BAB III

METODOLOGI PENELITIAN DAN PERANCANGAN SISTEM

3.1 Metodologi Penelitian

Metode penelitian ini metode yang digunakan adalah sebagai berikut.

3.1.1 Analisis Kebutuhan Sistem

Pada tahap ini akan dilakukan analisis apa saja yang dibutuhkan pada aplikasi *chatbot* yang akan dibangun, baik dari hasil analisis individu terhadap aplikasi-aplikasi yang ada maupun dari *paper*.

3.1.2 Telaah Literatur

Pada tahap ini akan dilakukan telaah literatur, yaitu membaca dan mempelajari *paper*, jurnal, buku, dan sumber-sumber lainnya mengenai *chatbot*, NLP, AIML, algoritma *stemming* Nazief & Adriani dan literatur-literatur lain yang berkaitan dengan aplikasi *chatbot* yang akan dibangun.

3.1.3 Design (Perancangan Sistem)

Berdasarkan hasil analisis kebutuhan sistem yang telah dilakukan sebelumnya, maka akan dilakukan perancangan sistem. Dalam melakukan perancangan sistem dibuatlah UML (*Unified Modeling Language*) yang biasa digunakan sebagai untuk pemodelan sistem berorientasi objek. Use case terdiri dari *use case diagram*, *activity diagram*, *sequence diagram*, serta *class diagram*. Dibuat juga suatu gambaran *collection* dari *database* yang digunakan. *Collection* dibuat karena *database* yang digunakan adalah Firebase dan Firebase tidak memiliki struktur tabel.

3.1.4 Pemrograman Sistem

Setelah dilakukan perancangan sistem, maka akan dimulai pemrograman untuk aplikasi *chatbot* ini sesuai perancangan yang telah dibuat sebelumnya. Aplikasi ini akan dibuat dengan menggunakan Android Studio sebagai *tools*-nya.

3.1.5 Testing dan Debugging

Tahap *testing* dan *debugging* ini dilakukan ketika pemrograman sistem telah selesai. *Testing dan debugging* dilakukan dengan tujuan untuk memastikan aplikasi berjalan dengan baik sesuai dengan alur yang telah dirancang sebelumnya, serta memastikan juga semua fungsionalitas baik pemrosesan *input* dan *output* aplikasi berjalan dengan lancar dan tidak terdapat *error* atau *bug*. *Testing* akan dilakukan dengan menyebarkan aplikasi ke pengguna agar pengguna dapat memberikan *user story* melalui aplikasi *chatbot* yang telah dibuat. Setelah itu pengguna akan diminta untuk mengisi kuisioner yang telah dibuat berdasarkan kuisioner TAM yang telah dijelaskan di literatur sebelumnya.

3.1.6 Evaluasi Sistem

Setelah dilakukan *testing* dan *debugging*, maka selanjutnya akan dilakukan tahap evaluasi sistem. Tahap ini bertujuan untuk mengevaluasi hasil yang telah dilakukan pada tahap *testing*, dari hasil tersebut akan disimpulkan bagaimana tingkat penerimaan pengguna terhadap aplikasi *chatbot* yang telah dibuat berdasarkan hasil kuisioner yang telah didapat.

3.1.7 Konsultasi & Penulisan Laporan

Pada tahap ini akan dilakukan konsultasi dengan dosen pembimbing dan akan disusun laporan dari hasil pembangunan aplikasi *chatbot* tersebut.

3.2 Perancangan Sistem

Model yang digunakan pada perancangan sistem ini adalah dengan metode *Object-Oriented* sehingga yang perlu dibuat adalah UML berupa *use case diagram, sequence diagram, class diagram,* dan *activity diagram.*

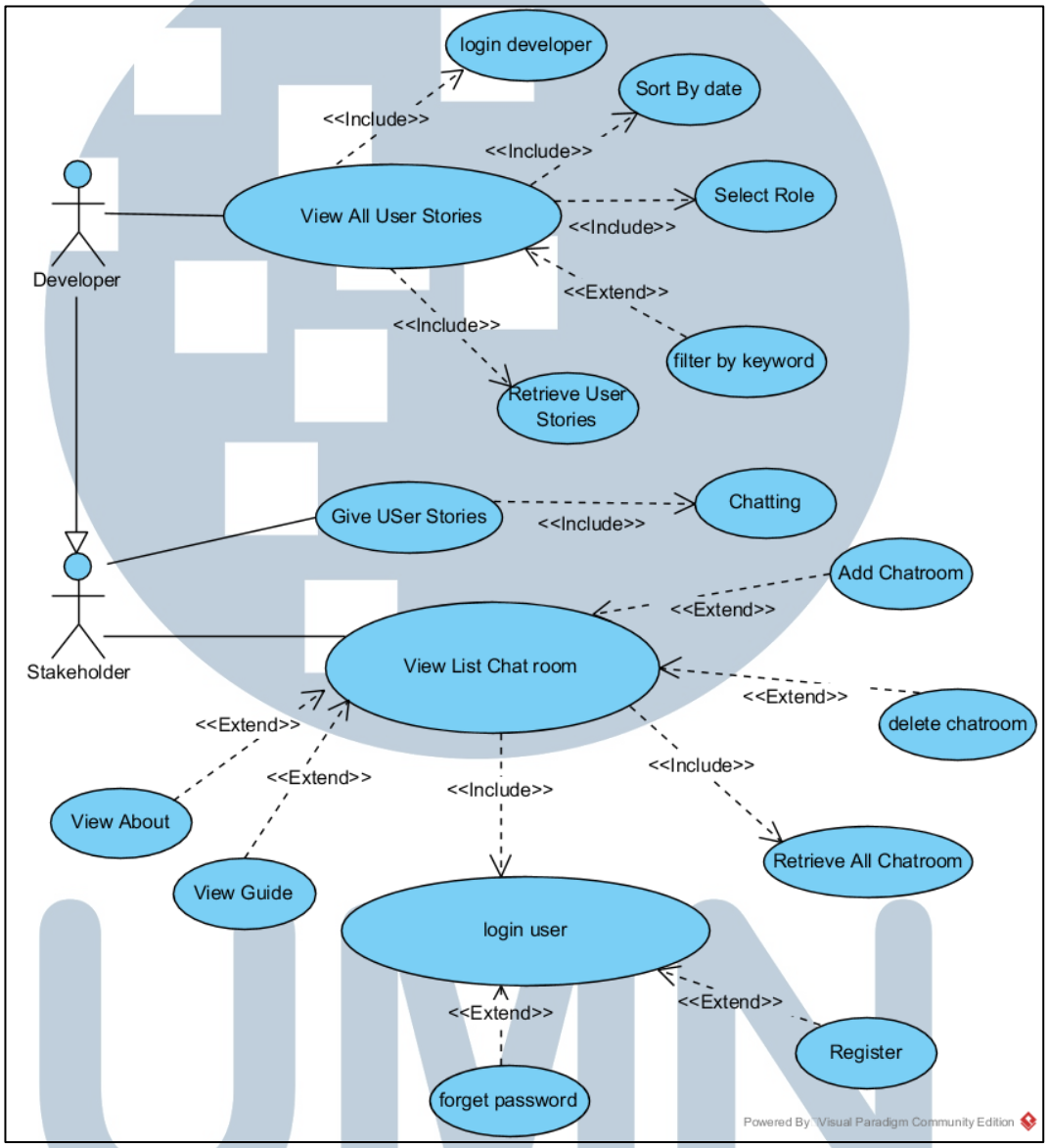
3.2.1 Use case Diagram

Use case dari aplikasi ini dapat dilihat pada Gambar 3.1. Pada Gambar tersebut dapat dilihat bahwa terdapat 2 aktor yaitu *user* dan *developer*, dimana beberapa *user* merupakan *developer.*

