Indikator *tuner* akan menunjukkan status senar dari instrumen yang di *tuning* apakah sudah *in tune*, *too low*, atau *too high*
***Error Situation*:**	Ketika frekuensi senar yang dipetik pengguna lebih dari batas maksimal atau kurang dari batas minimal frekuensi yang dapat terdeteksi
Status sistem saat	Sistem akan melakukan set frekuensi senar secara otomatis

terjadi *error*:	menjadi frekuensi minimal ketika frekuensi yang terdeteksi dibawah batas minimal dan menjadi frekuensi maksimal ketika frekuensi yang terdeteksi diatas batas maksimal
***Actor(s)*:**	Pengguna
***Trigger*:**	Pengguna memetik senar instrumennya pada saat berada pada halaman *tuner*
***Standard process*:**	(1) Pengguna memetik senar instrumennya (2) *System* menangkap suara yang dihasilkan senar instrumen (3) *System* melakukan perhitungan frekuensi dari suara yang ditangkap (4) *System* menentukan status senar instrumen apakah sudah *in tune*, *too low*, atau *too high* (5) *System* akan melakukan *update* terhadap tampilan sesuai status *tuning* senar instrumen
***Alternative process*:**	-

Tabel 4.11 *Use Case Description* untuk Memilih *Chord* yang Ingin Ditampilkan pada Diagram *Chord*

Nama:	Memilih *chord* yang ingin ditampilkan pada Diagram *Chord*
Deskripsi Singkat:	Pengguna dapat memilih *chord* mana yang ingin ditampilkan pada diagram c*hord* dari antara *chord* yang ada pada daftar *chord*
***Precondition*:**	Pengguna harus berada di halaman *chord library*
***Postcondition*:**	Diagram c*hord* akan berubah sesuai dengan *chord* yang dipilih
***Error Situation*:**	Ketika pengguna memilih *chord* yang sama dengan *chord* yang sebelumnya digunakan
Status sistem saat terjadi *error*:	*Chord* yang digunakan untuk menampilkan diagram *chord* dan diagram *chord* tidak akan terganti
***Actor(s)*:**	Pengguna

***Trigger*:**	Pengguna menekan salah satu pilihan pada daftar *chord* atau melakukan *scroll* pada daftar *chord*
***Standard process*:**	(1) Pengguna menekan salah satu pilihan pada daftar *chord* (2) Sistem mengambil data tentang *chord* dari *chord* yang dipilih dari *database* (3) Sistem mengubah tampilan diagram chord sesuai dengan *chord* dan varian *chord* yang dipilih
***Alternative process*:**	(1') Pengguna melakukan *scroll* pada daftar *chord* (2') Sistem mengambil data tentang *chord* dari *chord* yang dipilih dari *database* (3') Sistem mengubah tampilan diagram chord sesuai dengan *chord* dan varian *chord* yang dipilih

Tabel 4.12 *Use Case Description* untuk Memilih Varian *Chord* yang Ingin Ditampilkan pada Diagram *Chord*

Nama:	Memilih varian *chord* yang ingin ditampilkan pada Diagram *Chord*
Deskripsi Singkat:	Pengguna dapat memilih varian *chord* mana yang ingin ditampilkan pada diagram c*hord* dari antara varian *chord* yang ada pada daftar varian *chord*
***Precondition*:**	Pengguna harus berada di halaman *chord library*
***Postcondition*:**	Diagram c*hord* akan berubah sesuai dengan varian *chord* yang dipilih
***Error Situation*:**	Ketika pengguna memilih varian *chord* yang sama dengan varian *chord* yang sebelumnya digunakan
Status sistem saat terjadi *error*:	Varian *chord* yang digunakan untuk menampilkan diagram *chord* dan diagram *chord* tidak akan terganti
***Actor(s)*:**	Pengguna
***Trigger*:**	Pengguna menekan salah satu pilihan pada daftar varian *chord* atau melakukan *scroll* pada daftar varian *chord*
***Standard process*:**	(1) Pengguna menekan salah satu pilihan pada daftar varian *chord*

	(2) Sistem mengambil data tentang varian *chord* dari *chord* yang dipilih dari *database* (3) Sistem mengubah tampilan diagram chord sesuai dengan *chord* dan varian *chord* yang dipilih
***Alternative process*:**	(1') Pengguna melakukan *scroll* pada daftar varian *chord* (2') Sistem mengambil data tentang varian *chord* dari *chord* yang dipilih dari *database* (3') Sistem mengubah tampilan diagram chord sesuai dengan *chord* dan varian *chord* yang dipilih

Tabel 4.13 *Use Case Description* untuk Mengganti Mode untuk Menampilkan *Chord*

Nama:	Mengganti mode untuk menampilkan c*hord*
Deskripsi Singkat:	Pengguna dapat mengganti mode untuk menampilkan *chord diagram* sesuai tangan yang dominan (*left handed* atau *right handed*)
***Precondition*:**	Pengguna harus berada di halaman *chord library* Mode yang dipilih sebelumnya harus berlawanan dengan mode yang ingin dipilih selanjutnya
***Postcondition*:**	Sistem akan menampilkan *chord diagram* sesuai dengan mode yang dipilih oleh pengguna
***Error Situation*:**	Ketika pengguna merupakan orang yang menggunakan tangan kiri (kidal) namun melihat diagram *chord* pada mode tangan kanan
Status sistem saat terjadi *error*:	Sistem akan tetap menampilkan *chord diagram* sesuai dengan mode yang dipilih oleh pengguna
***Actor(s)*:**	Pengguna
***Trigger*:**	Pengguna menekan tombol mode *left handed*
***Standard process*:**	(1) Pengguna menekan tombol mode *left handed* (2) Sistem mengganti arah baca diagram *chord* sesuai dengan status mode *left handed* apakah sedang aktif atau tidak

***Alternative process*:**	-

Tabel 4.14 *Use Case Description* untuk Menyalakan atau Mematikan *Metronome* Sesuai dengan Tempo dan Jenis Ketukan yang Dipilih Pengguna

Nama:	Menyalakan atau mematikan *metronome* sesuai dengan tempo dan jenis ketukan yang dipilih pengguna
Deskripsi Singkat:	Pengguna dapat menyalakan atau mematikan *metronome* sesuai keinginannya setelah melakukan pemilihan tempo dan jenis ketukan
***Precondition*:**	Pengguna harus berada di halaman *metronome*
***Postcondition*:**	*Metronome* akan mati dan suara dari metronome juga akan mati jika sebelumnya *metronome* dalam keadaan menyala *Metronome* akan menyala dan akan menghasilkan suara sesuai tempo dan ketukan yang telah ditentukan jika sebelumnya *metronome* dalam keadaan mati
***Error Situation*:**	Ketika pengguna mematikan *metronome* saat ketukan tidak berada di awal
Status sistem saat terjadi *error*:	Sistem akan mengembalikan ketukan secara otomatis ke bagian awal
***Actor(s)*:**	Pengguna
***Trigger*:**	Pengguna menekan tombol *play / stop* pada halaman *metronome*
***Standard process*:**	(1) Pengguna menekan tombol *play / stop* (2) Sistem akan mematikan *metronome* dan suara dari metronome juga akan mati jika sebelumnya *metronome* dalam keadaan menyala
***Alternative process*:**	(2') Sistem akan menyalakan *metronome* dan akan menghasilkan suara sesuai tempo dan ketukan yang telah ditentukan jika sebelumnya *metronome* dalam keadaan mati

Tabel 4.15 *Use Case Description* untuk Mengatur Kecepatan Ketukan *Metronome*

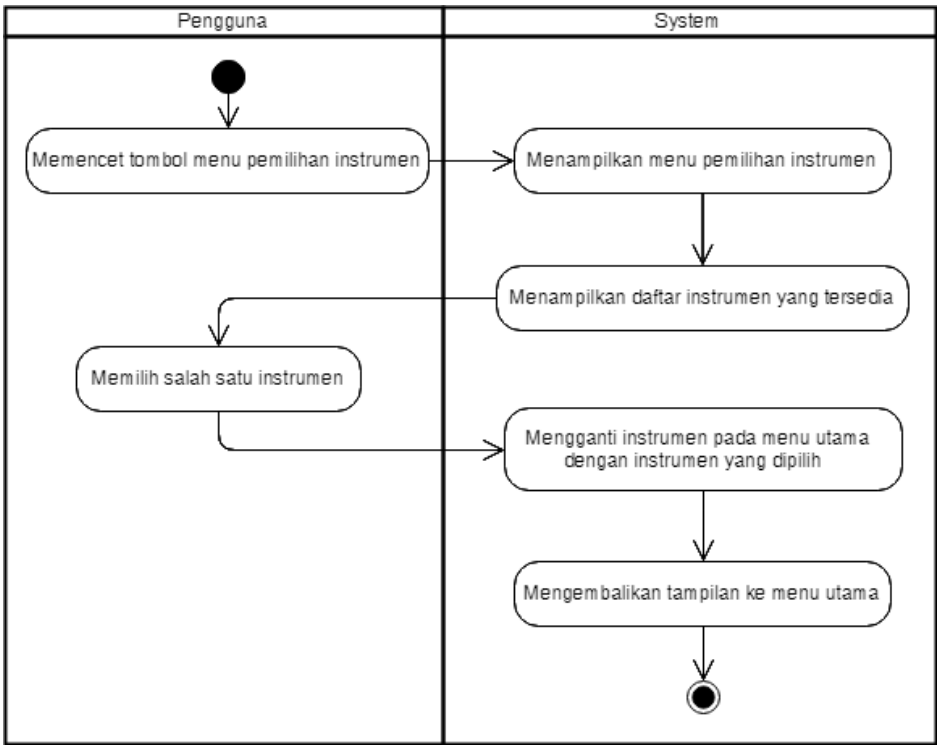
Nama:	Mengatur kecepatan ketukan (tempo) *metronome*
Deskripsi Singkat:	Pengguna mengatur kecepatan ketukan (tempo) *metronome* sesuai dengan tempo lagu yang ingin dimainkannya
***Precondition*:**	Pengguna harus berada di halaman *metronome* *Metronome* harus dalam keadaan mati
***Postcondition*:**	Tempo *metronome* akan berubah, suara yang dihasilkan ataupun indikator metronome akan bergerak dengan selang waktu sesuai tempo yang ditentukan
***Error Situation*:**	Ketika pengguna melakukan set tempo lebih dari batas maksimal atau kurang dari batas minimal tempo Ketika set tempo dilakukan pada saat *metronome* dalam keadaan menyala
Status sistem saat terjadi *error*:	Sistem akan melakukan set tempo secara otomatis menjadi tempo minimal ketika pengguna melakukan set tempo dibawah batas minimal dan menjadi tempo maksimal ketika pengguna melakukan set tempo diatas batas maksimal Sistem akan mematikan *metronome* terlebih dahulu kemudian akan menyalakan *metronome* kembali ketika set tempo telah selesai dilakukan
***Actor(s)*:**	Pengguna
***Trigger*:**	Melakukan penggeseran pada meteran untuk set tempo atau melakukan penekanan pada tombol set tempo
***Standard process*:**	(1) Pengguna melakukan penggeseran pada meteran untuk set tempo (2) Sistem akan melakukan set tempo sesuai tempo yang ditunjuk oleh meteran
***Alternative process*:**	(1') Pengguna melakukan penekanan pada tombol set tempo (2') Sistem akan melakukan set tempo sesuai kecepatan pengguna menekan tombol set tempo

Tabel 4.16 *Use Case Description* untuk Mengatur Jenis Ketukan yang Ingin Digunakan saat Menggunakan *Metronome*

Nama:	Mengatur jenis ketukan yang ingin digunakan saat menggunakan *metronome*
Deskripsi Singkat:	Pengguna melakukan pemilihan jenis ketukan yang ingin digunakan saat menggunakan *metronome*
***Precondition*:**	Pengguna harus berada di halaman *metronome* Pengguna harus membuka menu penggantian jenis ketukan
***Postcondition*:**	Jenis ketukan yang digunakan untuk metronome akan berubah sesuai dengan jenis ketukan yang dipilih
***Error Situation*:**	Ketika pengguna memilih jenis ketukan yang sama dengan jenis ketukan yang sebelumnya digunakan
Status sistem saat terjadi *error*:	Jenis ketukan yang digunakan pada sistem tidak terganti
***Actor(s)*:**	Pengguna
***Trigger*:**	Memilih salah satu jenis ketukan dari menu pemilihan jenis ketukan
***Standard process*:**	(1) Pengguna memilih salah satu jenis ketukan dari menu pemilihan jenis ketukan (2) *System* akan melakukan setting terhadap *metronome* sesuai dengan jenis ketukan yang dipilih
***Alternative process*:**	-

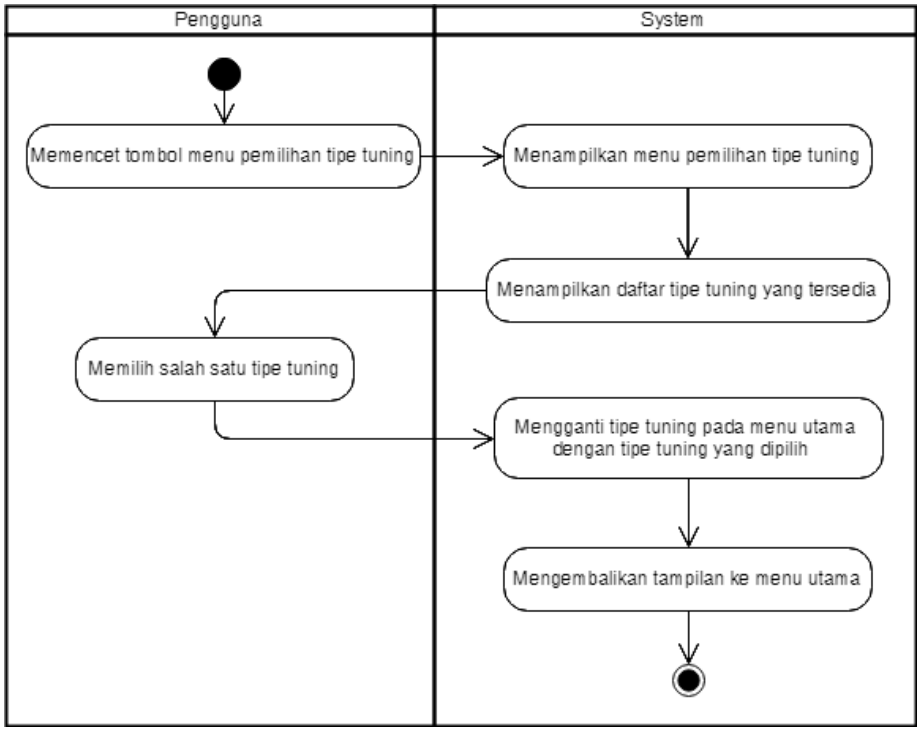
4.2.3 *Activity Diagram*

1. Memilih instrumen yang ingin di *tuning* dan ditampilkan *chord* nya



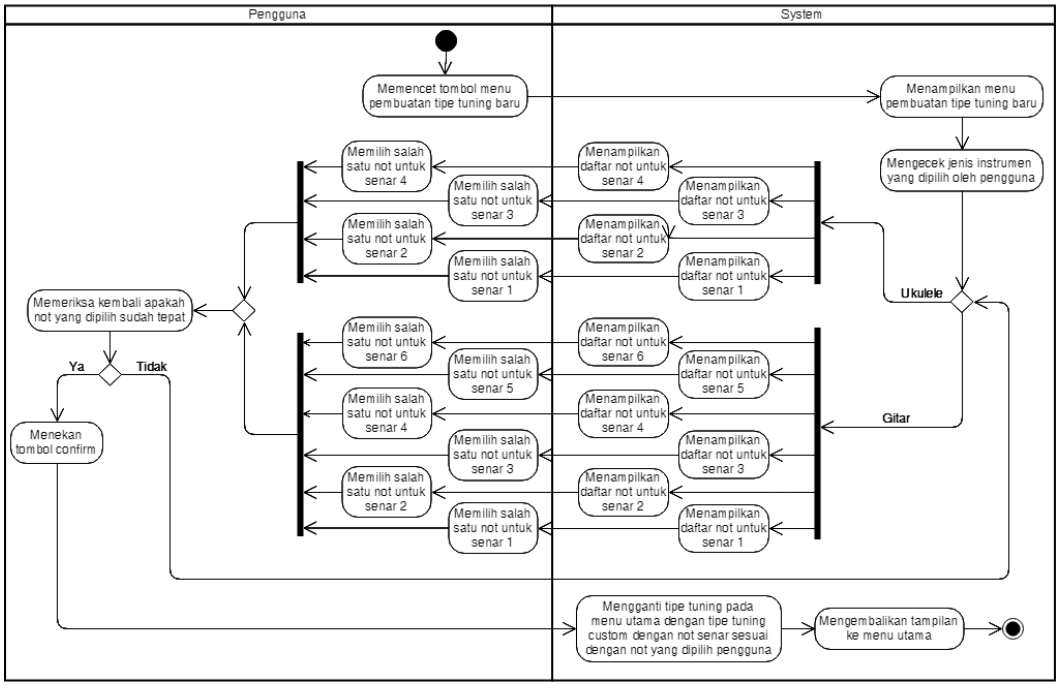
Gambar 4.2 *Activity* Diagram untuk Memilih Instrumen yang Ingin di *Tuning* dan Ditampilkan *Chord* nya

2. Memilih tipe *tuning*



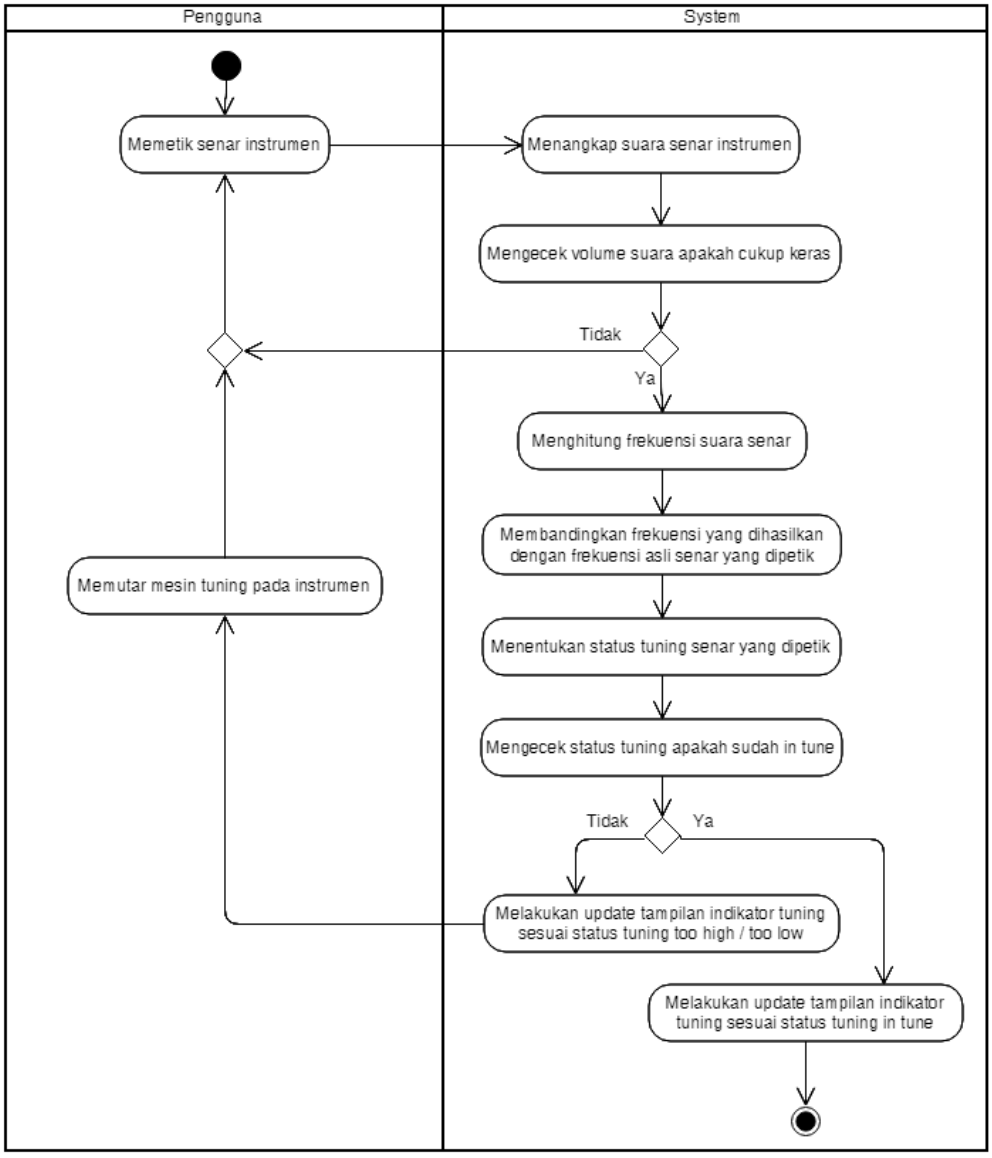
Gambar 4.3 *Activity* Diagram untuk Memilih Tipe *Tuning*

3. Membuat tipe *tuning* baru



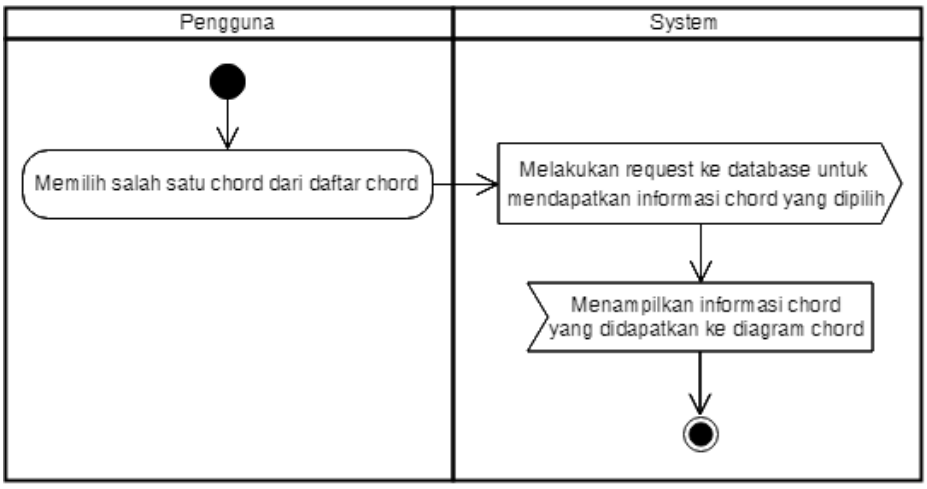
Gambar 4.4 *Activity* Diagram untuk Membuat Tipe *Tuning* Baru

4. Melakukan *tuning* terhadap instrumen yang dipilih oleh pengguna



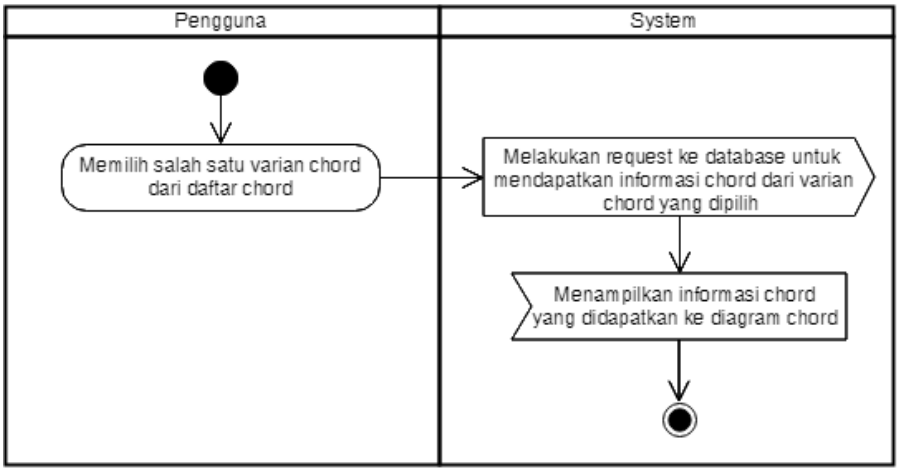
Gambar 4.5 *Activity* Diagram untuk Melakukan *Tuning* terhadap Instrumen yang Dipilih Oleh Pengguna

5. Memilih *chord* yang ingin ditampilkan pada Diagram *Chord*



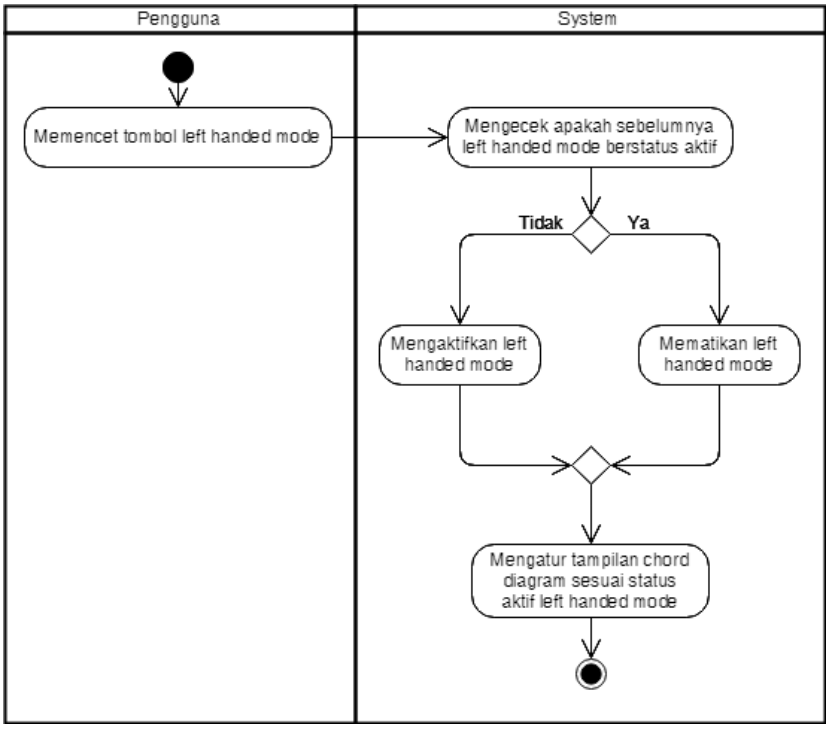
Gambar 4.6 *Activity* Diagram untuk Memilih *Chord* yang Ingin Ditampilkan pada Diagram *Chord*

6. Memilih varian *chord* yang ingin ditampilkan pada Diagram *Chord*



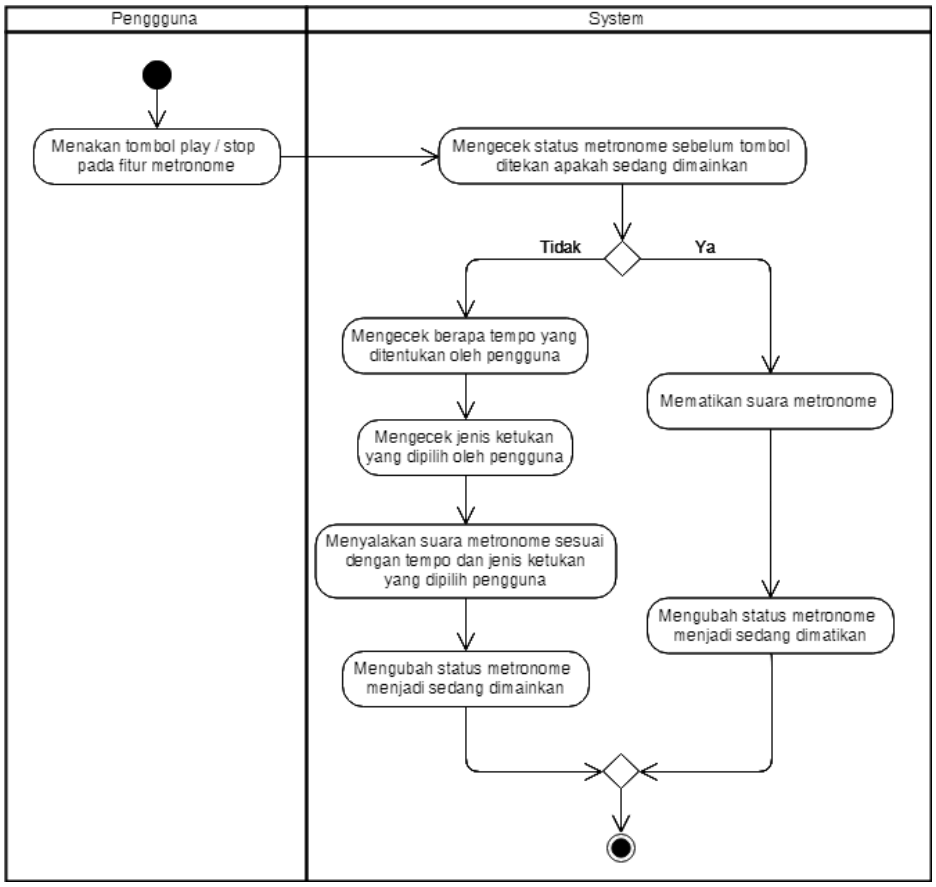
Gambar 4.7 *Activity* Diagram untuk Memilih Varian *Chord* yang Ingin Ditampilkan pada Diagram *Chord*

7. Mengganti mode untuk menampilkan *chord*



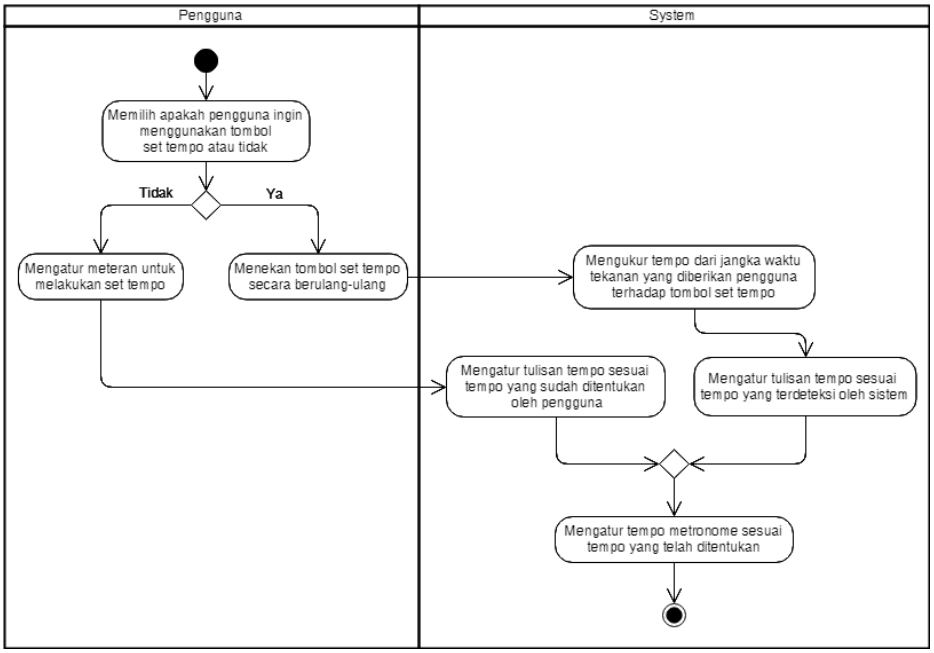
Gambar 4.8 *Activity* Diagram untuk Mengganti Mode untuk Menampilkan *Chord*

8. Menyalakan atau mematikan *metronome* sesuai dengan tempo dan jenis ketukan yang dipilih pengguna



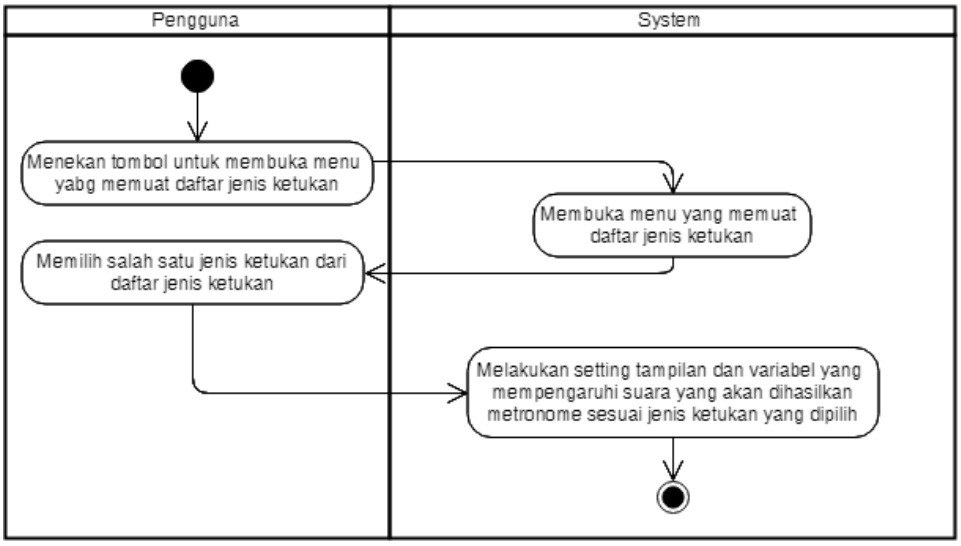
Gambar 4.9 *Activity* Diagram untuk Menyalakan atau Mematikan *Metronome* Sesuai dengan Tempo dan Jenis Ketukan yang Dipilih Pengguna

9. Mengatur kecepatan ketukan (tempo) *metronome*



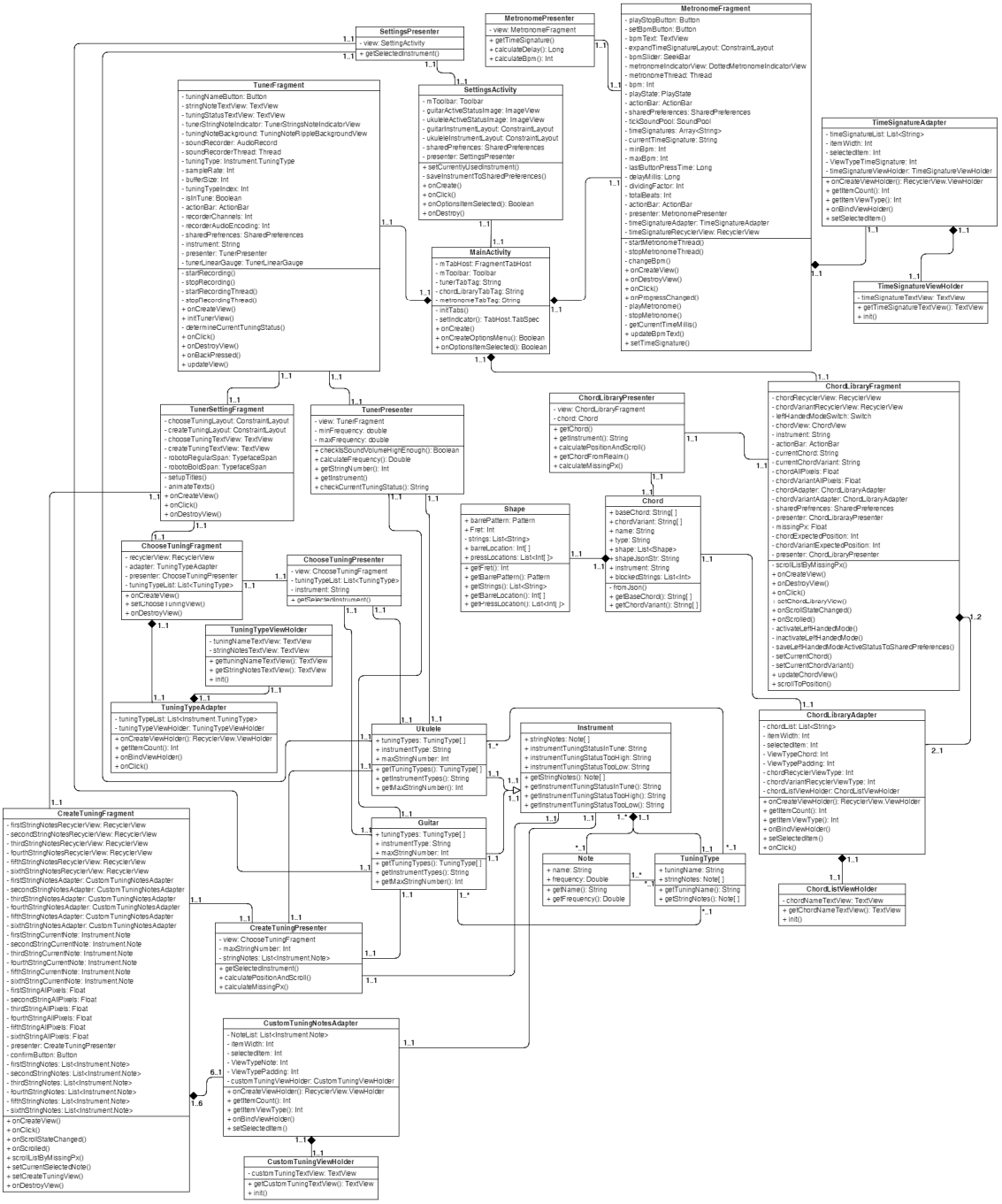
Gambar 4.10 *Activity* Diagram untuk Mengatur Kecepatan Ketukan *Metronome*

10. Mengatur jenis ketukan yang ingin digunakan saat menggunakan *metronome*



Gambar 4.11 *Activity* Diagram untuk Mengatur Jenis Ketukan yang Ingin Digunakan saat Menggunakan *Metronome*