Pada *use case* tersebut *user* terhubung dengan 4 *use case* yaitu *use case login*, *use case register*, *use case give user stories* dan *use case view list chat room* sedangkan *developer* terhubung dengan *use case view all user stories.*

Untuk *user* terdapat beberapa *use case* lain seperti *retrieve all chat room, use case add chat room, use case delete chat room,* dan *use case chatting*. Use case *chatting* ter-*include* di *use case give user stories* karena *user* harus melakukan pertukaran pesan untuk dapat memberikan *user stories*. *Use case View List Chat Room* ter-*include use case login user* karena sebelumnya *user* harus melakukan *login* terlebih dahulu. *Use case forget password* dan *use case register* merupakan *extended use cases* dari *use case login user* karena *user* dapat melakukan *register* maupun *forget password* akan tetapi hal itu tidak harus dilakukan oleh *user*. *Use case retrieve all chat room* juga ter-*include* di *use case View List Chat room* karena sebelum *user* melihat daftar *chat room, user* harus me-*retrieve* data *chat room* terlebih dahulu. *Use case add chat room* dan *use case delete chat room* merupakan *extended use cases* dari *use case view all chat room*

karena *user* dapat melakukan penambahan *chat room* maupun penghapusan *chat room* tetapi tidak harus dilakukan.



Gambar 3.1 *Use Case Diagram*

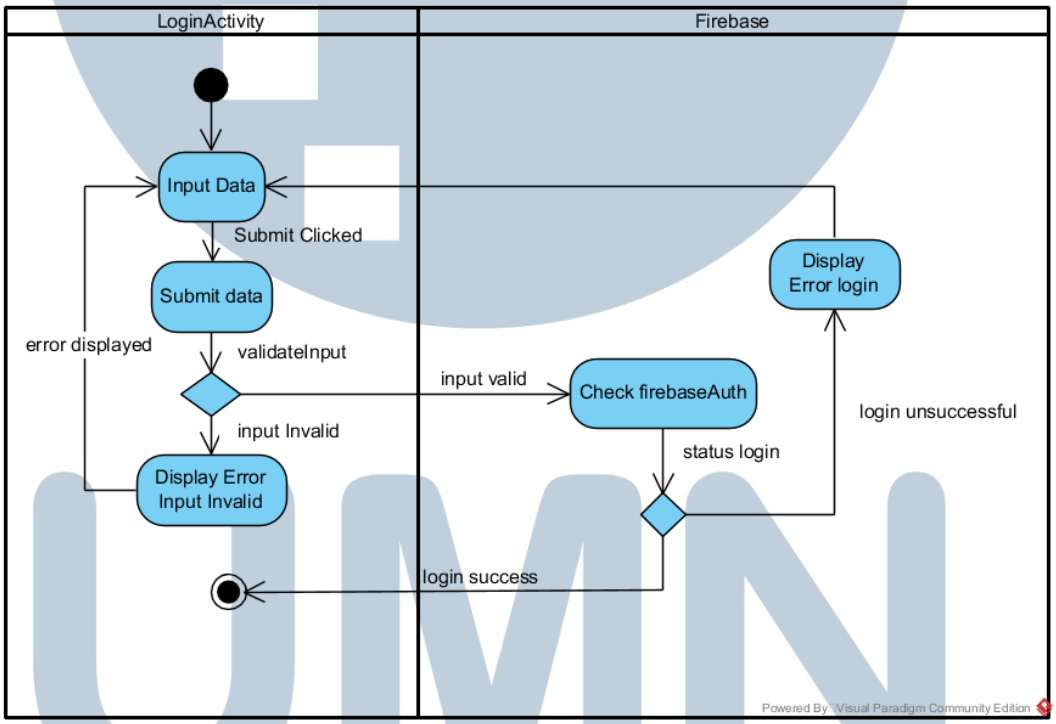
Untuk *developer* terdapat beberapa *use case* lain seperti *use case sort by date, use case select role, use case filter by keyword,* dan *use case retrieve user stories. Use case* retrieve *user stories, use case sort by date,* dan *use case select role* ter-*include* di *use case view all user stories* karena sebelum *developer* melihat data *user stories* harus dilakukan pemilihan *role,* data juga harus di-*retrieve,* dan

data diurutkan berdasarkan tanggal. *Use case filter by keyword* merupakan *extended use case* dari *use case view all user stories* karena *developer* dapat melakukan pencarian (*filtering*) berdasarkan *keyword* yang dimasukkan tetapi hal tersebut tidak harus dilakukan.

3.2.2 Activity Diagram

Terdapat beberapa *activity diagram* untuk *use case* yang telah dibuat pada Gambar 3.1.

A. *Activity Diagram Login*



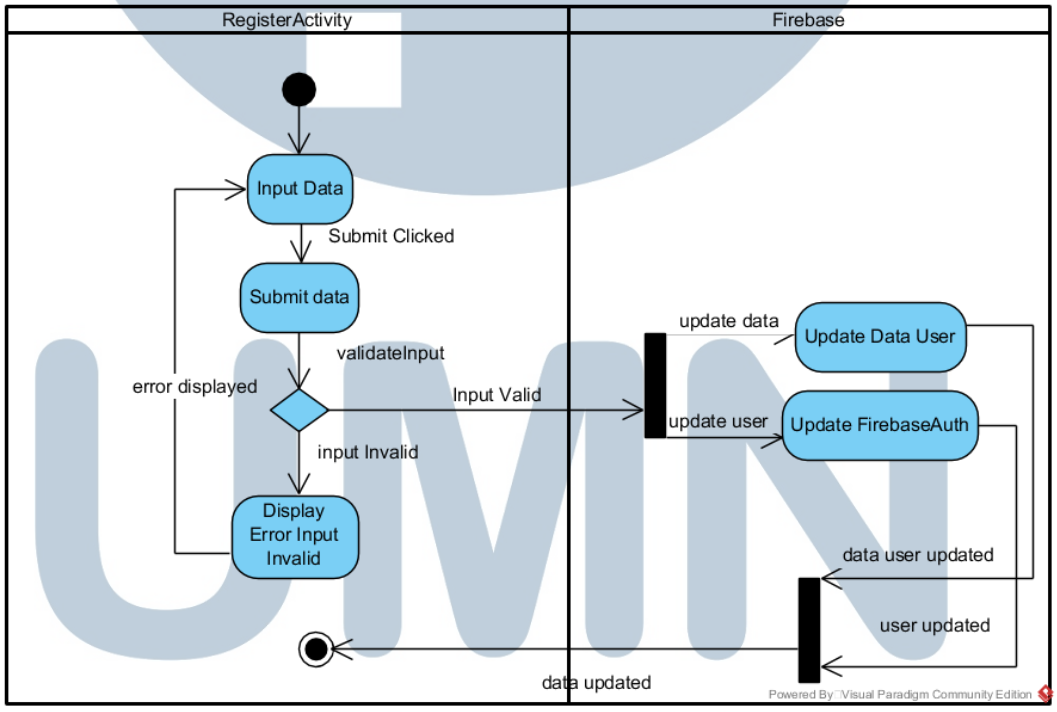
Gambar 3.2 *Activity Diagram Login*

Activity diagram login dapat dilihat pada Gambar 3.2. Untuk melakukan *login,* pertama *user* harus mengisi data. Data kemudian di-*submit* oleh *user,* akan tetapi sebelum data di-*submit* akan dilakukan pengecekan apakah *input user* tersebut *valid* atau tidak. Jika *input* tidak *valid* maka akan muncul pesan *error* sedangkan jika *input valid* maka akan diteruskan ke *database* dan dilakukan

pengecekan di Firebase *authentication* apakah *user* tersebut telah terdaftar. Jika *user* belum terdaftar maka akan muncul pesan *error* dan *user* dapat meng-*input* ulang data sedangkan jika telah terdaftar maka *login* akan berhasil.

B. *Activity Diagram Register*

Activity diagram *register* dapat dilihat pada Gambar 3.3. Untuk melakukan *register user* harus meng-*input* data terlebih dahulu selanjutnya ketika *user* melakukan *submit* data maka data akan dicek terlebih dahulu apakah data *valid* atau tidak. Jika data tidak *valid* maka akan muncul pesan *error* bahwa *input*-nya *invalid* sedangkan jika *input*-nya *valid* maka akan diteruskan ke *database* dan dilakukan *update* ke Firebase *authentication*-nya dan *update* ke data *user*.



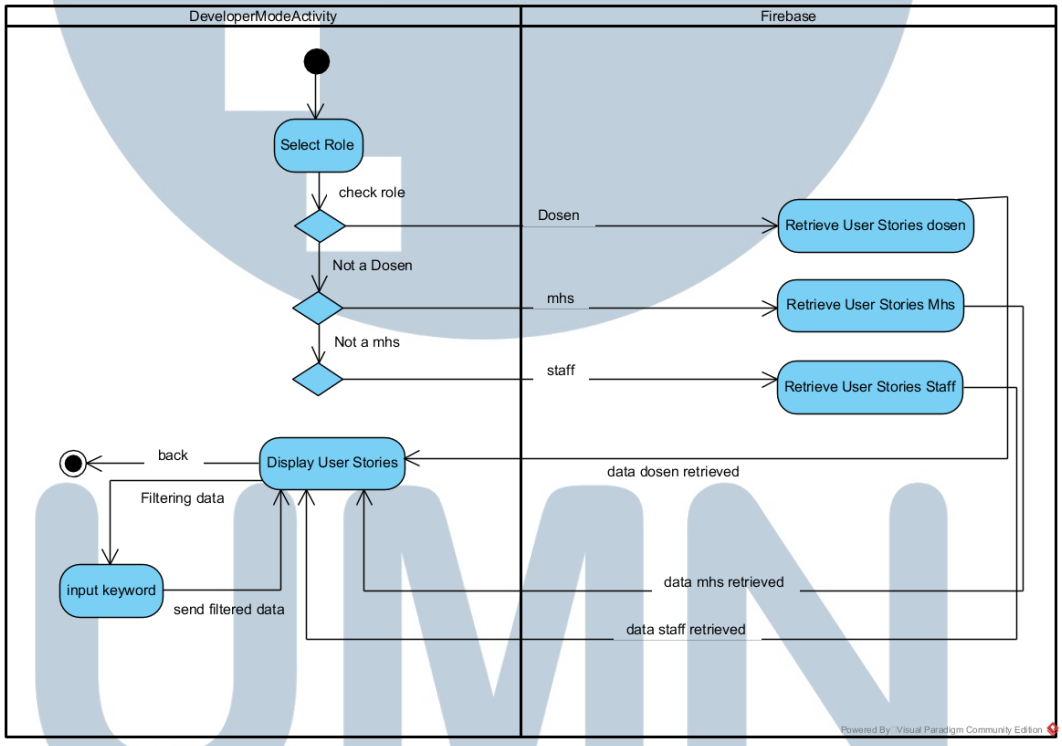
Gambar 3.3 *Activity Diagram Register*

C. *Activity Diagram View All User stories*

Activity diagram untuk *view all user story* dapat dilihat pada Gambar 3.4. Pertama-tama *user* akan memilih *role* yang akan dilihat *user stories*-nya. Terdapat 3 *role* yaitu dosen, mahasiswa dan staff. Ketika *user* sudah memilih salah satu

dari *role* tersebut maka akan dilakukan pengambilan data *user stories* dari Firebase. Data *user stories* yang diambil adalah berdasarkan dari *role* yang telah dipilih sebelumnya. Setelah data diterima maka data akan ditampilkan. Selain itu *user* juga dapat melakukan *filtering* data atau pencarian. *Filtering* dilakukan berdasarkan *keyword* yang dimasukkan oleh *user*. Setelah *user* memasukkan *keyword* maka *user stories* akan di-*filter* berdasarkan *keyword* tersebut dan data *filtering* pun akan ditampilkan.