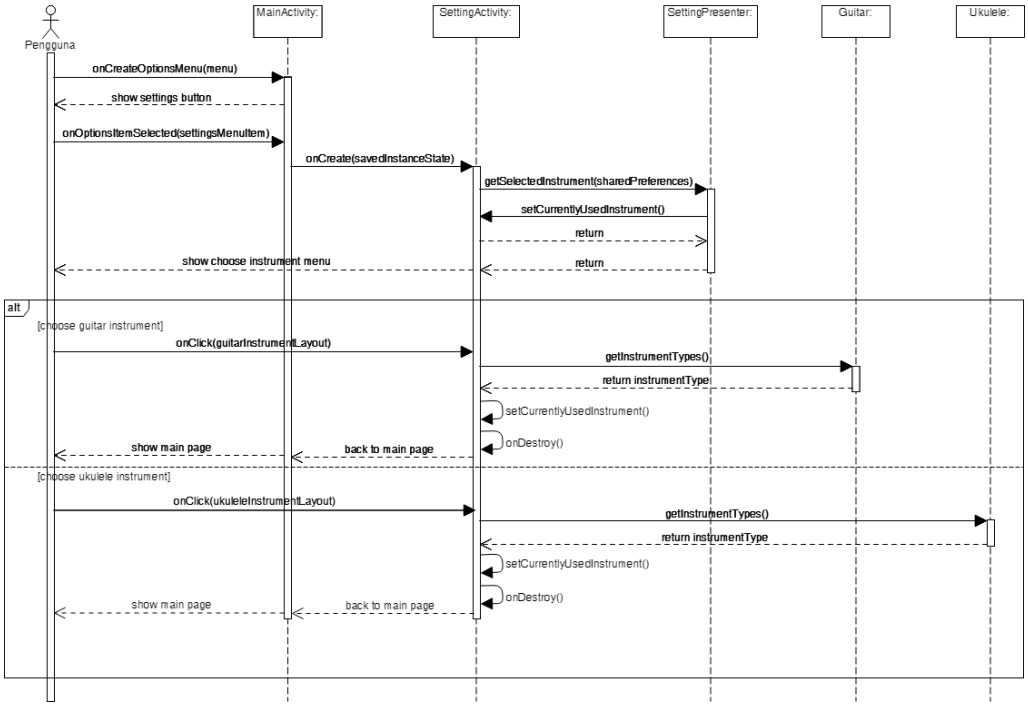
4.2.4 *Class Diagram*



Gambar 4.12 *Class* Diagram Aplikasi Chorus Tuner

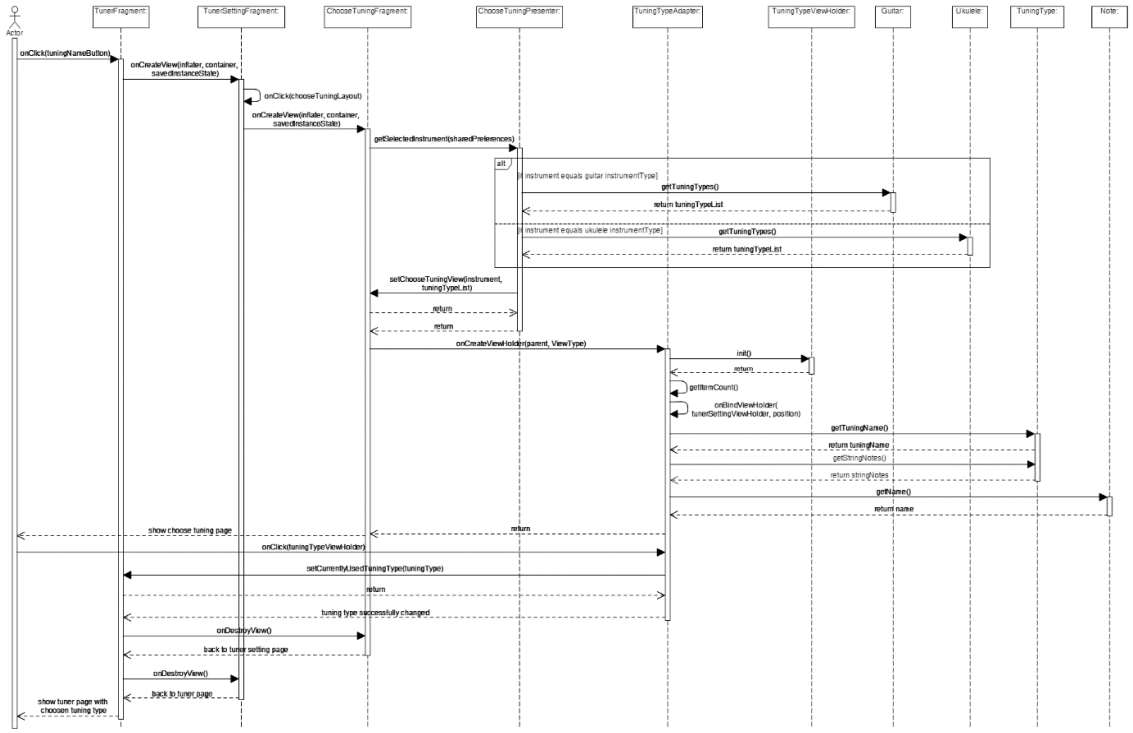
4.2.5 *Sequence Diagram*

1. Memilih instrumen yang ingin di *tuning* dan ditampilkan *chord* nya



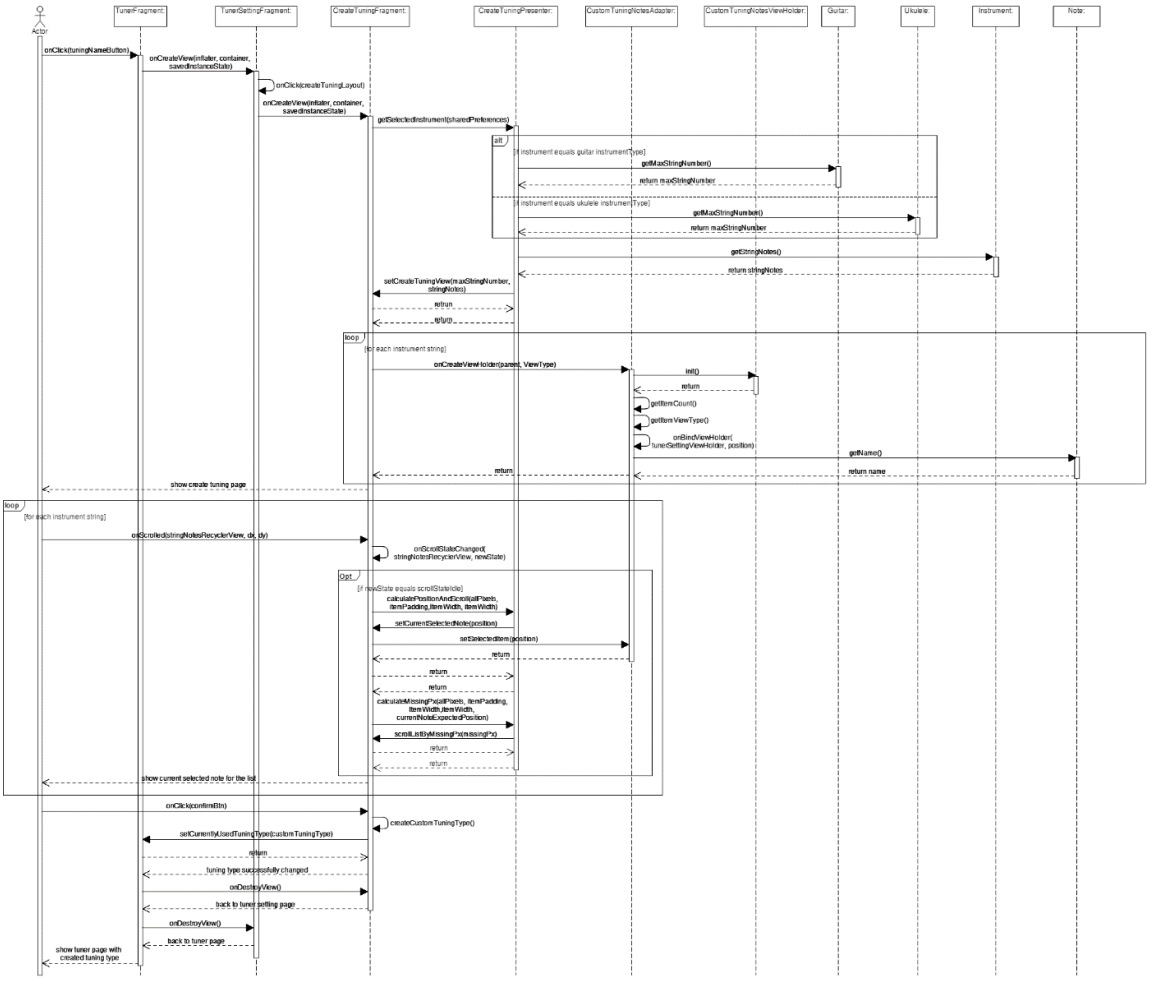
Gambar 4.13 *Sequence* Diagram untuk Memilih Instrumen yang Ingin di *Tuning* dan Ditampilkan *Chord* nya

2. Memilih tipe *tuning*



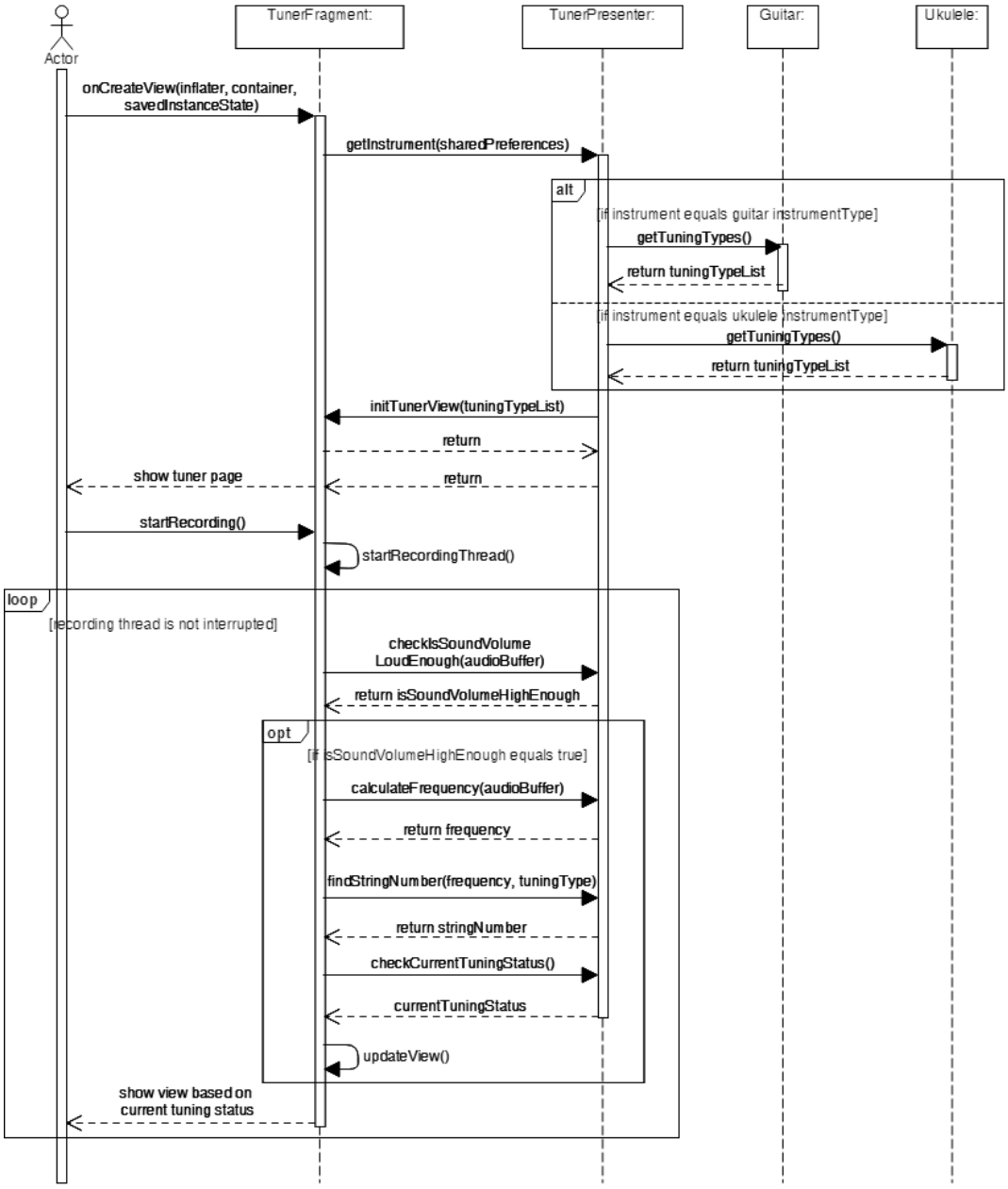
Gambar 4.14 *Sequence* Diagram untuk Memilih Tipe *Tuning*

3. Membuat Tipe *Tuning* Baru



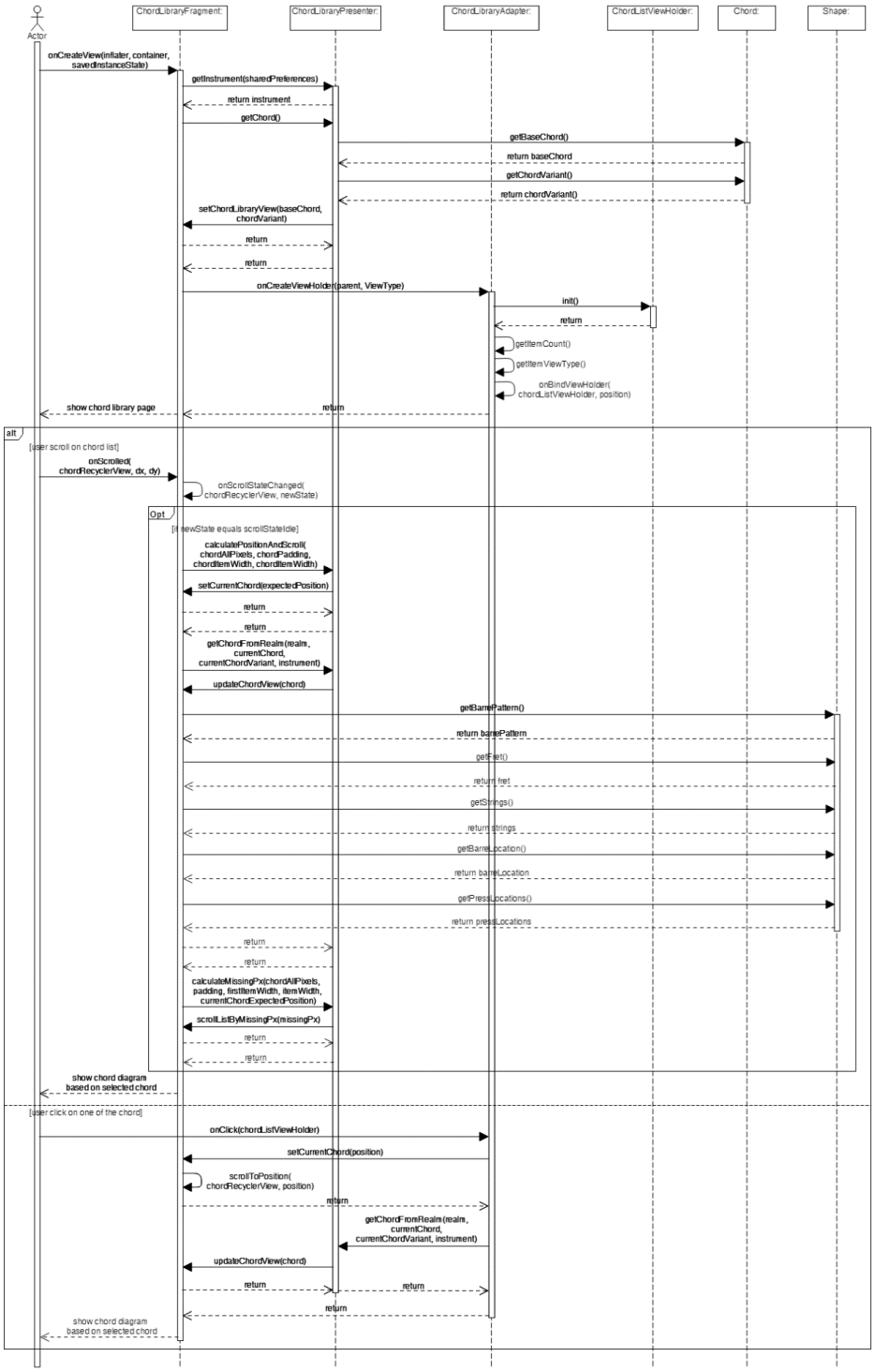
Gambar 4.15 *Sequence* Diagram untuk Membuat Tipe *Tuning* Baru

4. Melakukan *tuning* terhadap instrumen yang dipilih oleh pengguna



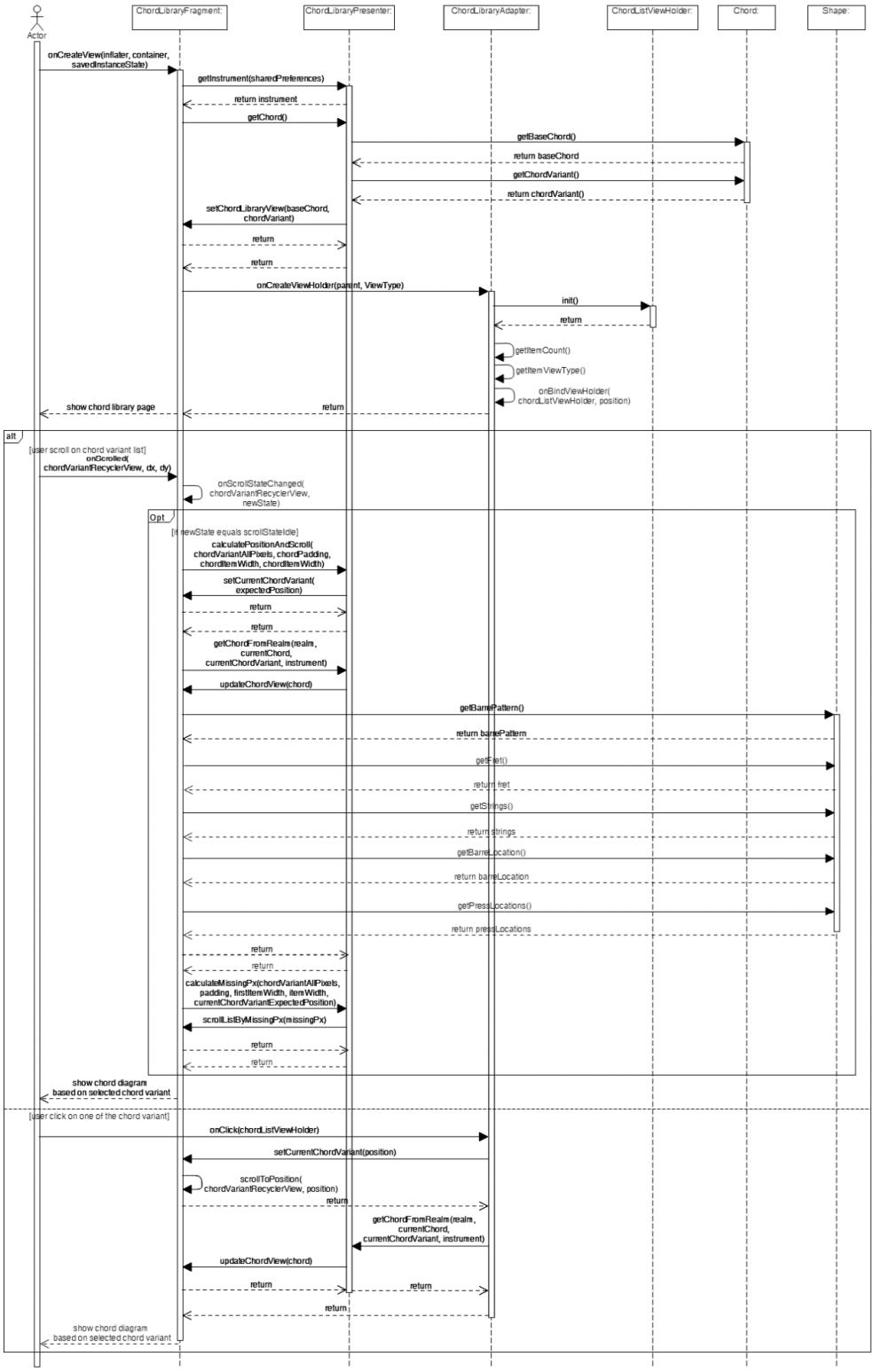
Gambar 4.16 *Sequence* Diagram untuk Melakukan *Tuning* terhadap Instrumen yang Dipilih *Oleh Pengguna*

5. Memilih *chord* yang ingin ditampilkan pada Diagram *Chord*



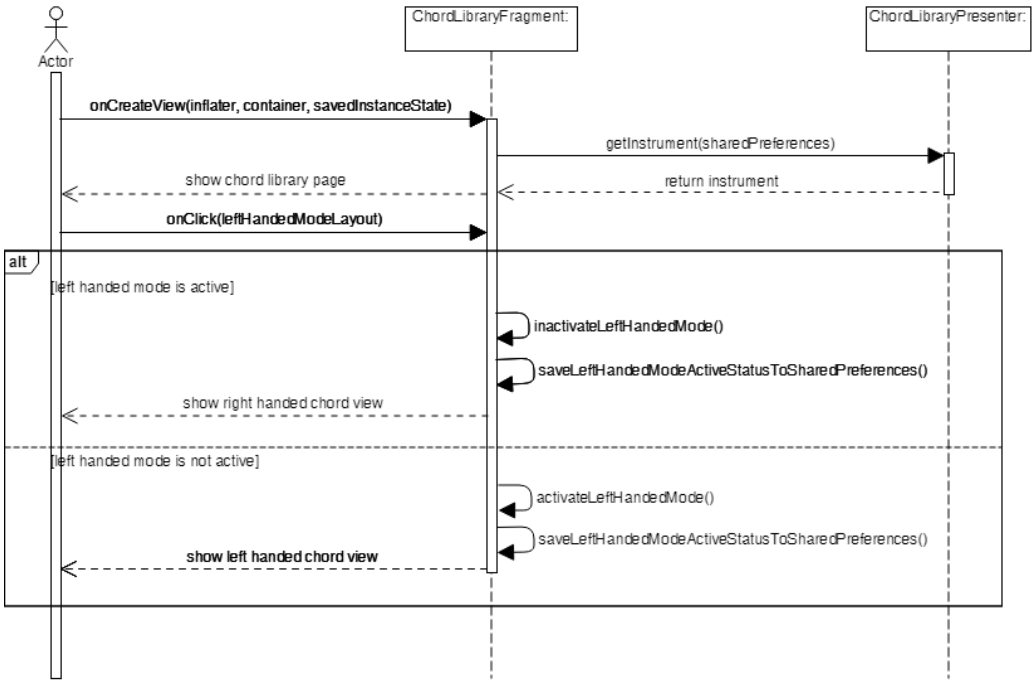
Gambar 4.17 *Sequence* Diagram untuk Memilih *Chord* yang Ingin Ditampilkan pada Diagram *Chord*