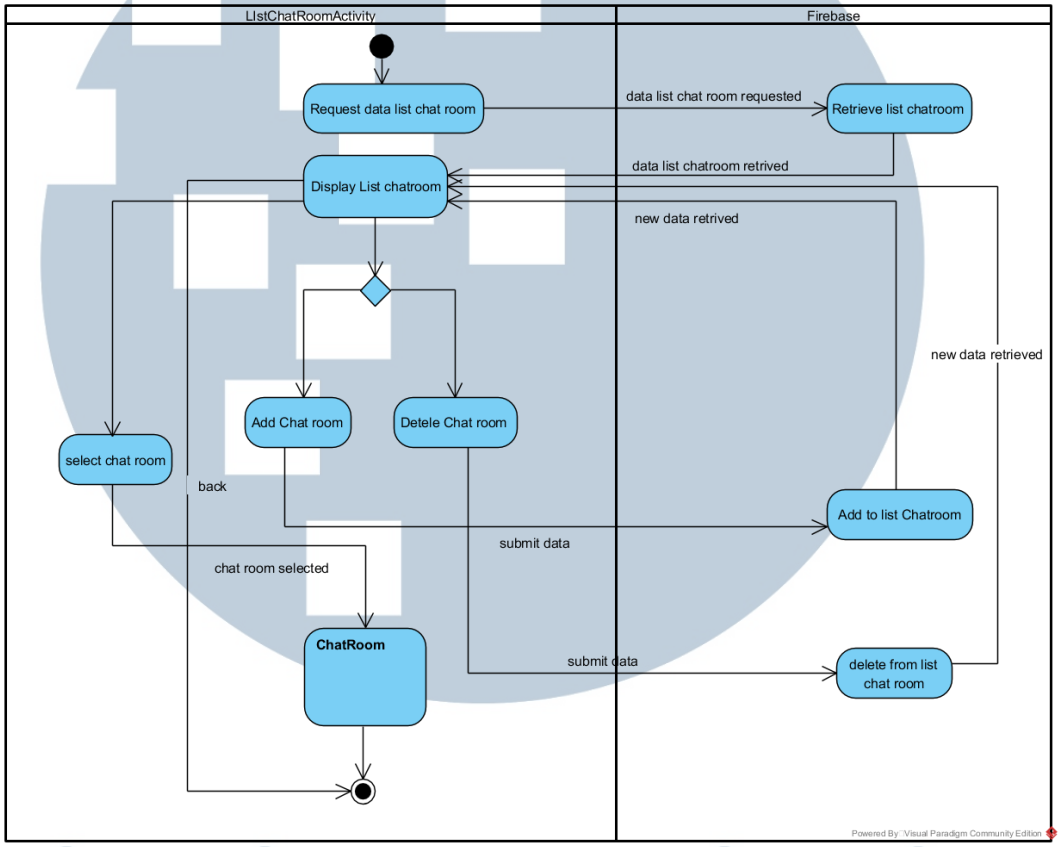
D. *Activity Diagram View List Chat room*



Gambar 3.4 *Activity Diagram View All User stories*

Activity view list chat room dapat dilihat pada Gambar 3.5. Pertama-tama akan dilakukan pengambilan data *list chat room* dari Firebase. Setelah itu data tersebut akan ditampilkan. *User* dapat melakukan penambahan *chat room* atau penghapusan *chat room*. Ketika *user* menambahkan *chat room* maka *list chat room* pada Firebase akan bertambah sedangkan ketika *user* menghapus *chat room*

maka *chat room* yang dihapus akan dihapus juga dari Firebase, setelah itu data baru akan dikirim. *User* membuka *chat room user* dapat memilih *chat room* yang terdapat pada *list chat room*.

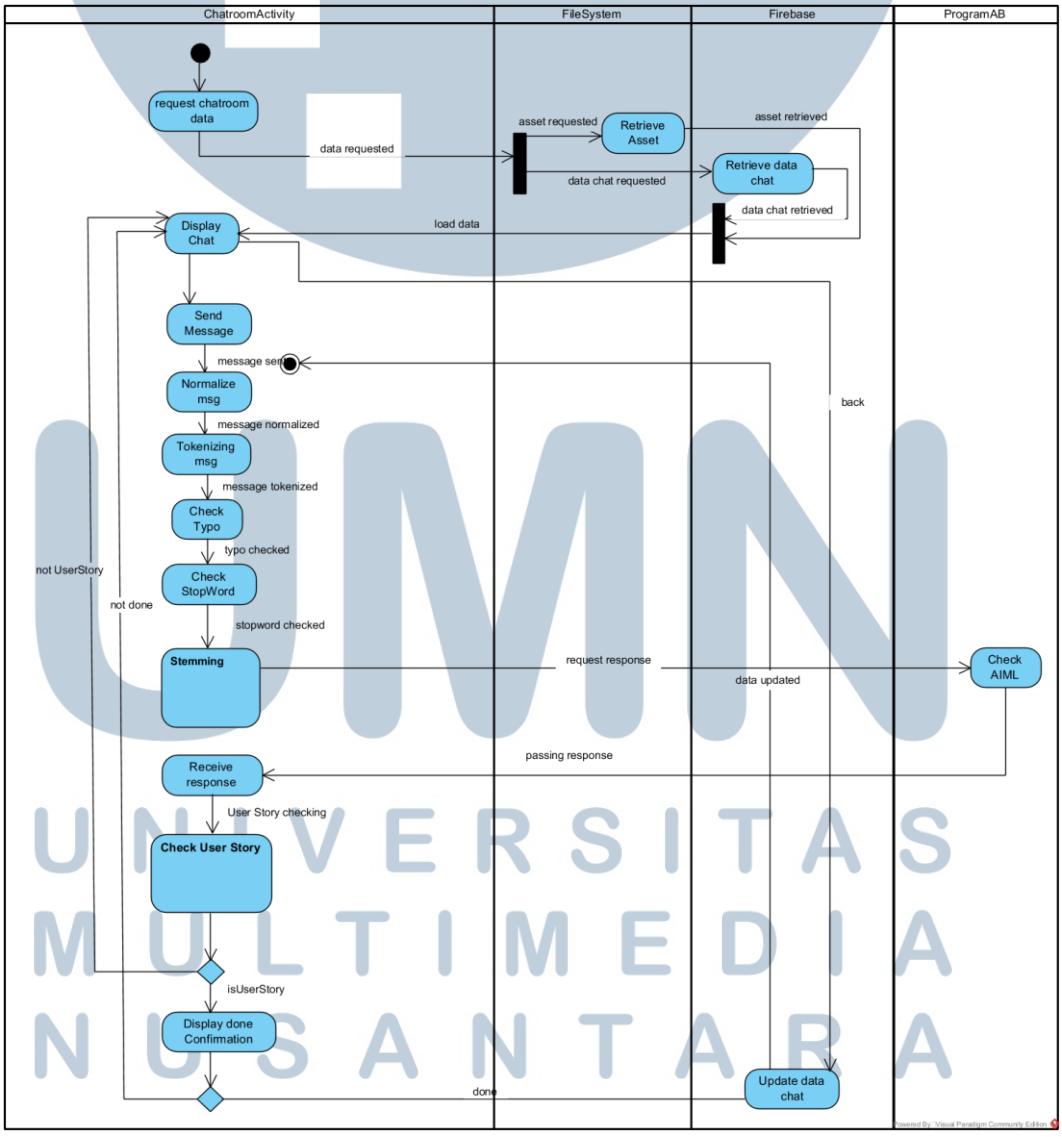


Gambar 3.5 *Activity Diagram View List Chat room*

E. *Activity Diagram Chatting*

Activity diagram Chatting dapat dilihat pada Gambar 3.6. Pertama-tama akan dilakukan pengambilan data *chat room* yang meliputi aset-aset yang dibutuhkan di *file system* maupun data *chat* dari Firebase. Setelah itu *chat* akan ditampilkan. Pada *activity* ini *user* dapat mengirimkan pesan ke *bot* atau jika *user* keluar dari *activity* ini maka akan dilakukan peng-*update*-an data *chat* di Firebase. Pesan yang dikirim pertama-tama akan dinormalisasi setelah itu dari hasil normalisasi akan dilakukan *tokenizing*. Setiap token akan dicek apakah terdapat