6. Memilih varian *chord* yang ingin ditampilkan pada Diagram *Chord*



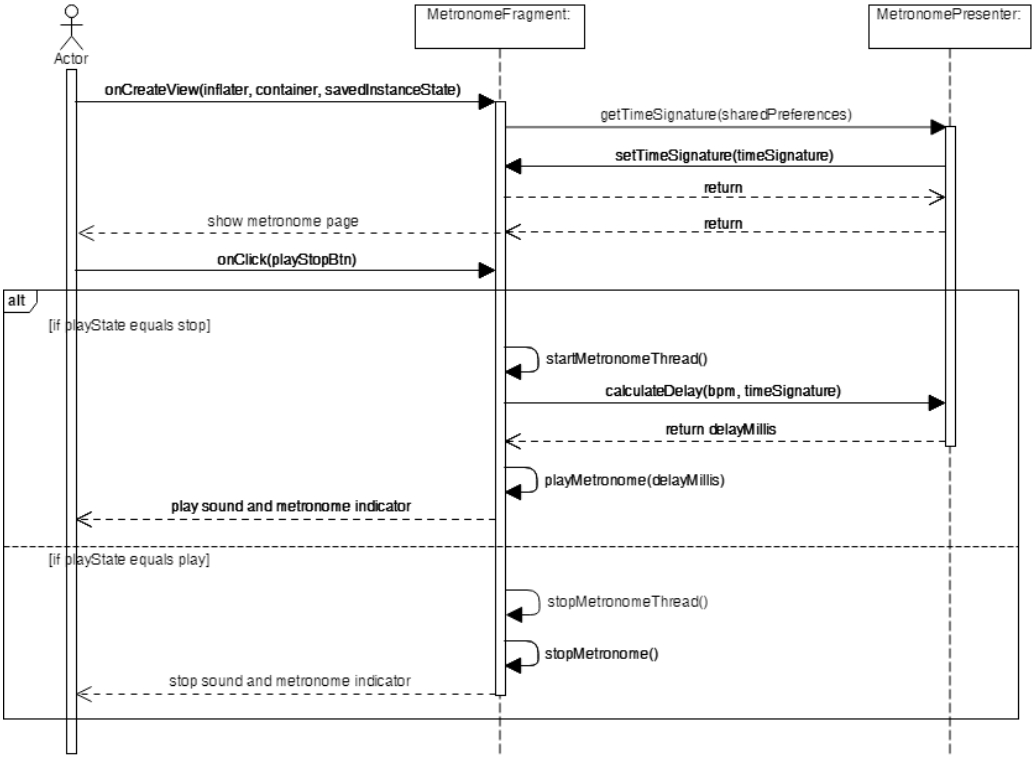
Gambar 4.18 *Sequence* Diagram untuk Memilih Varian *Chord* yang Ingin Ditampilkan pada *Diagram Chord*

7. Mengganti Mode untuk Menampilkan *Chord*



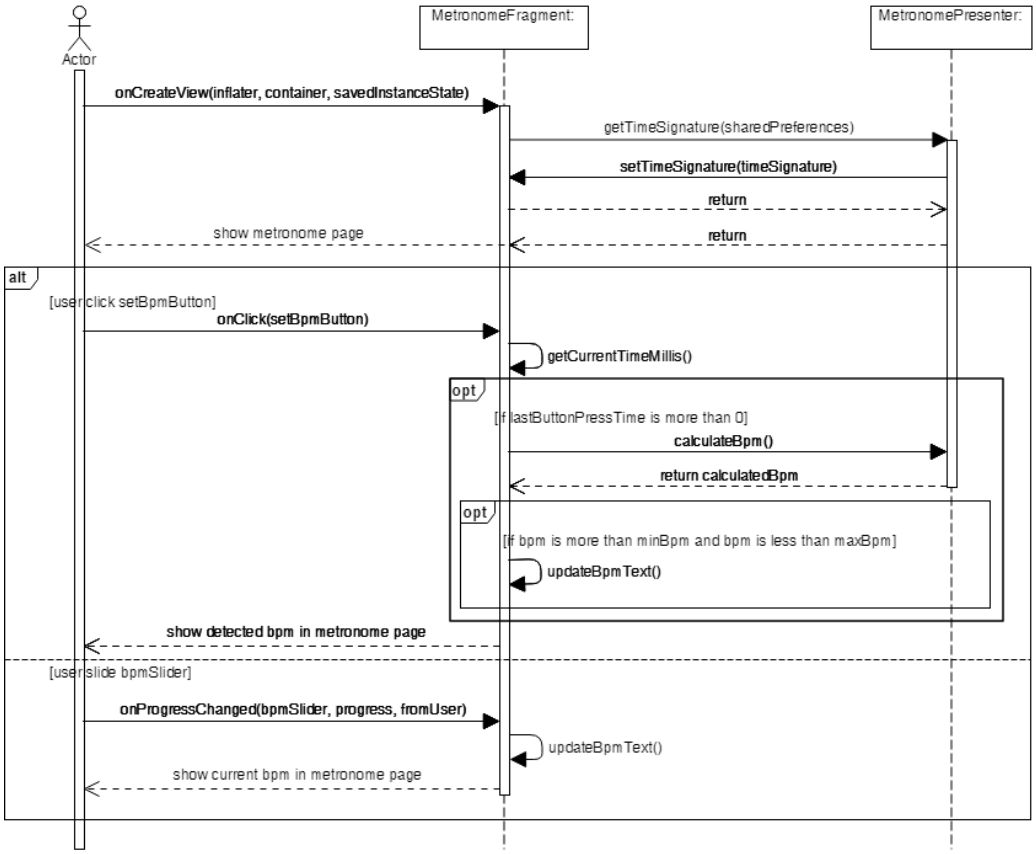
Gambar 4.19 *Sequence* Diagram untuk Mengganti Mode untuk Menampilkan *Chord*

8. Menyalakan atau mematikan *metronome* sesuai dengan tempo dan jenis ketukan yang dipilih pengguna



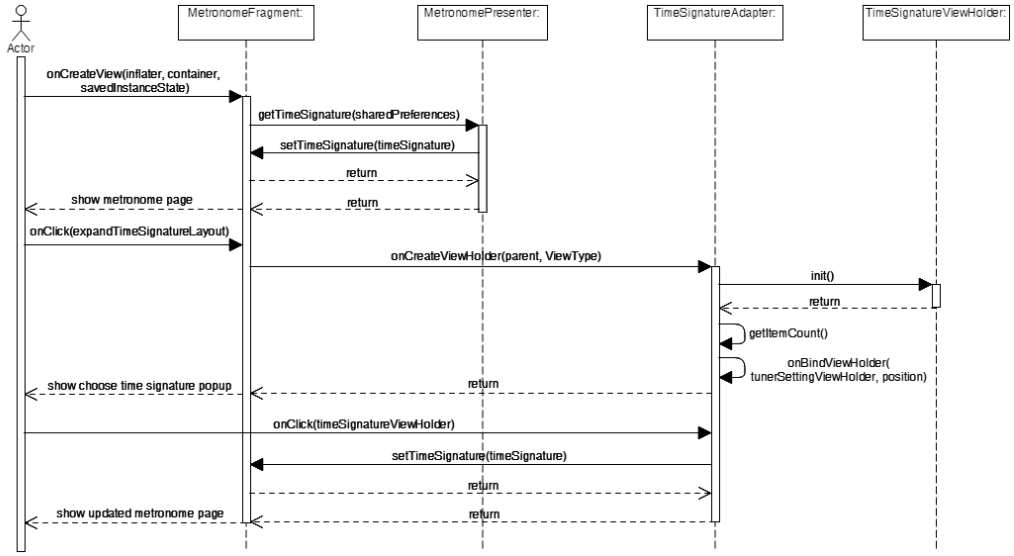
Gambar 4.20 *Sequence* Diagram untuk Menyalakan atau Mematikan *Metronome* Sesuai dengan Tempo dan Jenis Ketukan yang Dipilih Pengguna

9. Mengatur kecepatan ketukan (tempo) *metronome*



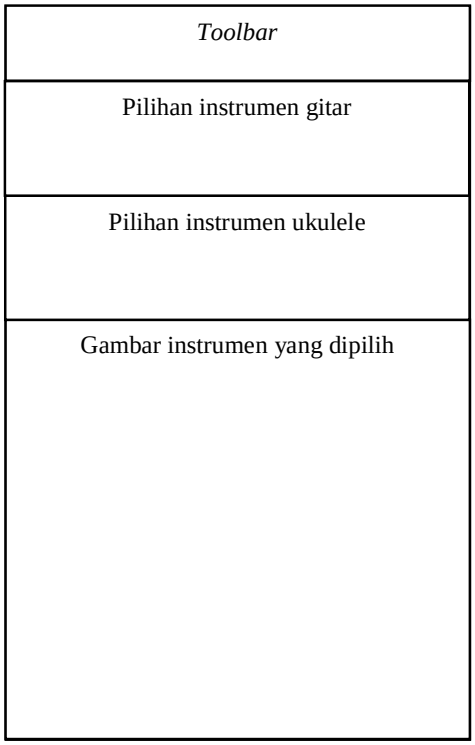
Gambar 4.21 *Sequence* Diagram Mengatur Kecepatan Ketukan *Metronome*

10. Mengatur jenis ketukan yang ingin digunakan saat menggunakan *metronome*

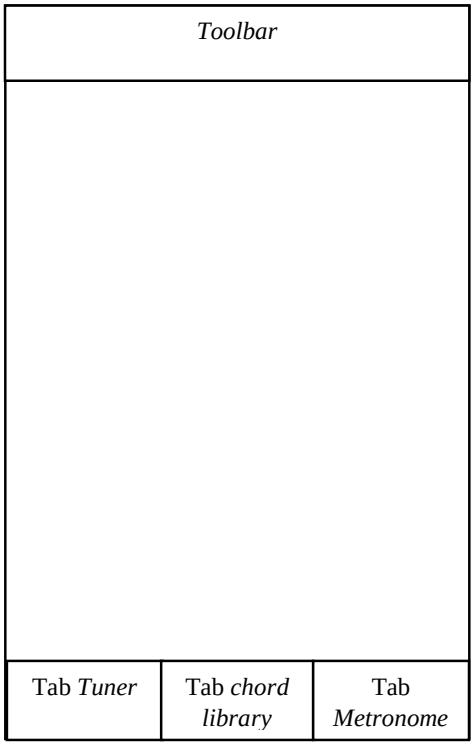


Gambar 4.22 *Sequence* Diagram untuk Mengatur Jenis Ketukan yang Ingin Digunakan saat Menggunakan *Metronome*

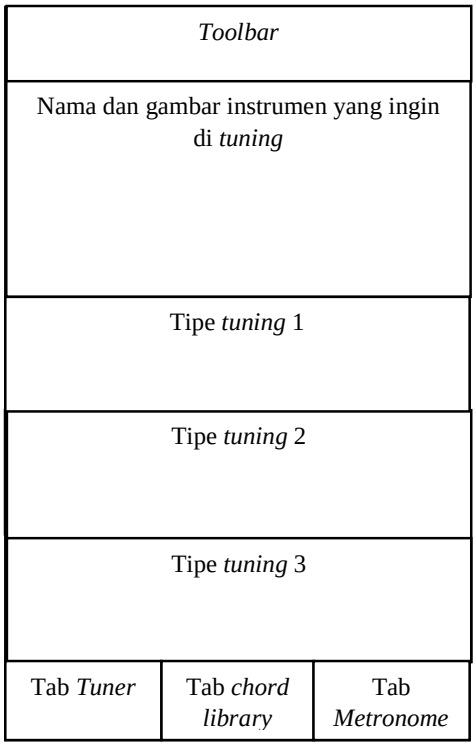
4.3 Rancangan Layar



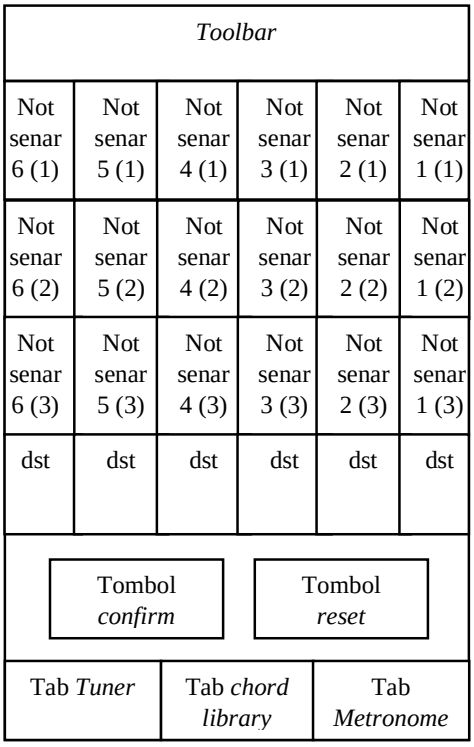
Gambar 4.23 *Wireframe* untuk Menu Pemilihan Instrumen yang Ingin Digunakan



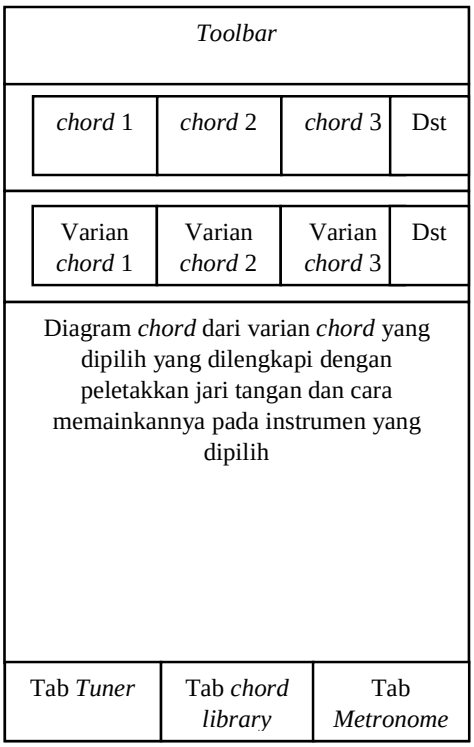
Gambar 4.24 *Wireframe* untuk Fitur *Tuner*



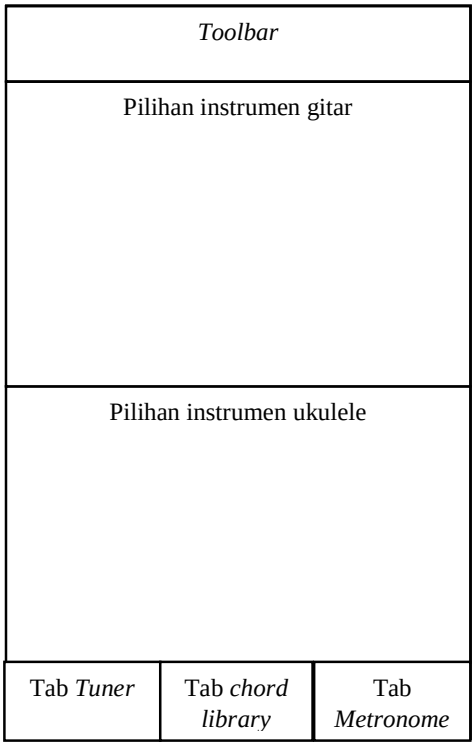
Gambar 4.25 *Wireframe* untuk Menu Pemilihan Tipe *Tuning* yang Ingin Digunakan



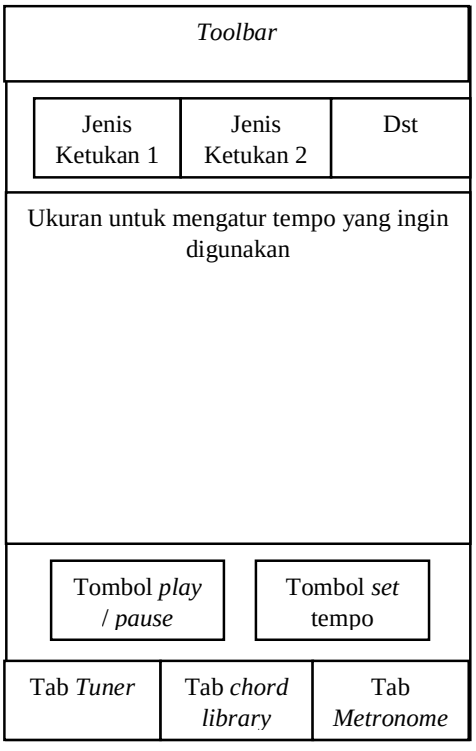
Gambar 4.26 *Wireframe* untuk Menu Pembuatan Tipe *Tuning* Baru



Gambar 4.27 *Wireframe* untuk Fitur *Chord Library*



Gambar 4.28 *Wireframe* Menu Pemilihan Instrumen yang Ingin Di *Tuning* atau yang Ingin Ditampilkan Akordnya



Gambar 4.29 *Wireframe* untuk Fitur *Metronome*

4.4 Implementasi

4.4.1 Spesifikasi yang Digunakan

Pada aplikasi dibuat melalui skripsi ini tentu ada syarat yang harus dipenuhi untuk setiap perangkat keras yang nantinya akan menentukan di perangkat keras mana saja aplikasi yang akan dibuat dapat berjalan. Syarat pertama untuk perangkat keras yang digunakan harus memiliki sistem operasi Android. Syarat kedua adalah perangkat keras harus memenuhi spesifikasi minimal yang diantaranya adalah:

Tabel 4.17 Spesifikasi Minimal Perangkat Keras

Operating System	Android Lollipop 5.1
RAM	2 GB
Storage	40 MB
Processor	Quad-core 1.5 GHz Krait

4.4.2 Dokumentasi Program

Aplikasi Chorus Tuner dibuat dengan menggunakan pola arsitektur MVP (*Model – View – Presenter*) dimana *model* disini merupakan objek yang dierlukan di dalam aplikasi, *view* merupakan komponen pengatur tampilan seperti *activity* dan *fragment*, dan *presenter* merupakan komponen yang mengatur semua logika yang ingin digunakan untuk mengatur tampilan berdasarkan input yang diberikan pengguna pada tampilan dan berdasarkan data yang diberikan dari *model*. *Presenter* di sini tidak perlu dibuat ketika elemen *view* tidak melakukan akses terhadap model dan tidak memiliki proses logika. Aplikasi Chorus Tuner memiliki beberapa fitur di dalamnya yang meliputi:

1. Fitur *Tuner*

   Tampilan layar fitur *tuner*:

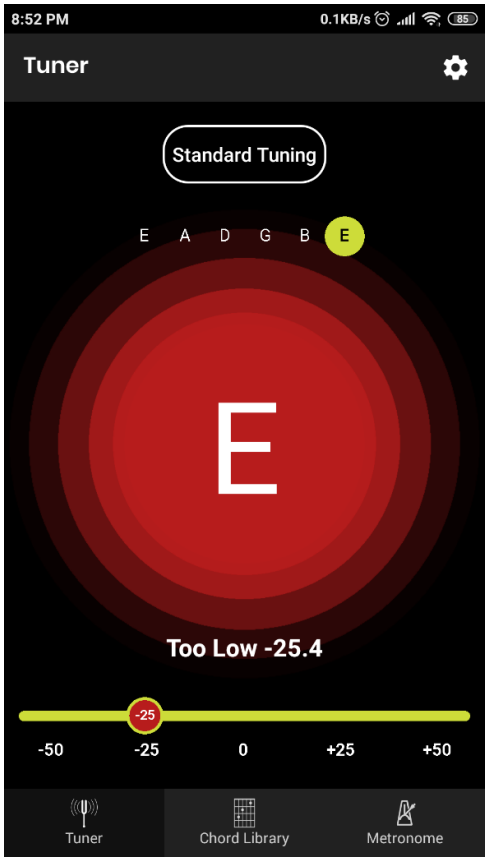
a. Tampilan layar awal untuk instrumen ukulele