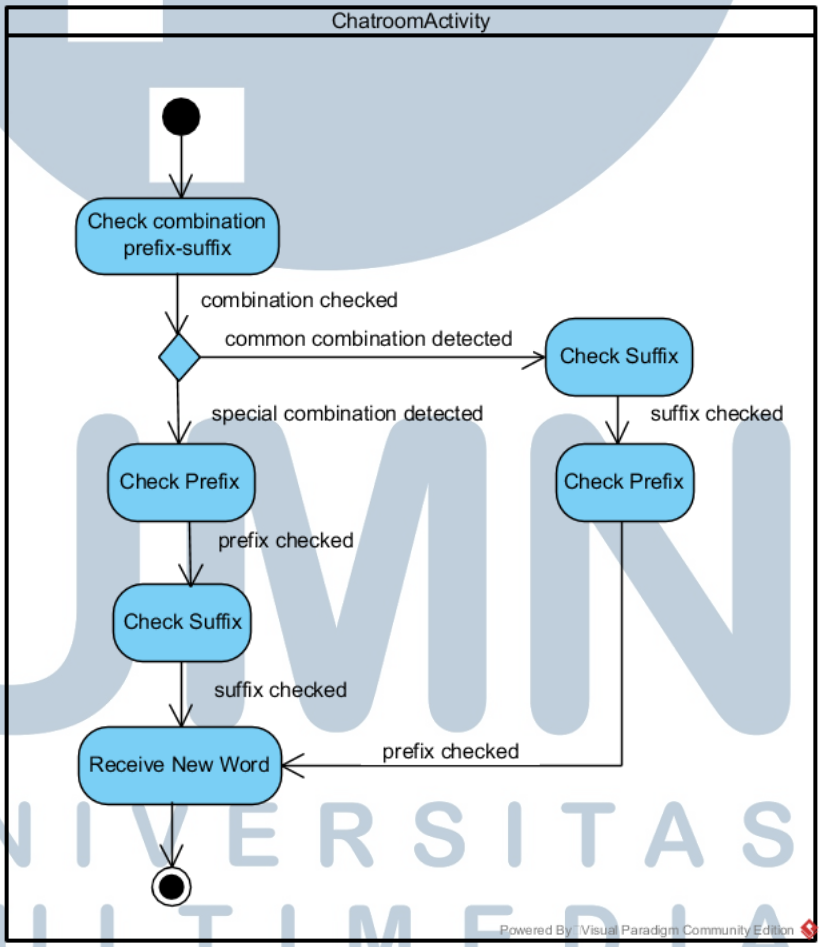
kesalahan pengetikan pada kata tersebut setelah itu akan dilakukan pengecekan terhadap *stop list* yang ada. Dari hasil pengecekan *stop list* tersebut akan dilakukan proses *stemming.* Setelah proses *stemming* selesai maka akan dilakukan pengecekan ke AIML atau *brain file*-nya untuk mendapatkan respon yang sesuai. Setelah respon diterima maka respon tersebut akan dicek apakah pesan yang dikirim *user* tadi merupakan *user story* atau bukan. Setelah itu jika pesan tersebut merupakan *user story* maka *bot* akan memberikan pesan konfirmasi apakah *user* masih ingin memberikan *user story* atau tidak. Jika sudah selesai maka akan dilakukan peng-*update*-an data *chat* pada Firebase.



Gambar 3.6 *Activity Diagram Chatting*

F. *Activity Diagram Stemming*

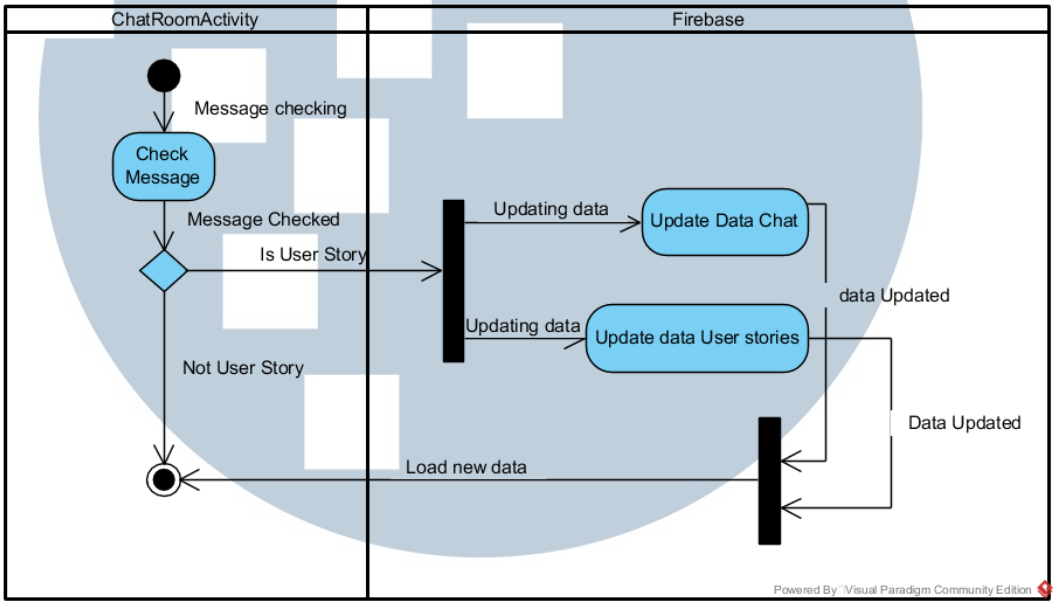
Activity Diagram Stemming dapat dilihat pada Gambar 3.8. Proses *stemming* dimulai dengan pengecekan kombinasi *prefix-suffix* karena terdapat kombinasi-kombinasi khusus yang akan menentukan urutan pengecekan *prefix-suffix*. Jika pasangan *prefix-suffix* tersebut merupakan kombinasi umum maka akan dilakukan pengecekan *suffix* terlebih dahulu setelah itu baru *prefix* akan dicek sedangkan jika pasangan *prefix-suffix* tersebut merupakan kombinasi khusus maka *prefix* akan dicek terlebih dahulu setelah itu baru *suffix* akan dicek. Setelah proses tersebut selesai maka kata baru akan diterima.



Gambar 3.7 *Activity Diagram Stemming*

G. *Activity Diagram Give User stories*

Activity diagram give user story dapat dilihat pada Gambar 3.8. Pada *activity* ini akan dilakukan pengecekan pesan. Jika pesan tersebut merupakan *user story* maka akan dilakukan peng-*update*-an data pada *database*. Data yang di-*update* adalah data *chat* dan data *user stories*.