Gambar 4.30 Tampilan Layar Halaman *Tuner* Saat Instrumen yang dipilih adalah Ukulele

b. Tampilan layar pada saat *tuning* senar gitar dalam keadaan *in tune*

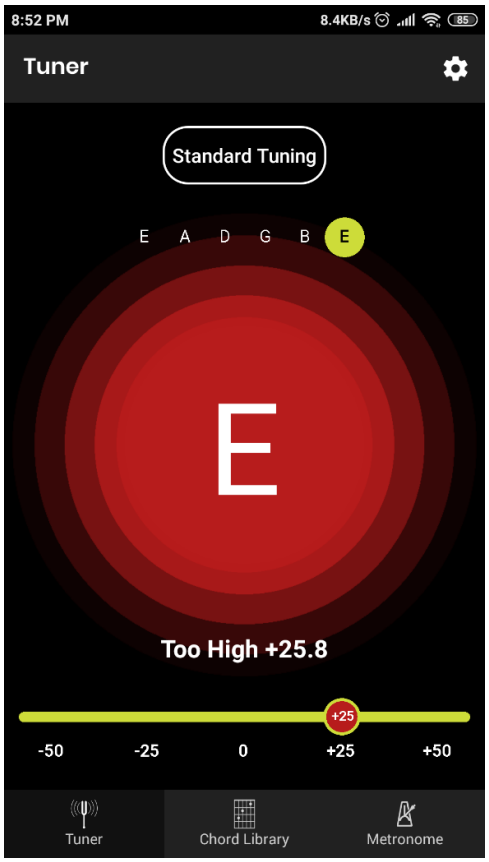
Gambar 4.31 Tampilan Layar Halaman *Tuner* pada Saat *Tuning* Senar Gitar dalam Keadaan *In Tune*

c. Tampilan layar pada saat *tuning* senar gitar dalam keadaan *too low*



Gambar 4.32 Tampilan Layar Halaman *Tuner* pada Saat *Tuning* Senar Gitar dalam Keadaan *Too Low*

d. Tampilan layar pada saat *tuning* senar gitar dalam keadaan *too high*

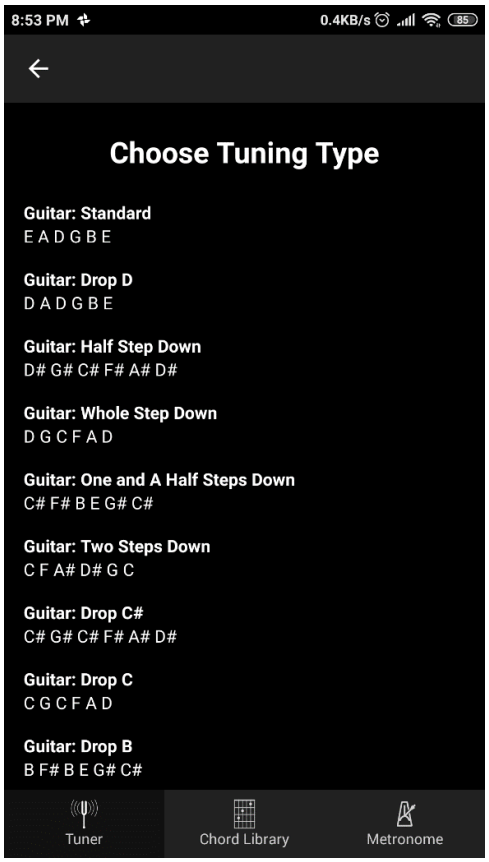


Gambar 4.33 Tampilan Layar Halaman *Tuner* pada Saat *Tuning* Senar Gitar dalam Keadaan *Too High*

e. Tampilan layar halaman *tuner setting*

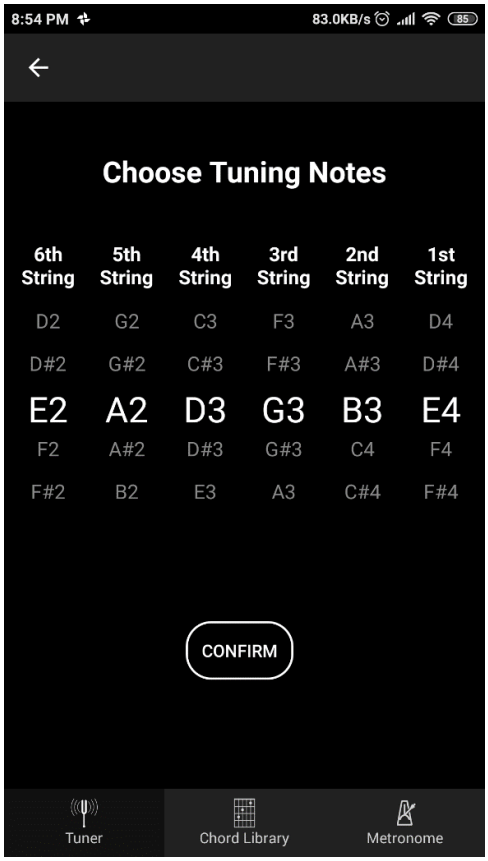
Gambar 4.34 Tampilan Layar Halaman *Tuner Setting*

f. Tampilan layar pemilihan tipe *tuning*



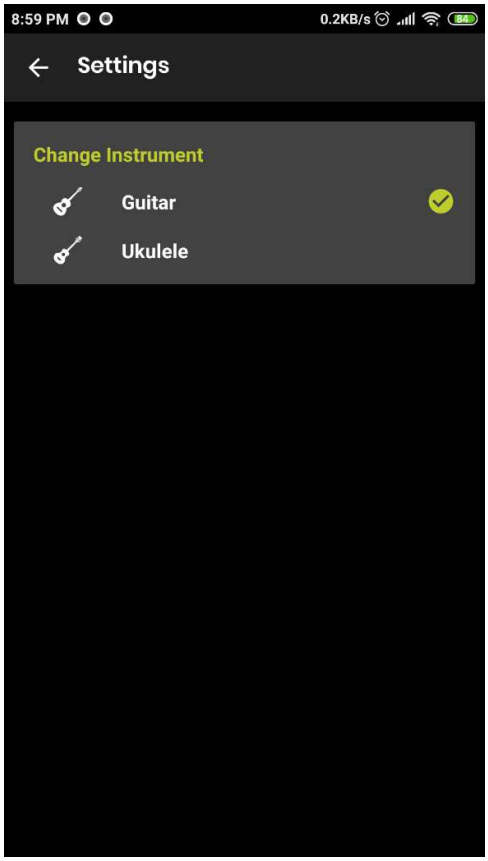
Gambar 4.35 Tampilan Layar Halaman Pemilihan Tipe *Tuning*

g. Tampilan layar pembuatan tipe *tuning* baru

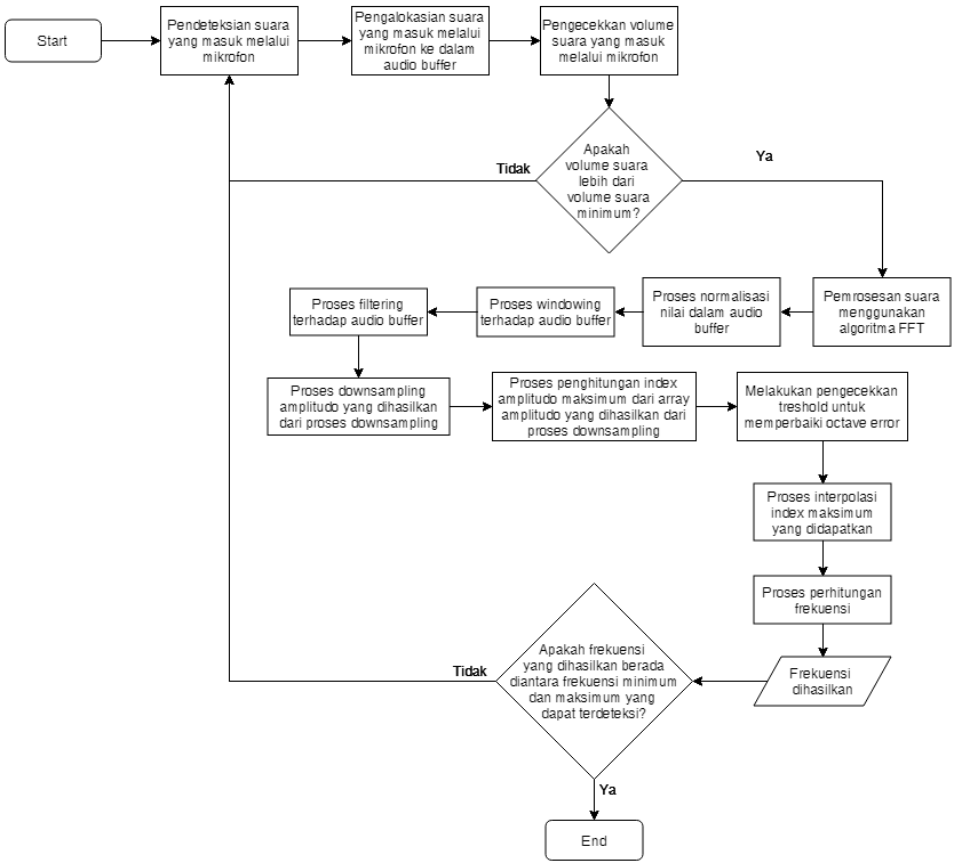


Gambar 4.36 Tampilan Layar Halaman Pembuatan Tipe *Tuning* Baru

h. Tampilan layar menu pemilihan instrumen



Gambar 4.37 Tampilan Layar Menu Pemilihan Instrumen pada Halaman *Settings*



Gambar 4.38 Proses *Tuning* dalam Chorus Tuner

Proses *Tuning* dalam fitur *tuner* ini merupakan pengembangan dari proses *tuning* seperti yang dijelaskan oleh Lourde, R. M. dan Saji, A. K. (2009, p.1) dalam artikelnya. Algoritma yang digunakan dalam fitur *tuner* adalah algoritma HPS (*Harmonic Product Spectrum*) yang prosesnya meliputi:

a. Pengalokasian suara yang masuk melalui mikrofon.

   Pada langkah pertama ini dilakukan pengalokasian data suara yang didapatkan melalui mikrofon dengan frekuensi sampel (*sampling rate*) 44100Hz ke dalam *audio buffer* yang merupakan sebuah *array* dengan panjang 22050. Ukuran Panjang *array* didapatkan dari perhitungan ukuran *audio buffer* minimal yang dibutuhkan untuk meletakkan data yang didapatkan dengan frekuensi sampel (*sampling rate*) 44100Hz, channel recorder mono, dan audio encoding PCM (Pulse-Code Modulation) 16 bit yang kemudian dikalikan dengan 6 supaya waktu perekaman suara menjadi lebih lama sehingga data suara yang ada di

dalam *audio buffer* cukup banyak untuk menghasilkan *frekuensi* yang akurat. Data suara yang dimasukkan ke dalam *audio buffer* merupakan data amplitudo yang dihasilkan oleh suara sehingga *audio buffer* berisi nilai amplitudo dari suara yang terdengar melalui mikrofon.

b. Proses pengecekkan volume suara yang masuk ke dalam mikrofon.

   Proses pengecekkan volume suara dilakukan dengan tujuan untuk menyaring suara yang memiliki volume yang kecil yang dapat mengganggu proses perhitungan frekuensi (*noise*). Pengecekkan volume suara dilakukan dengan cara menentukan *threshold* amplitudo minimal dari suara di dalam *audio buffer* yang ingin dihitung frekuensinya. *Treshold* minimal yang digunakan oleh penulis adalah 550. Kemudian proses dilanjutkan dengan menjumlahkan semua amplitudo dari data suara yang ada di dalam *audio buffer* kemudian hasil penjumlahan tersebut dibagi dengan amplitudo *array audio buffer* untuk mendapatkan volume dari suara yang terdengar pada saat itu. Setelah itu volume dari suara yang terdengar pada saat itu dibandingkan dengan *threshold* amplitudo minimal, jika volume dari suara yang terdengar pada saat itu kurang dari *threshold* amplitudo minimal maka suara tersebut tidak dapat dihitung frekuensinya dan dianggap sebagai *noise*, namun jika volume dari suara yang terdengar pada saat itu lebih dari *threshold* amplitudo minimal maka suara tersebut akan melalui proses selanjutnya dan dihitung frekuensinya.

c. Pemrosesan signal yang didapatkan menggunakan algoritma FFT (*Fast Fourier Transform*).

   Untuk pemrosesan signal menggunakan algoritma FFT, penulis menggunakan *library* dengan nama JTransform yang dapat mengubah *array* dari *audio buffer* yang berisi amplitudo menjadi satu *array* FFT dimana *array* dengan index ganjil berisi bilangan real dan array dengan index genap berisi bilangan imajiner. Dari *array* FFT yang ada maka dapat dihasilkan satu *array* yang berisi amplitudo yang sudah melalui perhitungan FFT dengan panjang array setengah dari panjang array *audio buffer*. Untuk mengisi array amplitudo yang telah melalui proses FFT dilakukan perhitungan $\sqrt{real^2 + imajiner^2}$ sebanyak

panjang array amplitudo yang telah melalui proses FFT. Pada perhitungan tersebut real merupakan bilangan real dari array FFT dan imajiner merupakan bilangan imajiner dari array FFT.

d. Proses normalisasi nilai dalam *audio buffer*.

Proses normalisasi ini bertujuan agar nilai amplitudo yang ada di dalam *audio buffer* berada pada Batasan antara -1 dan 1. Proses normalisasi ini dilakukan dengan cara membagi setiap nilai amplitudo di dalam *audio buffer* dengan nilai amplitudo maksimum yang ada di dalam *audio buffer* tersebut.

e. Proses *windowing* dan *filtering* menggunakan *hamming window* dan *band pass filter*.

Proses *windowing* dan *filtering* terhadap amplitudo yang ada di dalam *array audio buffer* dilakukan dengan tujuan untuk mengurangi gangguan (*noise*) yang ikut masuk ke dalam *audio buffer* melalui mikrofon. Proses *windowing* dilakukan menggunakan *hamming window* yang dilakukan dengan cara mengalikan setiap *audio buffer* dengan nilai *hamming* dari *audio buffer* tersebut yang didapatkan dengan rumus (1). Proses *filtering* dilakukan menggunakan *band pass filter* yang dilakukan dengan cara mengalikan setiap *audio buffer* yang sudah melalui proses *windowing* dengan nilai *band pass* dari *audio buffer* tersebut yang didapatkan dari rumus (2) dan rumus (3). Rumus yang digunakan untuk mendapatkan nilai *hamming* dan nilai *band pass* dari *audio buffer* pada index tertentu adalah sebagai berikut (Greensted, A., 2010, http://www.labbookpages.co.uk/audio/firWindowing.html):

- Rumus untuk mendapatkan nilai *hamming*

$$hamming\ value = 0.54 - 0.46\ cos\left(\frac{2\pi n}{M}\right) \quad (1)$$

- Rumus untuk mendapatkan nilai *band pass* ketika $\frac{M}{2} = n$

$$bandpass\ value = 2\ (ft2 - ft1) \quad (2)$$

- Rumus untuk mendapatkan nilai *band pass* ketika $\frac{M}{2} \neq n$

$$bandpass\ value = \frac{\sin\left[2\pi ft2\left(n-\frac{M}{2}\right)\right] - \sin\left[2\pi ft1\left(n-\frac{M}{2}\right)\right]}{\pi\left(n-\frac{M}{2}\right)} \quad (3)$$

Pada rumus diatas n adalah index dari *audio buffer* yang ditunjuk, M adalah panjang dari *array audio buffer* dikurang 1, ft2 adalah frekuensi maksimal yang kita tentukan yang telah di normalisasi dengan cara dibagi dengan frekuensi sampel, dan ft1 adalah frekuensi minimal yang kita tentukan yang telah di normalisasi dengan cara dibagi dengan frekuensi sampel.

f. Proses *downsampling* amplitudo yang dihasilkan dari algoritma FFT (Fast Fourier Transform).

   Langkah berikutnya adalah melakukan *downsampling* terhadap *array* amplitudo yang telah melalui proses FFT dengan cara mengalikan setiap amplitudo di dalam *array* amplitudo yang telah melalui proses FFT dengan nilai amplitudo yang telah melalui proses kompresi. Proses *downsampling* dilakukan sebanyak 2 kali yaitu proses *downsampling* dengan nilai kompresi 2 dan proses *downsampling* dengan nilai kompresi 3. Proses *downsampling* ini bertujuan untuk mendapatkan satu amplitudo yang kuat pada satu index tertentu yang nantinya akan digunakan untuk menghitung frekuensi akhir yang dihasilkan.

g. Proses penghitungan index amplitudo maksimum dari *array* amplitudo yang dihasilkan dari proses *downsampling*

   Proses kemudian dilanjutkan dengan mencari index amplitudo maksimum dari *array* amplitudo yang dihasilkan dari proses *downsampling*.

h. Proses perbaikkan *octave error* yang mengakibatkan salah deteksi frekuensi pada *tuner*

   Proses perbaikkan *octave error* ini diterapkan dengan menggunakan aturan yang dituliskan oleh Smyth, T. (2015, http://musicweb.ucsd.edu/~trsmyth/analysis/Harmonic_Product_Spectrum.html) yaitu ketika amplitudo maksimum kedua yang terletak pada posisi sebelum amplitudo maksimum yang didapatkan memiliki nilai setengah dari amplitudo maksimum dan rasio amplitudo (4) berada diatas *threshold* (misalnya 0,2 untuk gelombang dengan 5 harmonik), maka index amplitudo maksimum yang digunakan untuk perhitungan selanjutnya adalah index dari amplitudo maksimum kedua yang

terletak pada posisi sebelum amplitudo maksimum. Rumus yang digunakan untuk menghitung rasio dari amplitudo adalah sebagai berikut:

$$rasio\ amplitudo = \frac{amplitudo\ maksimum\ kedua}{amplitudo\ maksimum} \quad (4)$$

i. Proses interpolasi index maksimum yang didapatkan.

Setelah index dari amplitudo maksimum ditemukan maka index tersebut akan melalui proses interpolasi untuk mendapatkan frekuensi yang lebih akurat (5) dan (6) (Smith, J. O., 2011, https://ccrma.stanford.edu/~jos/sasp/Quadratic_Interpolation_Spectral_Peaks.html):

$$index\ terinterpolasi = index\ amplitudo\ maksimum + shift \quad (5)$$

$$shift = \frac{0.5\ (left - right)}{(2*left - 2*mid + right)} \quad (6)$$

Pada rumus diatas *left* adalah nilai amplitudo pada index sebelum amplitudo maksimum, *right* adalah nilai amplitudo pada index setelah amplitudo maksimum, dan *mid* adalah nilai amplitudo maksimum.

j. Proses penghitungan frekuensi.

Proses terakhir adalah proses perhitungan frekuensi yang dihitung menggunakan rumus:

$$frekuensi = \frac{frekuensi\ sampel * index\ terinterpolasi}{panjang\ array\ audio\ buffer} \quad (7)$$

Pada proses *windowing* dan *filtering* diatas penulis memilih untuk menggunakan *hamming window* dan *bandpass filter* sesuai dengan jenis *window* dan *filter* yang digunakan oleh Lourde, R. M. dan Saji, A. K. (2009, p.1) dalam artikelnya. Setelah penulis melakukan uji coba penggunaan jenis *window* dan *filter* yang lain, penulis semakin yakin untuk menggunakan *hamming window* dan *bandpass filter* karena kombinasi jenis *window* dan *filter* ini menghasilkan frekuensi yang paling akurat jika dibandingkan dengan kombinasi jenis *window* dan *filter* lainnya. Hasil dari uji coba yang dilakukan penulis adalah sebagai berikut:

Tabel 4.18 Hasil Uji Coba Kombinasi Berbagai Jenis *Window* dan *Filter*

Filter	***Window***	***String Frequency* (Hz)**					
		6	**5**	**4**	**3**	**2**	**1**
Low Pass Filter	*Rectangular*	82	110.22	146	194	246	328
	Bartlett	82	110.04	146.05	194.04	246	986
	Hanning	NaN	NaN	NaN	NaN	NaN	NaN
	Hamming	NaN	NaN	NaN	194	246	330
	Blackman	NaN	NaN	NaN	NaN	NaN	NaN
High Pass Filter	*Rectangular*	82.09	110.04	146.08	194.85	246.04	328.12
	Bartlett	82.09	110.01	146.07	194.57	246.48	328.03
	Hanning	NaN	NaN	NaN	NaN	494	658
	Hamming	82.56	110.44	146.11	194.28	246.24	328.38
	Blackman	NaN	NaN	NaN	NaN	NaN	6636
Band Pass Filter	*Rectangular*	82.36	110.05	146.07	194.35	246.22	328.22
	Bartlett	165.99	110.04	146.04	194.66	246.56	328.1
	Hanning	NaN	NaN	NaN	5366	6586	988.88
	Hamming	82.46	110.05	146.51	195.9	246.84	329.28
	Blackman	NaN	NaN	NaN	NaN	NaN	6974
Band Stop Filter	*Rectangular*	166.99	110.03	146.19	194.22	246.11	328.19
	Bartlett	165.99	110.07	146.07	194.04	246.06	328.06
	Hanning	NaN	NaN	NaN	5110	494	328
	Hamming	82.16	111.94	146.36	194.25	247.69	328.7
	Blackman	NaN	NaN	NaN	NaN	NaN	6636

Selain melakukan pengujian kombinasi jenis *window* dan *filter*, penulis juga melakukan uji coba penggunaan frekuensi sampel yang berbeda-beda guna mendapatkan keakuratan deteksi frekuensi yang maksimal. Pengujian ini dilakukan setelah mendapatkan kombinasi jenis *window* dan *filter* yang tepat yaitu *hamming window* dan *bandpass filter*. Melalui pengujian ini didapatkan temuan bahwa frekuensi sampel 44100 Hz menghasilkan keakuratan deteksi frekuensi yang lebih tinggi daripada frekuensi sampel yang lain. Hasil dari uji coba yang dilakukan penulis adalah sebagai berikut:

Tabel 4.19 Hasil Uji Coba Frekuensi Sampel (*Sampling Rate*) yang Berbeda

Sampling Rate	***String Frequency* (Hz)**					
	6	**5**	**4**	**3**	**2**	**1**
8000 Hz	166	110.07	146.01	195.98	247.82	329.97
44100 Hz	82.46	110.05	146.51	195.9	246.84	329.28
48000 Hz	166	110.01	147.68	195.87	247.93	329.97

Pada fitur *tuner* disediakan fitur untuk mengganti tipe *tuning* yang ingin digunakan oleh pengguna. Pengguna dapat mengakses fitur ini dengan cara menekan tombol tipe *tuning* yang ada pada halaman *tuner* kemudian memilih pilihan pemilihan tipe *tuning* pada halaman *tuner setting.* Pada halaman pemilihan tipe *tuning* pengguna akan memilih tipe *tuning* yang ingin digunakannya untuk melakukan *tuning* terhadap instrumennya. Setelah memilih not yang ada maka pengguna akan dikembalikan ke halaman *tuner.* Pada saat pengguna kembali ke halaman *tuner* maka pengguna akan melihat perubahan pada tulisan tombol tipe *tuning* yang akan berganti dengan nama dari tipe *tuning* yang dipilih dan perubahan pada not yang digunakan pada tiap senarnya sesuai dengan not dari tipe *tuning* yang dipilih oleh pengguna.