Gambar 3.8 *Activity Diagram Give User stories*

3.2.3 Sequence Diagram

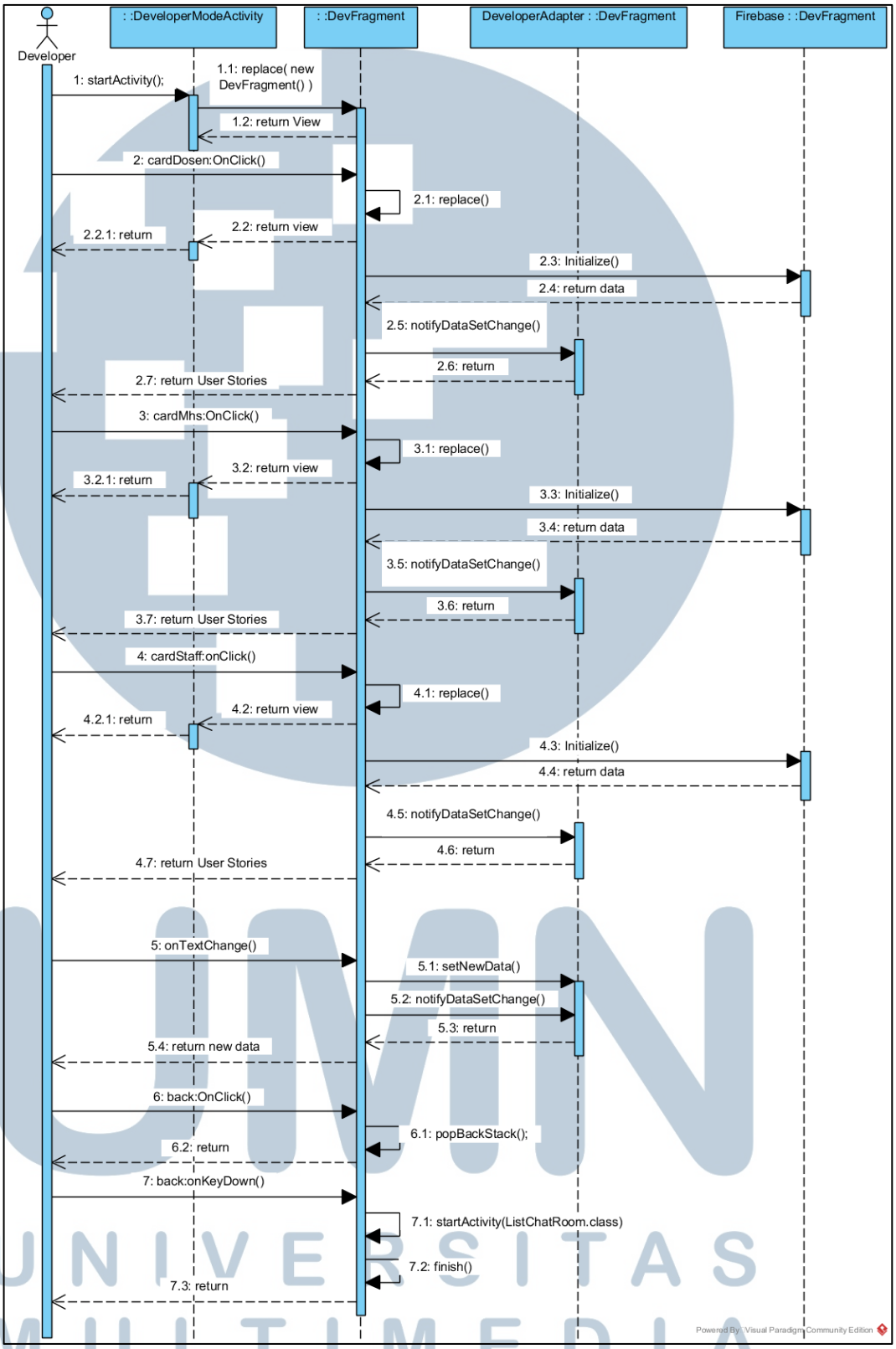
Terdapat 4 *sequence* diagram yang dirancang untuk aplikasi *chatbot* ini.

A. *Sequence Diagram* DeveloperMode

Sequence diagram untuk DeveloperMode dapat dilihat pada Gambar 3.9. Ketika DeveloperMode berjalan maka *function replace* akan langsung berjalan. *Function* tersebut digunakan untuk menimpa tampilan dari *class* tersebut dengan tampilan dari DevFragment. *User* dapat memilih cardView yang ada pada tampilan tersebut. Terdapat 3 cardView yaitu cardDosen, cardMhs, dan cardStaff. Ketika salah satu dari cardView tersebut dipilih maka *function* onClick pada cardView tersebut akan berjalan. Pada *function* tersebut akan dijalankan *function*

replace untuk menimpa tampilan tersebut dengan tampilan baru berdasarkan cardView yang dipilih lalu akan dilakukan *return* lalu *return* ke *developer*. Setelah itu akan dijalankan *function initialize* yang digunakan untuk mengambil data dari Firebase. Data yang diambil juga berdasarkan cardView yang dipilih. Setelah data didapatkan maka akan di-*return* dan *function* notifyDataSetChange akan berjalan. *Function* notifyDataSetChange berfungsi untuk memasukkan data yang didapat ke *class adapter* yang selanjutnya akan ditampilkan ke *user*.

User juga dapat melakukan *filter*/*search* dari data yang ditampilkan. Ketika *user* melakukan *search* maka *function* onTextChange akan berjalan. Pada *function* tersebut akan dilakukan proses *filtering* data berdasarkan *keyword* yang ditangkap dari *function* tersebut setelah itu data baru akan dikirim ke *class adapter* lewat *function* setNewData lalu *function* notifyDataSetChange akan berjalan dan data akan di-*return* dan ditampilkan. Terdapat tombol *back* di bagian bawah tampilan. Ketika tombol tersebut diklik maka *function* popBackStack akan berjalan. *Function* tersebut berfungsi untuk menghilangkan tampilan dari *fragment* cardView tersebut dan jika *user* menekan tombol *back* pada perangkat maka *function* startActivity(ListChatRoom.class) akan berjalan lalu *function* finish() akan berjalan, setelah itu akan dilakukan *return* ke *developer*/pengguna. *Function* startActivity(ListChatRoom.class) berfungsi untuk memulai *activity* dari *class* ListChatRoom sedangkan *function* finish() berfungsi untuk mengakhiri *activity* yang sedang berjalan sebelumnya.



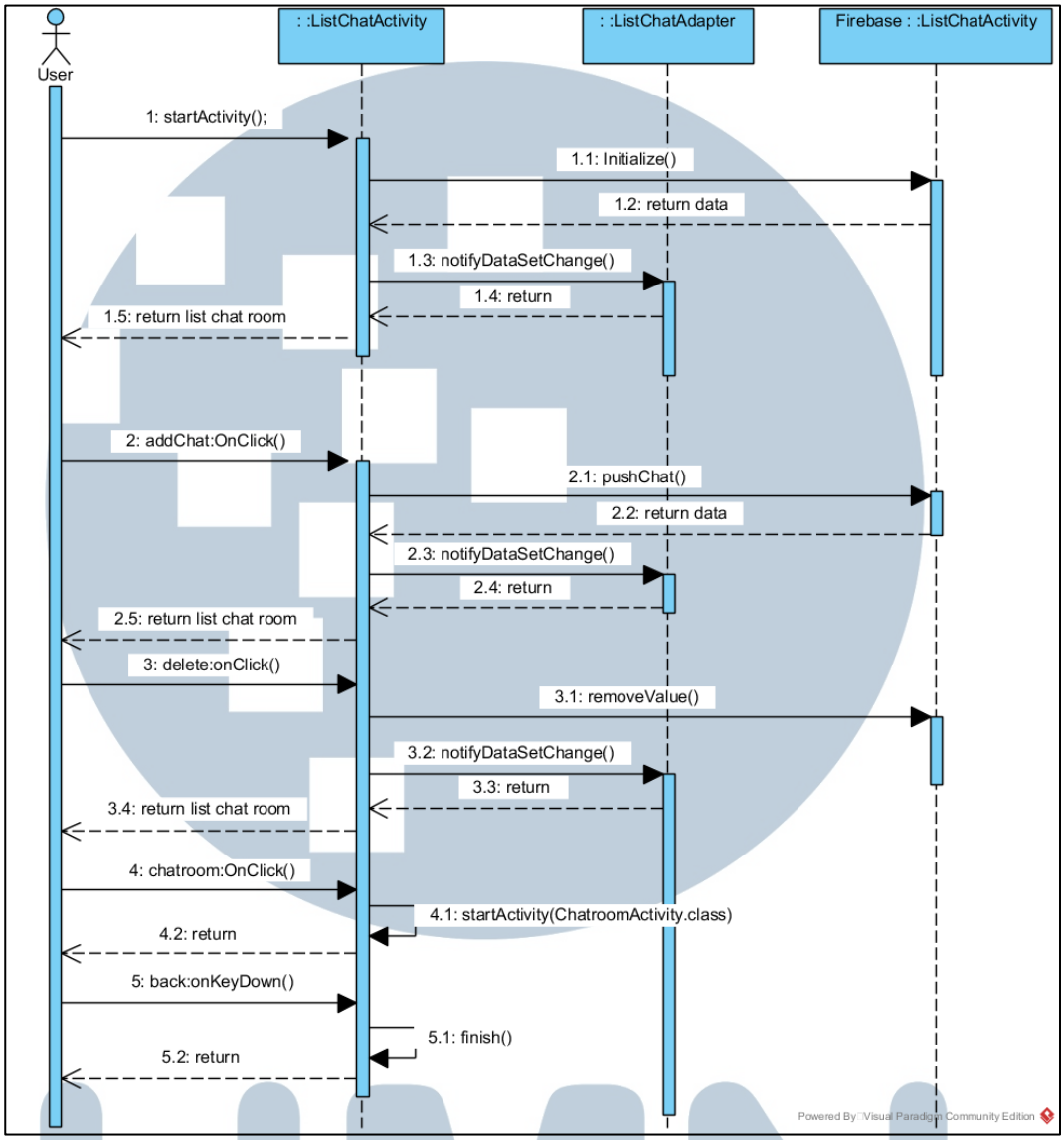
Gambar 3.9 *Sequence Diagram* DeveloperMode

B. *Sequence Diagram* ListChatActivity

Sequence diagram untuk ListChatActivity dapat dilihat pada Gambar 3.10. Ketika *class* ListChatActivity mulai maka akan dilakukan pemanggilan *function initialize* ke Firebase. *Function initialize* merupakan *function* yang digunakan untuk mengambil daftar *chat room* yang ada dari *database*. Setelah itu data akan di-*return* ke *activity* dan *function* notifyDataSetChange akan berjalan. *Function* notifyDataSetChange berfungsi untuk memasukkan data yang didapat ke *class adapter* yang selanjutnya akan ditampilkan ke *user*. Jika data kosong maka tidak ada yang ditampilkan.

Pengguna juga data menambahkan *chat room* dengan menekan suatu tombol yang nantinya akan menjalakan *function* onClick dari *button* tersebut. Ketika *function* tersebut jalan maka akan menjalankan *function* pushChat ke Firebase. *Function* pushChat berfungsi untuk memasukkan data *chat room* baru ke Firebase. Setelah data di-*push* maka data baru akan masuk sehingga *function* notifyDataSetChange akan kembali berjalan dan data baru tersebut akan ditampilkan ke *user*.

User juga dapat melakukan klik pada *chat room* yang ada. Ketika *chat room* diklik maka *function* startActivity(Chatroom.class) akan berjalan dan *activity* dari *class chat room* tersebut akan di-*return* dan terbuka. Ketika *user* menekan tombol *back* pada perangkat mereka maka *function* finish() akan berjalan dan dilakukan *return*.