Selain itu pada fitur *tuner* disediakan juga fitur untuk membuat tipe *tuning* baru jika pengguna tidak menemukan tipe *tuning* yang sesuai dengan keinginannya. Fitur ini dapat diakses dengan cara menekan tombol tipe *tuning* yang ada pada halaman *tuner* kemudian memilih pilihan pembuatan tipe *tuning* baru pada halaman *tuner setting.* Pada halaman pembuatan tipe *tuning* baru pengguna akan memilih not yang ingin digunakan pada tiap senar instrumen. Setelah memilih not yang ada maka pengguna dapat menekan tombol *confirm* untuk membuat tipe *tuning* baru dan pengguna akan dikembalikan ke halaman *tuner.* Pada saat pengguna kembali ke halaman *tuner* maka pengguna akan melihat perubahan pada tulisan tombol tipe *tuning* yang akan berganti dengan tulisan "*custom tuning*" dan perubahan pada not yang digunakan pada tiap senarnya sesuai dengan not yang dipilih oleh pengguna.

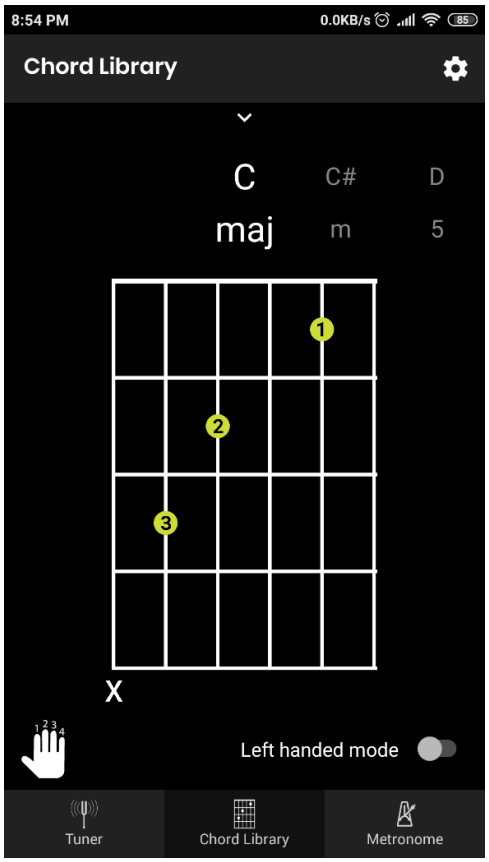
Fitur terakhir yang disediakan pada fitur *tuner* adalah fitur untuk mengganti instrumen yang ingin di *tuning.* Fitur ini dibuat dengan tujuan

supaya pengguna dapat melakukan *tuning* terhadap instrumen lain selain gitar. Saat ini jenis instrumen yang tersedia hanya ada ukulele dan gitar jadi sementara ini pengguna hanya bisa memilih diantara kedua instrumen itu saja. Menu untuk mengganti instrumen dapat diakses pada halaman *settings* yang dapat dibuka dengan menekan tombol *settings* yang ada di pojok kanan atas. Setelah pengguna melakukan penggantian instrumen maka seluruh informasi *tuning* yang ada di halaman *tuner* akan disesuaikan dengan instrumen yang dipilih mulai dari tipe *tuning* nya, jumlah senarnya dan not senarnya semua akan mengikuti instrumen yang dipilih.

2. Fitur *Chord Library*

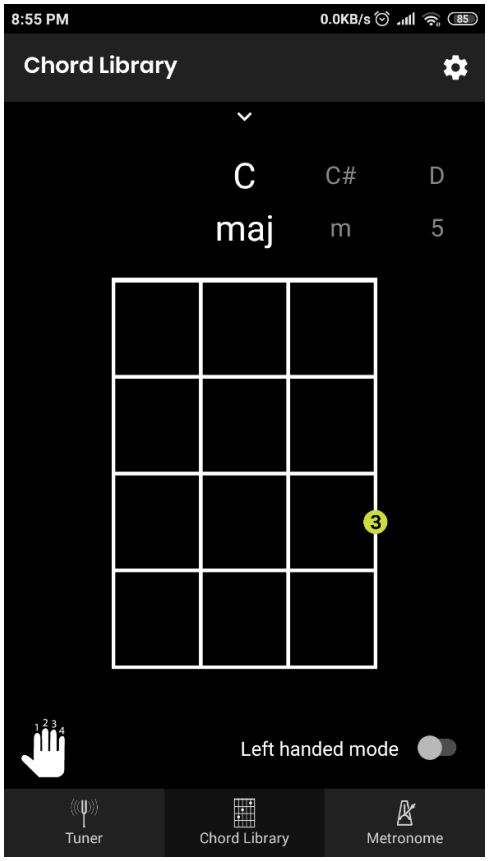
   Tampilan layar fitur *chord library*:

   a. Tampilan layar saat instrumen yang dipilih adalah gitar



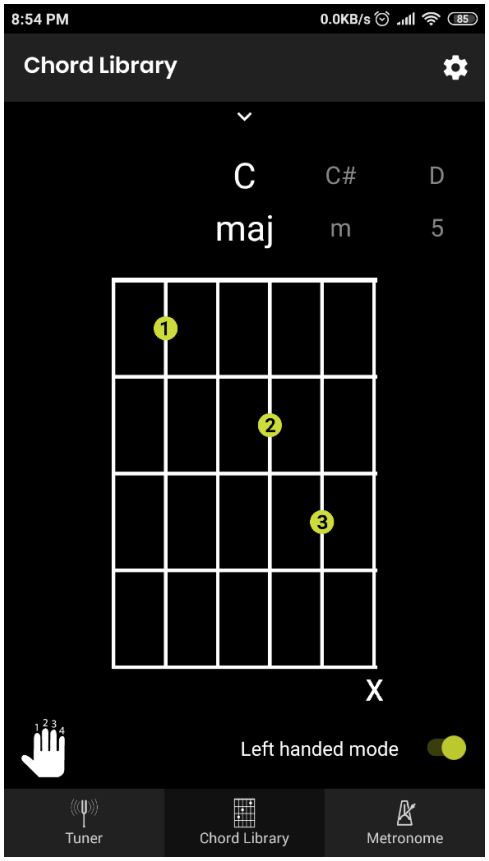
Gambar 4.39 Tampilan Layar Halaman *Chord Library* saat Instrumen yang Dipilih adalah Gitar

b. Tampilan layar saat instrumen yang dipilih adalah ukulele

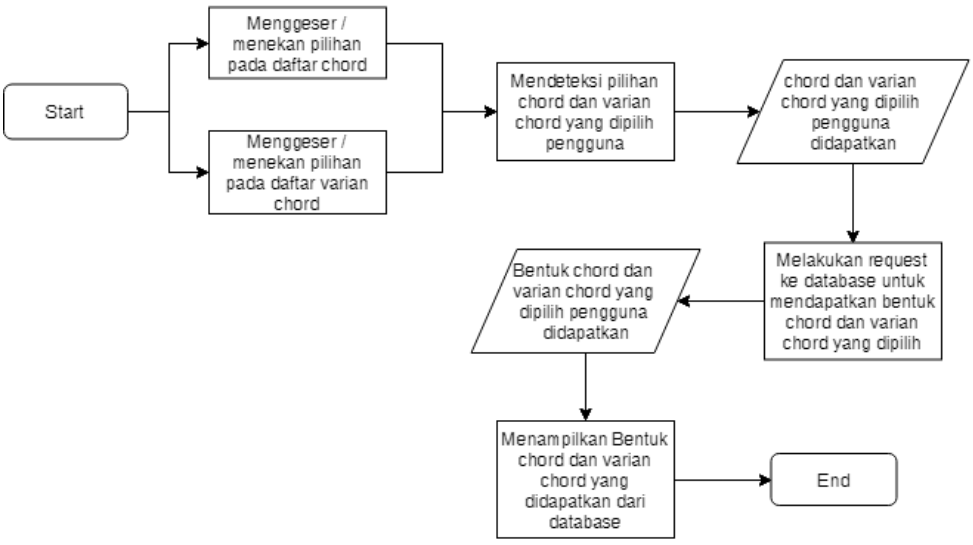


Gambar 4.40 Tampilan Layar Halaman *Chord Library* saat Instrumen yang Dipilih adalah Ukulele

c. Tampilan layar saat mode *left handed* aktif



Gambar 4.41 Tampilan Layar Halaman *Chord Library* saat Mode *Left Handed* Aktif



Gambar 4.42 Proses Menampilkan Diagram *Chord* pada Chorus Tuner

Cara untuk menampilkan diagram *chord* pada fitur *chord library* adalah:

a. Pengguna melakukan penggeseran terhadap daftar pilihan *chord* atau daftar varian *chord* yang ada.
b. Saat penggeseran berhenti pada salah satu *chord* atau varian *chord* atau ketika pengguna menekan salah satu pilihan *chord* atau varian *chord* yang ada maka aplikasi akan mendeteksi *chord* dengan varian *chord* apa yang ditunjuk sesuai dengan posisi dari *chord* atau carian *chord* yang ditunjuk.
c. Kemudian aplikasi akan mengakses *database* internal yang dibuat menggunakan *realm* yang tersimpan pada perangkat pengguna untuk mencari *chord* dengan varian *chord* yang ditunjuk. *Database* internal di perangkat milik pengguna akan ada dan terisi pada saat pengguna melakukan instalasi aplikasi pertama kali.
d. Setelah *chord* ditemukan maka aplikasi akan menggambar ulang diagram *chord* sesuai dengan bentuk *chord* yang didapatkan dari *database* internal yang dibuat menggunakan *realm* yang tersimpan pada perangkat pengguna. Bentuk chord didapatkan dalam bentuk JSON (*JavaScript Object Notation*) yang nantinya akan digunakan untuk mengisi variable pada *class shape* dan akan digunakan untuk melakukan *update* terhadap diagram *chord.*

Pada fitur *chord library* disediakan fitur untuk mengganti instrumen yang ingin ditampilkan diagram *chord* nya. Fitur ini dibuat dengan tujuan supaya pengguna bisa melihat bentuk diagram *chord* pada instrumen lain. Saat ini jenis instrumen yang tersedia hanya ada ukulele dan gitar jadi sementara ini pengguna hanya bisa memilih diantara kedua instrumen itu saja. Menu untuk mengganti instrumen dapat diakses pada halaman *settings* yang dapat dibuka dengan menekan tombol *settings* yang ada di pojok kanan atas. Setelah pengguna melakukan penggantian instrumen maka diagram *chord* akan menyesuaikan tampilan dan bentuk *chord* yang ditampilkan dengan bentuk *chord* dari instrumen yang dipilih.

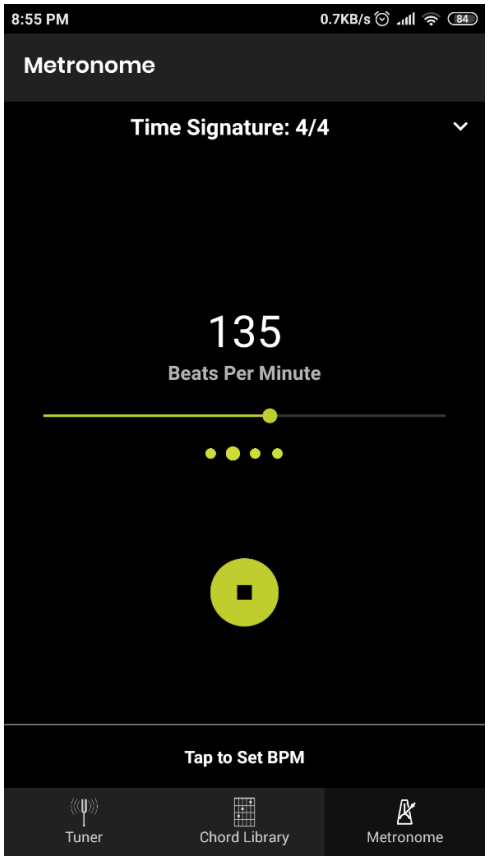
Selain fitur untuk mengganti instrumen yang ingin digunakan disediakan juga fitur untuk mengganti mode menjadi mode *left handed* untuk menampilkan diagram *chord* jika pengguna menggunakan tangan yang terbalik untuk memainkan gitar (kidal). Untuk mengaktifkan mode *left*

handed pengguna cukup menekan tombol *left handed mode* yang ada di bawah diagram *chord.* Setelah pengguna menekan tombol *left handed mode,* jika mode *left handed* dalam diaktifkan maka diagram *chord* akan ditampilkan dengan arah baca kanan ke kiri, sebaliknya jika mode *left handed* dimatikan maka diagram *chord* akan ditampilkan dengan arah baca kiri ke kanan.

3. Fitur *Metronome*

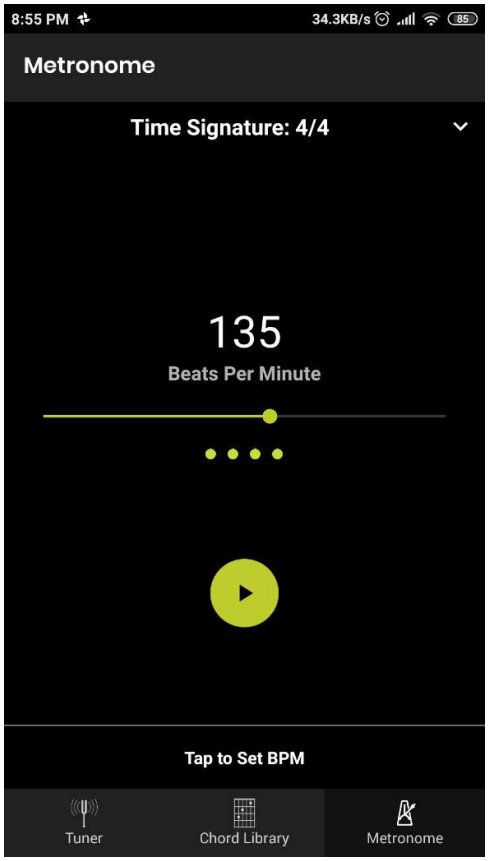
   Tampilan layar fitur *metronome*:

   a. Tampilan layar saat *metronome* dimainkan



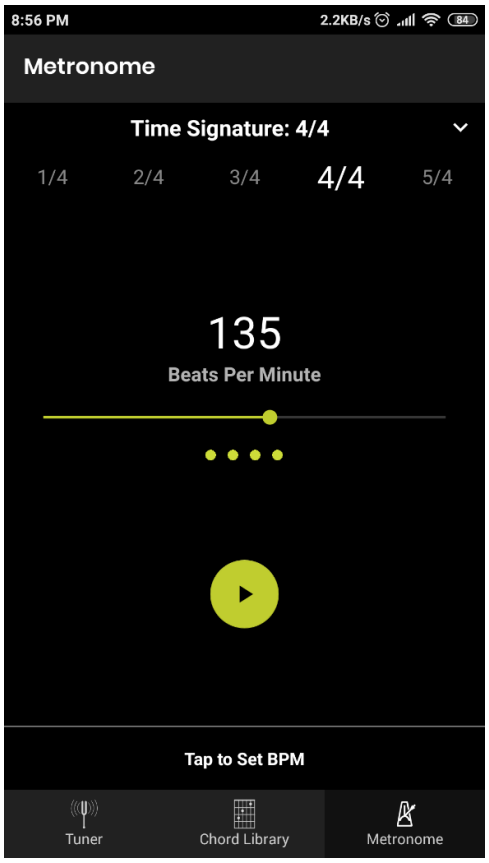
Gambar 4.43 Tampilan Layar Halaman *Metronome* saat *Metronome* Dimainkan

b. Tampilan layar saat *metronome* dihentikan

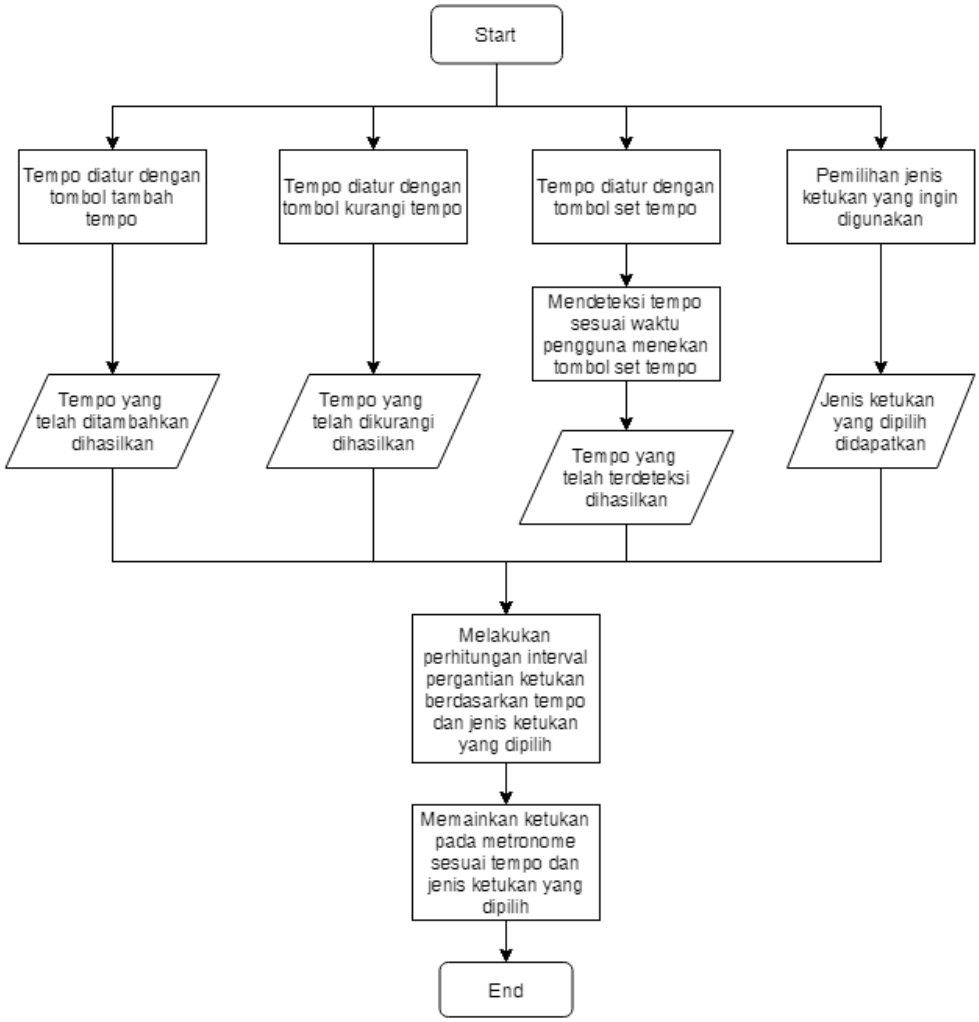


Gambar 4.44 Tampilan Layar Halaman *Metronome* saat *Metronome* Dihentikan

c. Tampilan layar saat menu pemilihan jenis ketukan dibuka



Gambar 4.45 Tampilan Layar Halaman *Metronome* saat Menu Pemilihan Jenis Ketukan Dibuka



Gambar 4.46 Proses Pengaturan dan Pengaktifan *Metronome* dalam Chorus Tuner

Cara untuk mengatur dan mengaktifkan *metronome*:

a. Pengguna melakukan pengaturan tempo yang ingin digunakan dengan cara melakukan pengaturan pada *slider* tempo atau menekan tombol pengaturan tempo untuk mendapatkan tempo sesuai dengan kecepatan pengguna menekan tombol. Perhitungan yang digunakan untuk mendeteksi tempo dari kecepatan pengguna menekan tombol adalah sebagai berikut:

$$waktu = waktu\ sekarang - waktu\ terakhir \quad (7)$$

$$tempo - \frac{60000}{waktu} \quad (8)$$

Pada rumus diatas waktu sekarang merupakan waktu pengguna memencet tombol saat itu, waktu terakhir adalah waktu terakhir pengguna memencet tombol sebelumnya, dan waktu merupakan interval waktu antara waktu sekarang dan waktu terakhir.

b. Pengguna mengatur jenis ketukan yang ingin digunakannya dengan cara membuka menu yang menampilkan jenis ketukan yang ada dan memilih salah satu jenis ketukan.
c. Setelah pengguna mengatur jenis ketukan maka tampilan indikator *metronome* akan berubah dimana jumlah titik *metronome* akan menyesuaikan dengan jumlah ketukan yang ada pada jenis ketukan.
d. Pengguna menekan tombol *play/stop metronome*, ketika simbol yang ada di tombol adalah simbol *play* maka *metronome* akan menyala dan ketika simbol yang ada di tombol adalah simbol *stop* maka *metronome* akan mati. Ketika *metronome* menyala maka aplikasi akan menyalakan suara dan menjalankan indikator pada *metronome* sesuai dengan jenis ketukan yang dipilih pengguna dan tempo yang telah ditentukan oleh pengguna yang kemudian dihitung menjadi satuan detik. Dari detik yang didapatkan maka suara dan indikator *metronome* akan berjalan dengan interval sesuai dengan detik yang didapatkan. Rumus yang digunakan untuk mendapatkan detik dari jenis ketukan yang dipilih pengguna dan tempo yang telah ditentukan oleh pengguna adalah:

$$delay = \frac{60000}{tempo * dividing\ factor} \quad (9)$$

Pada rumus diatas, tempo merupakan tempo yang ditentukan oleh pengguna dan *dividing factor* merupakan angka penentu yang nilainya sama dengan 1 jika tipe not yang digunakan adalah not seperempat dan 2 jika tipe not yang digunakan adalah not seperdelapan.

4.5 *Testing*

4.5.1 *Black-Box Testing*

Tabel 4.20 *Test Case* Fitur *Tuner*

No	*Test Case*	*Action*	*Expected Result*	*Result*
1	*Status text* dan indikator *tuner* akan menunjukkan status "*In Tune*" ketika *tuning* instrumen sudah pas	Membuat *tuning* pada instrumen menjadi tepat dan memetiknya	Pengguna dapat melihat tulisan "*In Tune*" pada *status text* dan melihat indikator *tuner* berada pada posisi garis yang tepat dan berubah warna menjadi hijau	**Berhasil**
2	*Status text* dan indikator *tuner* akan menunjukkan status "*Too Low*" ketika *tuning* instrumen terlalu rendah	Mengecilkan *tuning* pada instrumen dan memetiknya	Pengguna dapat melihat tulisan "*Too Low*" pada *status text* dan melihat indikator *tuner* berada pada posisi kurang dari garis yang tepat dan berubah warna menjadi merah	**Berhasil**
3	*Status text* dan indikator *tuner* akan menunjukkan status "*Too High*" ketika *tuning* instrumen	Membesarkan *tuning* pada instrumen dan memetiknya	Pengguna dapat melihat tulisan "*Too High*" pada *status text* dan melihat indikator *tuner* berada pada posisi lebih dari garis yang	**Berhasil**

No	*Test Case*	*Action*	*Expected Result*	*Result*
	terlalu tinggi		tepat dan berubah warna menjadi merah	
4	Indikator nada senar akan berganti sesuai dengan senar yang dipetik	Memetik beberapa senar yang berbeda pada instrumen	Ketika pengguna memetik senar 6 maka pengguna dapat melihat indikator nada senar menunjukkan nada senar 6, begitu pula dengan senar lainnya	**Berhasil**
5	Membuka halaman *tuner setting*	Menekan tombol tipe *tuning* pada halaman *tuner*	Pengguna dapat melihat halaman *tuner setting*	**Berhasil**
6	Membuka halaman daftar tipe *tuning*	Menekan menu pemilihan tipe *tuning* pada halaman *tuner setting*	Pengguna dapat melihat halaman daftar tipe *tuning*	**Berhasil**
7	Kembali ke halaman *tuner setting* dari halaman penggantian tipe *tuning*	Menekan tombol *back* pada halaman penggantian tipe *tuning*	Pengguna dapat melihat halaman *tuner setting* kembali	**Berhasil**
8	Membuka halaman pembuatan tipe	Menekan menu pembuatan tipe *tuning* baru	Pengguna dapat melihat halaman pembuatan tipe	**Berhasil**

No	*Test Case*	*Action*	*Expected Result*	*Result*
	tuning baru	pada halaman *tuner setting*	*tuning* baru	
9	Kembali ke halaman *tuner* dari halaman *tuner setting*	Menekan tombol *back* pada halaman *tuner setting*	Pengguna dapat melihat halaman *tuner* kembali	**Berhasil**
10	Kembali ke halaman *tuner setting* dari halaman pembuatan tipe *tuning* baru	Menekan tombol *back* pada halaman pembuatan tipe *tuning* baru	Pengguna dapat melihat halaman *tuner setting* kembali	**Berhasil**
11	Kembali ke halaman *tuner* dan mengganti tipe *tuning* yang digunakan di halaman *tuner*	Menekan salah satu pilihan pada daftar tipe *tuning* pada halaman pemilihan tipe *tuning*	Pengguna dapat melihat halaman *tuner* kembali dan melihat tampilan pada halaman *tuner* yang disesuaikan dengan tipe *tuning* yang dipilih pengguna	**Berhasil**
		Menekan tombol *confirm* ketika pengguna telah selesai memilih not yang ingin digunakan pada halaman pembuatan tipe *tuning* baru	Pengguna dapat melihat halaman *tuner* kembali dan melihat tampilan pada halaman *tuner* yang disesuaikan dengan tipe *tuning* yang dibuat pengguna	**Berhasil**

No	*Test Case*	*Action*	*Expected Result*	*Result*
12	Membuka halaman *setting*	Menekan tombol *setting* pada halaman *tuner*	Pengguna dapat melihat halaman *setting* dan instrumen yang saat ini dipilihnya	**Berhasil**
13	Kembali ke halaman *tuner* dari halaman *setting*	Menekan tombol *back* pada halaman *setting*	Pengguna dapat melihat halaman *tuner* kembali	**Berhasil**
14	Kembali ke halaman *tuner* dari halaman *setting* dan mengganti instrumen yang ingin di *tuning* pada halaman *tuner*	Menekan salah satu pilihan pada daftar instrumen	Pengguna dapat melihat halaman *tuner* kembali dan melihat tampilan pada halaman *tuner* yang disesuaikan dengan instrumen yang dipilih pengguna	**Berhasil**

Tabel 4.21 *Test Case* Fitur *Chord Library*

No	*Test Case*	*Action*	*Expected Result*	*Result*
1	Mengganti *chord* yang ingin ditampilkan diagramnya	Melakukan scroll pada daftar *chord* pada halaman *chord library*	Pengguna akan melihat diagram *chord* yang menampilkan *chord* sesuai dengan *chord* yang ditunjukkan oleh tempat pengguna berhenti melakukan *scroll*	**Berhasil**
		Menekan salah satu pilihan *chord* dari daftar *chord* pada halaman *chord library*	Pengguna akan melihat diagram *chord* yang menampilkan *chord* sesuai dengan *chord* yang ditekan oleh pengguna	**Berhasil**
2	Mengganti varian dari *chord* yang ingin ditampilkan diagramnya	Melakukan *scroll* pada daftar varian *chord* pada halaman *chord library*	Pengguna akan melihat diagram *chord* yang menampilkan varian *chord* sesuai dengan varian *chord* yang ditunjukkan oleh tempat pengguna berhenti	**Berhasil**

No	*Test Case*	*Action*	*Expected Result*	*Result*
			melakukan *scroll*	
		Menekan salah satu pilihan varian *chord* dari daftar varian *chord* pada halaman *chord library*	Pengguna akan melihat diagram *chord* yang menampilkan varian *chord* sesuai dengan varian *chord* yang ditekan oleh pengguna	**Berhasil**
3	Mengganti mode tampilan diagram *chord* menjadi *left handed mode*	Menekan *switch left handed mode* saat *left handed mode* sedang tidak aktif	Pengguna akan melihat diagram *chord* merubah posisi baca yang tadinya dari kanan ke kiri menjadi kiri ke kanan	**Berhasil**
4	Mengembalikan mode tampilan diagram *chord* dari *left handed mode* ke mode asal	Menekan *switch left handed mode* saat *left handed mode* sedang aktif	Pengguna akan melihat diagram *chord* merubah posisi baca yang tadinya dari kiri ke kanan menjadi kanan ke kiri	**Berhasil**
5	Mengganti status aktif pada *switch left handed mode* menjadi aktif	Menekan *switch left handed mode* saat *left handed mode sedang tidak*	Pengguna akan melihat posisi dan tampilan *switch* menjadi aktif	**Berhasil**

No	*Test Case*	*Action*	*Expected Result*	*Result*
		aktif		
6	Mengganti status aktif pada switch *left handed mode* menjadi tidak aktif	Menekan *switch left handed mode* saat *left handed mode* sedang aktif	Pengguna akan melihat posisi dan tampilan *switch* menjadi tidak aktif	**Berhasil**
7	Membuka halaman *setting*	Menekan tombol *setting* pada halaman *chord library*	Pengguna dapat melihat halaman *setting* dan instrumen yang saat ini dipilihnya	**Berhasil**
8	Kembali ke halaman *chord library* dari halaman *setting*	Menekan tombol back pada halaman *setting*	Pengguna dapat melihat halaman *chord library* kembali	**Berhasil**
9	Kembali ke halaman *chord library* dari halaman *setting* dan mengganti instrumen yang ingin di tampilkan diagram *chord* nya pada halaman *chord library*	Menekan salah satu pilihan pada daftar instrumen	Pengguna dapat melihat halaman *chord library* kembali dan melihat tampilan pada halaman *chord library* yang disesuaikan dengan instrumen yang dipilih pengguna	**Berhasil**

Tabel 4.22 *Test Case* Fitur *Metronome*

No	*Test Case*	*Action*	*Expected Result*	*Result*
1	Memperlambat tempo suara yang dihasilkan	Menekan tombol untuk mengurangi tempo	Pengguna akan mendengar suara dengan tempo yang lebih lambat dihasilkan oleh *metronome* dan akan melihat tulisan tempo akan berkurang sesuai dengan tempo yang ditentukan oleh pengguna	**Berhasil**
		Menggeser *slider* untuk tempo ke arah kiri		**Berhasil**
2	Mempercepat tempo suara yang dihasilkan	Menekan tombol untuk menambah tempo	Pengguna akan mendengar suara dengan tempo yang lebih cepat dihasilkan oleh *metronome* dan akan melihat tulisan tempo akan bertambah sesuai dengan tempo yang ditentukan oleh pengguna	**Berhasil**
		Menggeser *slider* untuk tempo ke arah kanan		**Berhasil**
3	Mengatur tempo sesuai kecepatan penekanan	Menekan tombol *set tempo* berulang kali	Pengguna akan mendengar suara dengan tempo yang sesuai	**Berhasil**

No	*Test Case*	*Action*	*Expected Result*	*Result*
	tombol *set tempo*		dengan kecepatan pengguna menekan tombol dihasilkan oleh *metronome* dan akan melihat tulisan tempo akan berubah sesuai dengan tempo yang ditentukan oleh pengguna	
4	Membuka menu pemilihan jenis ketukan	Menekan tombol panah ke bawah yang ada di bagian atas tampilan *metronome*	Pengguna akan melihat menu pemilihan jenis ketukan terbuka dari atas	**Berhasil**
5	Menutup menu pemilihan jenis ketukan	Menekan tombol panah ke atas yang ada di bagian bawah menu pemilihan jenis ketukan	Pengguna akan melihat menu pemilihan jenis ketukan tertutup kembali	**Berhasil**
6	Mengganti jenis ketukan yang ingin digunakan untuk *metronome*	Menekan salah satu pilihan jenis ketukan pada menu pemilihan jenis ketukan	Pengguna akan melihat menu pemilihan jenis ketukan tertutup kembali dan melihat indikator ketukan berubah	**Berhasil**

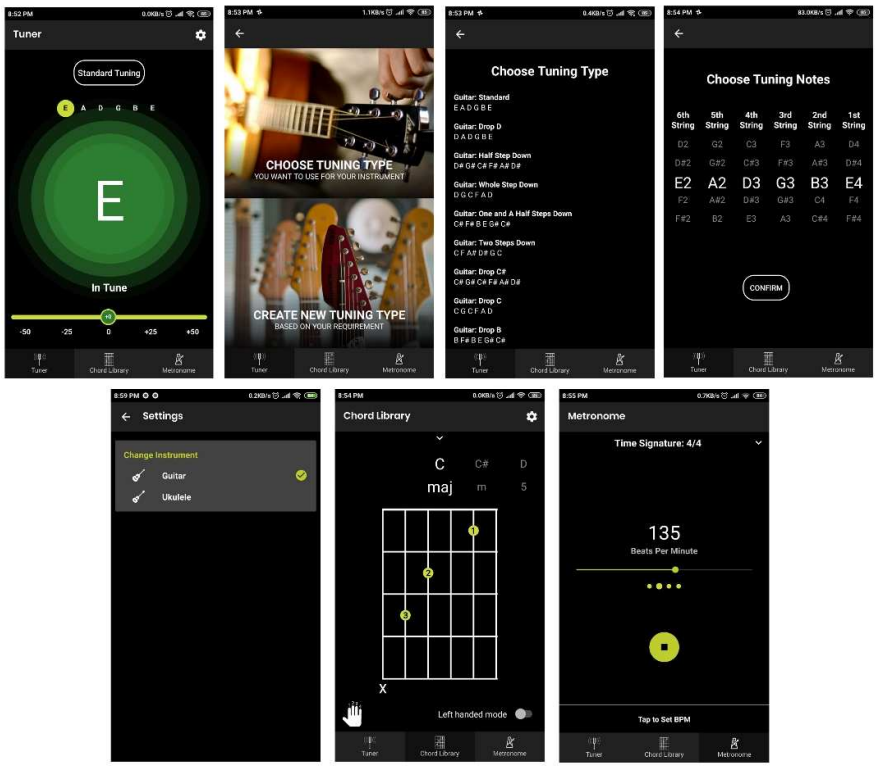
No	*Test Case*	*Action*	*Expected Result*	*Result*
			sesuai dengan jenis ketukan yang dipilih	
7	Memainkan suara sesuai dengan tempo dan jenis ketukan yang dipilih	Menekan tombol *play* / *stop* saat *metronome* sedang berhenti	Pengguna akan mendengar suara ketukan *metronome* bermain sesuai dengan tempo dan jenis ketukan yang dipilih pengguna	**Berhasil**
	Mengganti tulisan pada tombol *play* / *stop* menjadi *stop*		Pengguna akan melihat tulisan pada tombol *play* / *stop* berganti menjadi *stop*	**Berhasil**
8	Menghentikan suara *metronome* yang sedang bermain	Menekan tombol *play* / *stop* saat *metronome* sedang dimainkan	Pengguna akan mendengar suara ketukan *metronome* berhenti	**Berhasil**
	Mengganti tulisan pada tombol *play* / *stop* menjadi *play*		Pengguna akan melihat tulisan pada tombol *play* / *stop* berganti menjadi *play*	**Berhasil**

4.6 Evaluasi

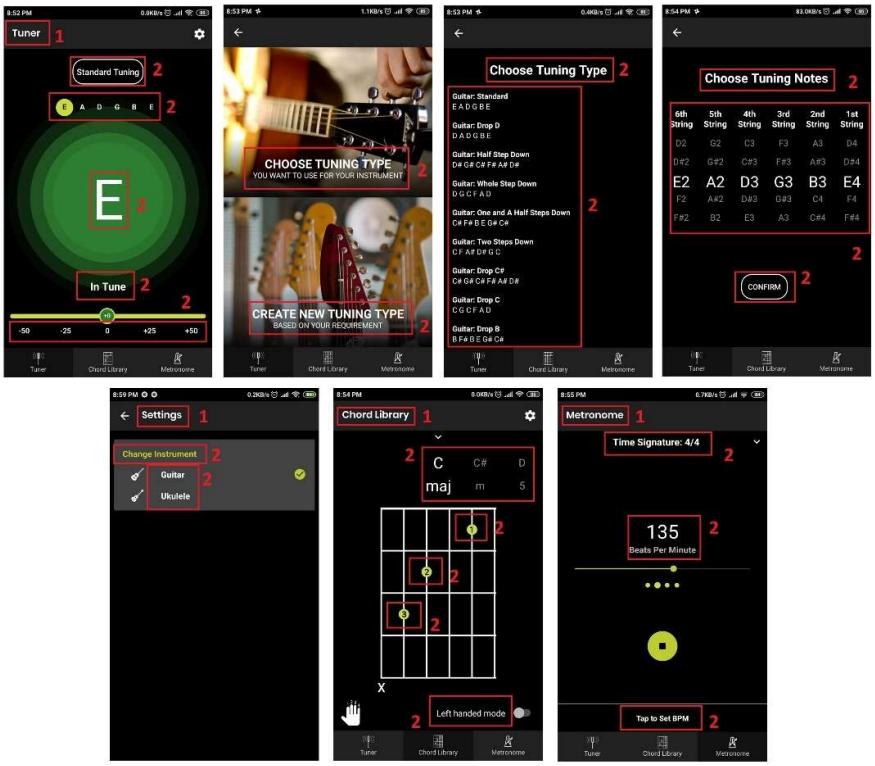
4.6.1. Evaluasi *User Interface* Berdasarkan Teori 8 *golden rules*

Berdasarkan teori mengenai 8 *golden rules* maka evaluasi yang dapat dihasilkan dari aplikasi ini meliputi:

1. *Strive for consistency*, konsistensi di dalam aplikasi ini dapat dilihat dari kesamaan jenis tulisan yang digunakan untuk judul halaman (ditandai dengan nomor 1 pada gambar di bawah), jenis tulisan yang digunakan di dalam konten dari setiap halaman (ditandai dengan nomor 2 pada gambar di bawah), dan tema warna dari setiap halaman yang sama yaitu warna hitam, putih, dan hijau lemon.

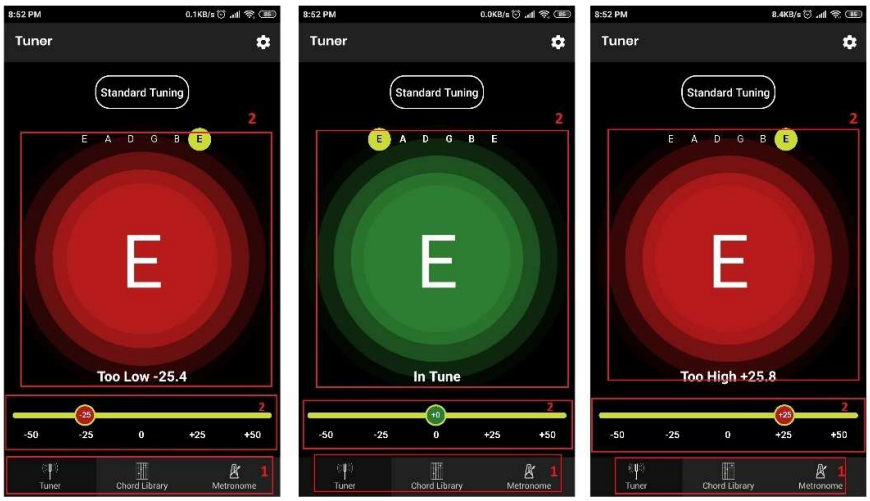


Gambar 4.47 Kesamaan Tema Warna dalam Aplikasi Chorus Tuner

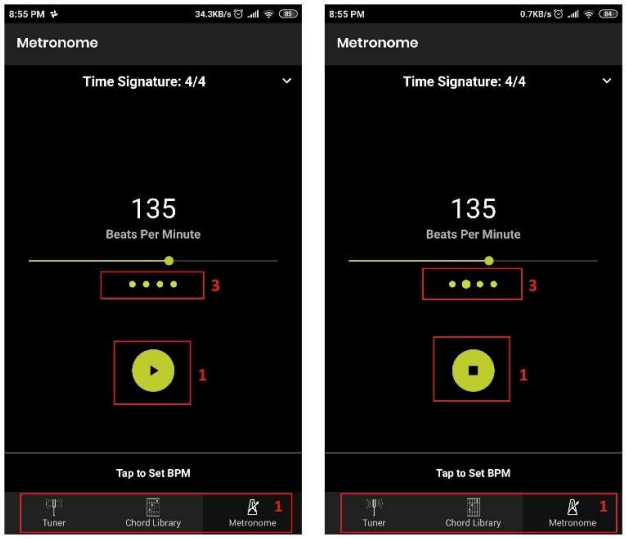


Gambar 4.48 Kesamaan Tulisan dalam Aplikasi Chorus Tuner

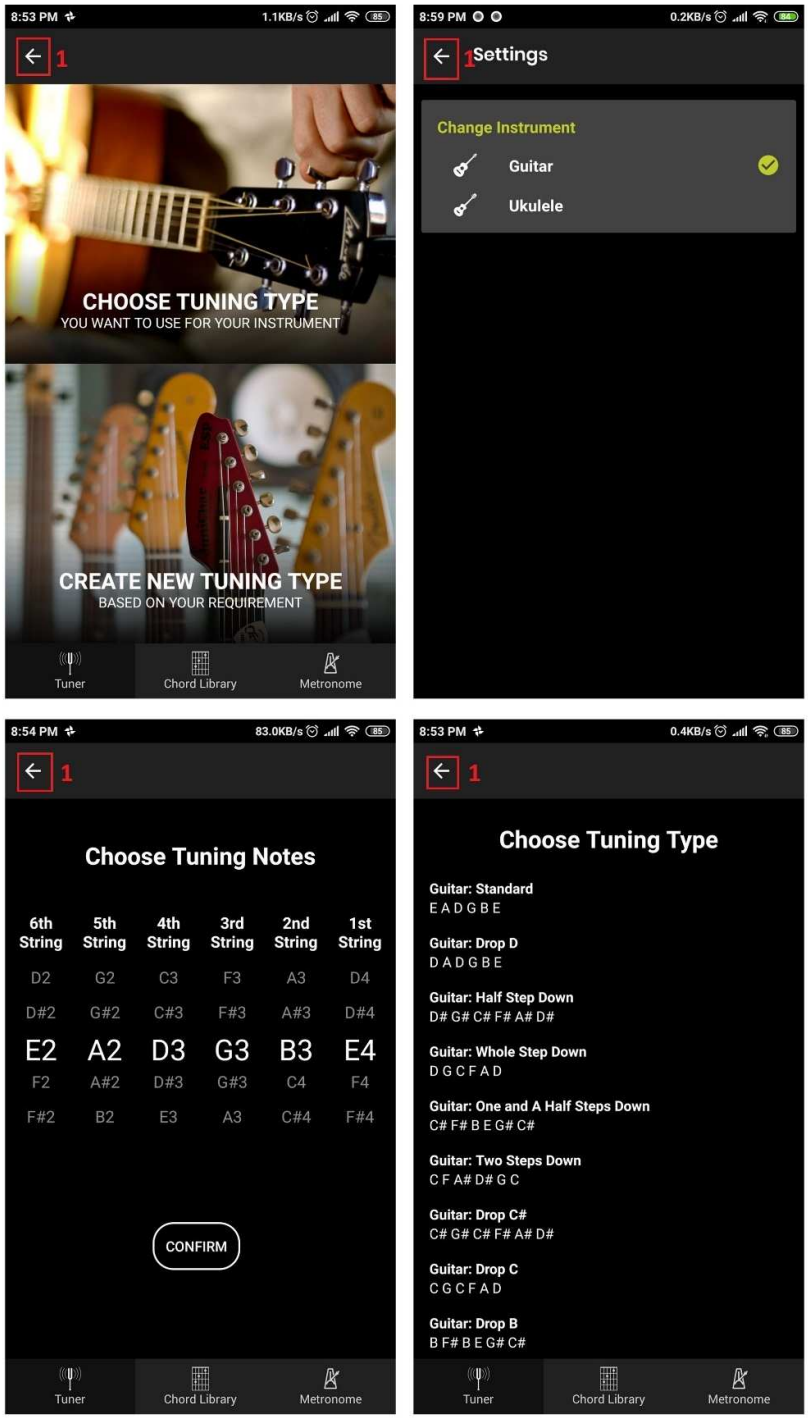
2. *Cater to universal usability*, rancangan *user interface* yang digunakan juga dirancang agar mudah dipahami terutama bagi pengguna yang masih baru. Hal ini dapat dilihat dari penggunaan simbol-simbol yang umum seperti panah ke kiri untuk *back*, penggunaan simbol pada tombol *play* / *stop*, dan simbol-simbol umum untuk menandakan *tab* masing-masing halaman (pada gambar di bawah ditandai dengan nomor 1). Pada fitur *tuner* warna pada *ripple* akan berubah menjadi hijau dan indikator akan berada tepat di tengah ketika kondisi *tuning* sudah "*in tune*" dan warna pada *ripple* akan berubah menjadi merah dan indikator akan bergerak ke kiri atau ke kanan ketika kondisi *tuning* instrumen pengguna "*too low*" atau "*too high*" (pada gambar di bawah ditandai dengan nomor 2). Bagian ini dirancang supaya pengguna mengetahui kondisi *tuning* dari senar instrumennya dengan mudah. Pada fitur *metronome* juga diletakkan sebuah indikator ketukan yang akan bergerak sesuai dengan tempo dan jenis ketukan yang digunakan (pada gambar di bawah ditandai dengan nomor 3). Bagian ini juga dirancang dengan tujuan supaya pengguna dapat memahami ketukan dengan mudah.



Gambar 4.49 Rancangan *User Interface* Fitur *Tuner* yang Dibuat Supaya Mudah Dipahami oleh Pengguna



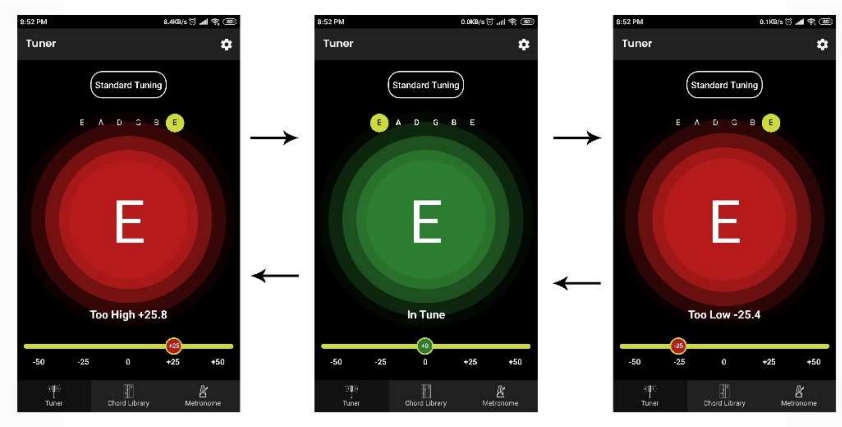
Gambar 4.50 Rancangan *User Interface* Fitur *Metronome* yang Dibuat Supaya Mudah Dipahami oleh Pengguna



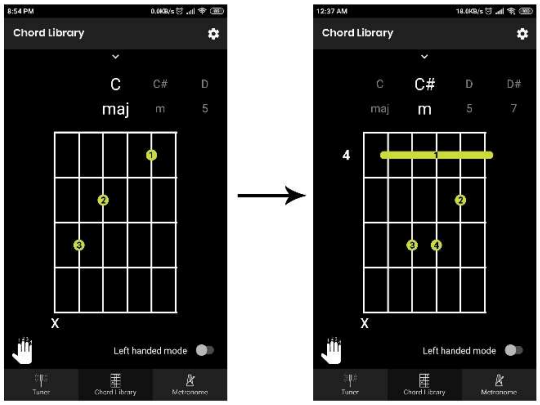
Gambar 4.51 Rancangan *User Interface* untuk Tombol *Back* yang Mudah Dipahami

3. *Offer informative feedback,* ditunjukkan melalui perubahan yang dominan ketika pengguna melakukan suatu *action,* misalnya ketika

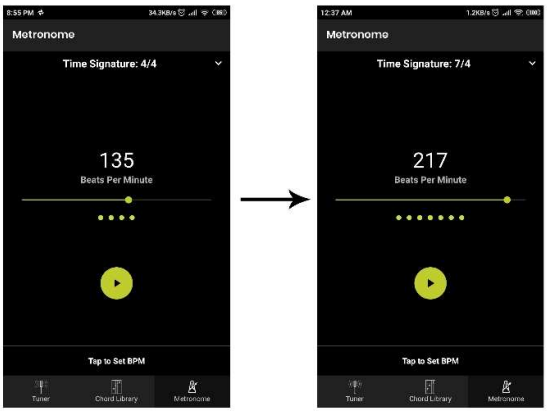
pengguna melakukan tuning pada halaman *tuner* maka indikator status *tuning* akan menunjukkan status *tuning*-nya (gambar 4.52), pada halaman *chord library* pengguna akan mendapatkan informasi mengenai *chord* yang ditampilkan yang akan berubah seiring pengguna menggeser daftar *chord* atau daftar varian *chord* (gambar 4.53), perubahan *layout* pada *metronome* ketika pengguna mengganti tempo atau jenis ketukan (gambar 4.54), aplikasi ini dapat memberikan informasi yang lengkap kepada pengguna tentang apa yang didapatkan oleh pengguna ketika dia melakukan sesuatu pada aplikasi.



Gambar 4.52 Perubahan pada Fitur *Tuner* Berdasarkan Status *Tuning* Senar pada Instrumen yang Dimainkan Pengguna



Gambar 4.53 Perubahan pada Fitur *Chord Library* Ketika Pengguna mengubah Chord dan Varian Chord yang Ingin Ditampilkan

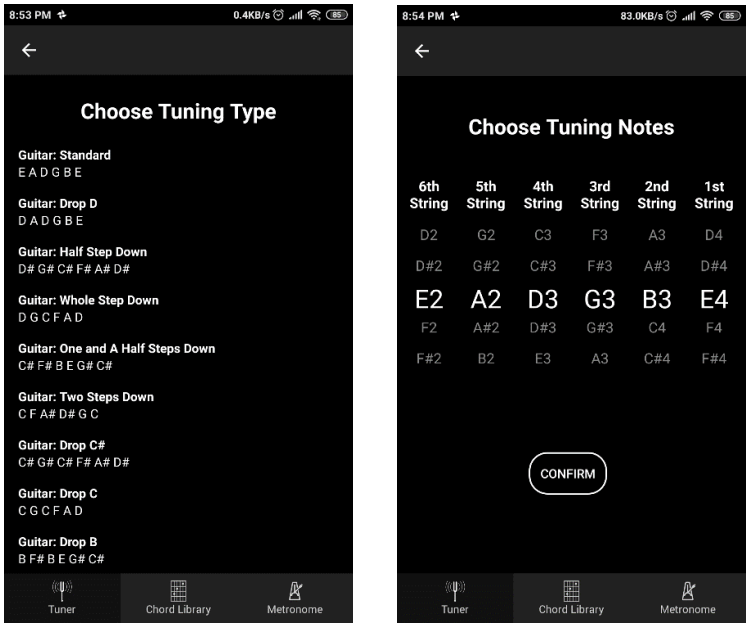


Gambar 4.54 Perubahan pada Fitur *Metronome* Ketika Pengguna Mengubah Jenis Ketukan dan Tempo yang Ingin Digunakan

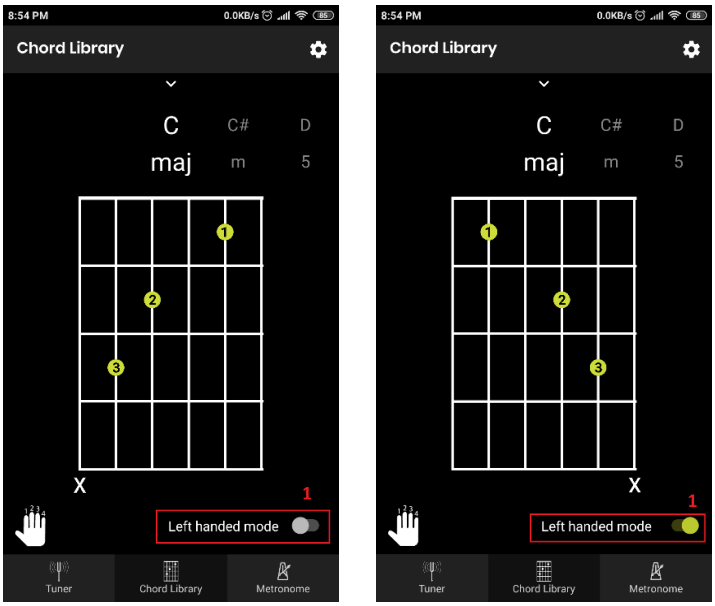
4. *Design dialogs to yield closure, action* yang dilakukan pengguna pada aplikasi ini tentu memiliki awalan yaitu pada saat pengguna membuka halaman masing-masing fitur, pertengahan yaitu pada saat pengguna mengoperasikan halaman masing-masing fitur, dan akhiran yaitu pada saat pengguna mengganti halaman dari satu fitur ke fitur lainnya atau pada saat pengguna keluar dari aplikasi.
5. *Prevent errors*, aplikasi yang dibuat diusahakan dapat terhindar dari adanya *bug* dan *error* dengan diadakannya *black-box testing* yang bertujuan untuk memastikkan tidak ada *bug* dan *error* yang ditemukan saat aplikasi digunakan oleh pengguna. Namun hal ini belum tentu menandakan aplikasi ini benar-benar terbebas dari *bug* dan *error*. Oleh karena itu aplikasi ini dibuat akan tertutup secara otomatis ketika mengalami *error* dan pengguna dapat menggunakannya kembali dengan membuka aplikasi itu kembali.
6. *Permit easy reversal of actions*, setiap pergantian halaman yang ada di aplikasi ini dapat dikembalikan ke kondisi semula menggunakan tombol back atau dengan mengganti halaman ke halaman fitur lain. Ketika sebuah halaman diganti ke halaman fitur lain maka halaman tersebut akan otomatis dikembalikan ke kondisi awal seperti ketika halaman tersebut baru pertama kali dibuka. Aplikasi ini juga menyediakan tombol *back* (ditunjukkan dengan nomor 1 pada gambar 4.51) untuk

mengembalikan halaman dalam satu fitur ke halaman sebelumnya ketika pengguna tidak jadi melakukan suatu *action.*

7. *Support internal locus of control*, saat menggunakan aplikasi ini pengguna akan merasa dirinya melakukan pengontrolan terhadap aplikasi dengan disediakannya banyak pilihan bagi pengguna untuk melakukan setting baik terhadap fitur *tuner* dengan pengguna dapat memilih tipe *tuning* atau bahkan dapat membuat tipe *tuning* sesuai dengan keinginan pengguna (gambar 4.55), pada fitur *chord library* dimana pengguna dapat melakukan pengaturan mode antara *left handed* atau *right handed* (ditandai dengan nomor 1 pada gambar 4.56) dan dapat mengganti diagram *chord* sesuai dengan *chord* yang ingin ditampilkan bentuknya (Gambar 4.53), dan pada fitur *metronome* dimana pengguna dapat memilih jenis ketukan dan melakukan pengaturan tempo yang ingin digunakannya untuk memainkan suara metronome (Gambar 4.54).



Gambar 4.55 Gambar Halaman Penggantian Tipe *Tuning* dan Pembuatan *Tipe* Tuning Baru



Gambar 4.56 Gambar Opsi Penggantian *Left Handed Mode* pada Fitur *Chord Library*

8. *Reduce short-term memory load*, jumlah halaman dari setiap fitur dibuat seminim mungkin dengan tujuan untuk menghemat memory namun tanpa menghilangkan fungsi-fungsi penting yang ada di dalam setiap fitur. Hal ini dilakukan dengan menyatukan halaman yang masih memiliki konteks yang sama ke dalam satu halaman, misalnya pada *chord library* setting untuk mode *left handed* sengaja diletakkan di halaman itu juga dan tidak dijadikan halaman baru, pada *metronome* menu penggantian jenis ketukan dibuat menggunakan *pop-up menu* supaya pengguna tidak perlu menuju ke halaman baru lagi untuk melakukan setting terhadap metronome.

4.6.2. Evaluasi Kepuasan Pengguna Berdasarkan Teori 5 Faktor Manusia Terukur

Pada tanggal 17 Januari 2019 sampai tanggal 18 Januari 2019 penulis melakukan evaluasi kepada lima orang pengguna potensial yang mungkin akan menggunakan aplikasi sejenis. Evaluasi ini dilakukan dengan cara menyebarkan beberapa pertanyaan yang dirancang berdasarkan teori 5 faktor manusia terukur untuk mengetahui seberapa puas pengguna ketika menggunakan aplikasi yang dirancang. Hasil eveluasi yang didapatkan meliputi:

1. *Time to learn*, menurut responden mereka dapat mengerti cara menggunakan fitur *tuner* dalam waktu paling lambat kurang dari 20 menit, fitur *chord library* dalam waktu paling lambat kurang dari 10 menit, dan fitur *metronome* dalam waktu paling lambat kurang dari 5 menit. Berdasarkan jawaban dari responden maka bisa dibilang responden dapat mengerti cara menggunakan aplikasi secara keseluruhan dalam waktu yang cepat.
2. *Speed of performance*, menurut responden kecepatan respons dari aksi yang diberikan pengguna pada fitur *tuner*, fitur *chord library*, dan fitur *metronome* dinilai cukup cepat. Berdasarkan jawaban tersebut maka penulis menarik kesimpulan bahwa performa dari aplikasi yang dibuat cukup cepat.
3. *Rate of errors by users*, menurut responden aplikasi yang dibuat dapat berfungsi dengan baik, hanya terdapat beberapa hal yang dapat ditambahkan seperti kurang akuratnya nada yang terbaca oleh *tuner* ketika suara gangguan di lingkungan sekitar pengguna sangat banyak dan ketika pengguna memetik senar gitar dengan suara yang kecil dan animasi yang terlihat kurang sinkron pada indikator metronome yang dapat terlihat ketika tempo yang digunakan memiliki nilai yang rendah.
4. *Retention overtime*, dari jawaban responden terhadap pertanyaan tentang kelebihan dan kekurangan fitur yang ada di dalam aplikasi maka penulis menyimpulkan aplikasi ini dapat diingat dalam waktu yang lama oleh pengguna karena tampilan yang memiliki ciri khas yang sederhana dan elegan namun informatif dan menarik.
5. *Subjective Satisfaction*, secara keseluruhan responden merasa puas setelah mencoba aplikasi yang dibuat karena tampilan yang sederhana dan elegan namun informatif dan menarik. Tampilan yang paling banyak disukai oleh responden adalah tampilan pada halaman *tuner* dimana pada halaman *tuner* terdapat animasi ripple yang menarik untuk dilihat. Meskipun responden sudah puas dengan tampilan yang ada sekarang namun menurut responden tampilan dapat dibuat lebih menarik lagi dengan adanya tambahan dekorasi terutama pada halaman *chord library* dan *metronome.*