Gambar 3.10 *Sequence Diagram* ListChatActivity

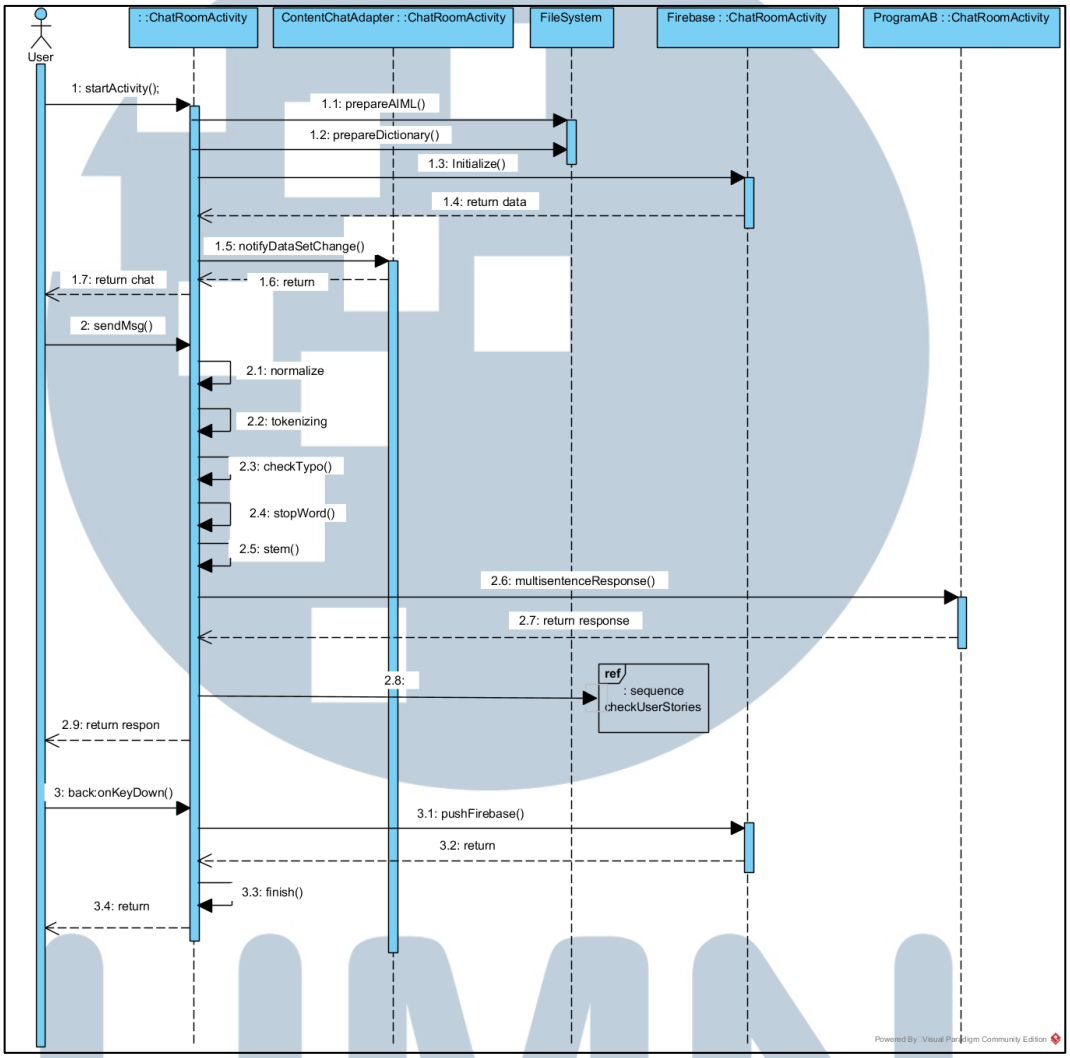
C. *Sequence Diagram* ChatRoomActivity

Sequence diagram ChatRoomActivity dapat dilihat pada Gambar 3.11. Ketika *activity* dari *class* ChatRoomActivity dimulai maka *function* prepareAIML, *function* prepareDictionary akan dijalankan ke *file system*. *Function* prepareAIML merupakan suatu *function* yang befungsi untuk mengambil data berupa *file* AIML yang nantinya akan di-*load* dan disimpan ke *memory* dan akan berfungsi sebagai *knowledge* dari *bot*. Setelah *bot* dibuat maka *function* prepareDictionary akan berjalan. *Function* tersebut merupakan *function*

yang berfungsi untuk mengambil data seperti daftar *stoplist*, daftar kata dasar, dan daftar kata *typo* yang nantinya akan digunakan untuk pemrosesan bahasa natural. Setelah kedua *function* itu berjalan maka *function initialize* akan berjalan. *Function initialize* berfungsi untuk mengambil data dari Firebase. Setelah itu data yang didapatkan akan di-*return* ke *activity* dan *function* notifyDataSetChange akan berjalan. *Function* notifyDataSetChange berfungsi untuk memasukkan data yang didapat ke *class adapter* yang selanjutnya akan ditampilkan ke *user*.

Ketika *user* mengirim pesan maka *function* sendMsg akan berjalan. Pada *function* tersebut, pesan yang dikirim akan diproses terlebih dahulu. Pertama-tama pesan akan dinormalisasi. Dari hasil normalisasi tersebut akan dilakukan *tokenizing*. Selanjutnya *function* checkTypo akan berjalan, dimana dari token-token yang ada akan dilakukan pengecekan *typo* pada setiap *token*-nya, jika terdapat token yang *typo* maka akan token tersebut akan diganti dengan kata yang benar. Selanjutnya jika pengecekan *typo* selesai maka *function* stopWord akan berjalan, dimana token-token tersebut akan dicek pada *stoplist* yang telah di-*load* diawal. Jika kata ditemukan maka kata tersebut akan dibuang. Setelah selesai maka *function stem* akan berjalan, dimana kata-kata tersebut akan di-*stemming*. Setelah proses *stemming* selesai maka akan didapatkan kata baru. Kata baru ini yang akan di-*passing* melalui *function* multisentenceResponse ke ProgramAB untuk mendapatkan respon dari *bot* tersebut. Selanjutnya akan dicek apakah melalui pesan dan respon tersebut merupakan *user story* apa bukan. Pada Gambar tersebut pengecekan *user story* merujuk pada *sequence diagram* checkUserStories yang diagramnya dapat dilihat pada Gambar 3.12. Setelah pengecekan dilakukan maka respon akan ditampilkan

Jika *user* menekan tombol *back* pada perangkat maka *function* finish() akan berjalan dan akan dilakukan *return*.

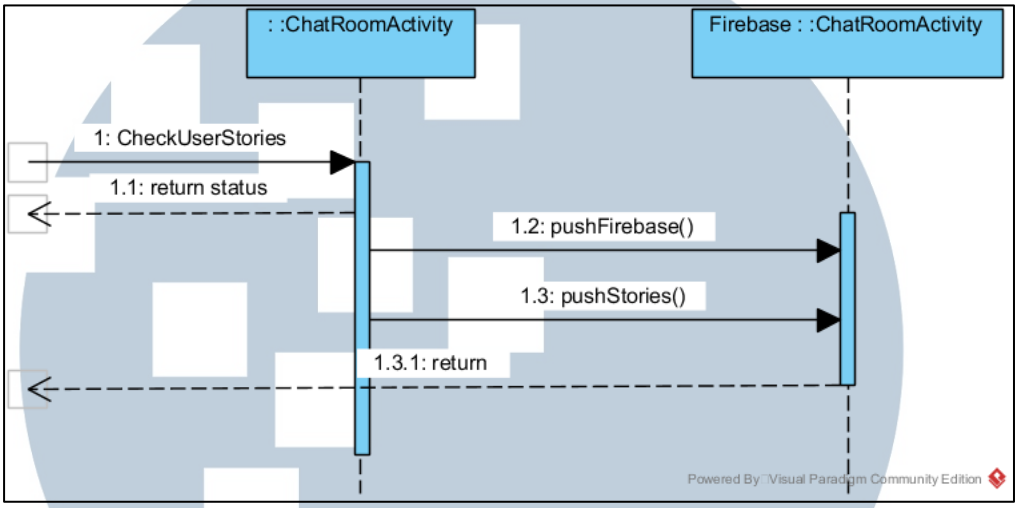


Gambar 3.11 *Sequence Diagram* ChatRoomActivity

D. Sequence Diagram CheckUserStory

Gambar 3.12 merupakan *sequence diagram* dari check*UserStory* dari proses pengecekan *user story* pada *sequence chat room*. Pada Gambar tersebut dapat dilihat jika pertama kali akan dilakukan pengecekan apakah pesan *user* adalah *user story*, setelah itu status tersebut akan di-*return*. Jika merupakan *user story* maka *function* pushFirebase dan pushStories akan berjalan. *Function* pushFirebase berfungsi untuk memasukkan data *chat* ke Firebase sedangkan

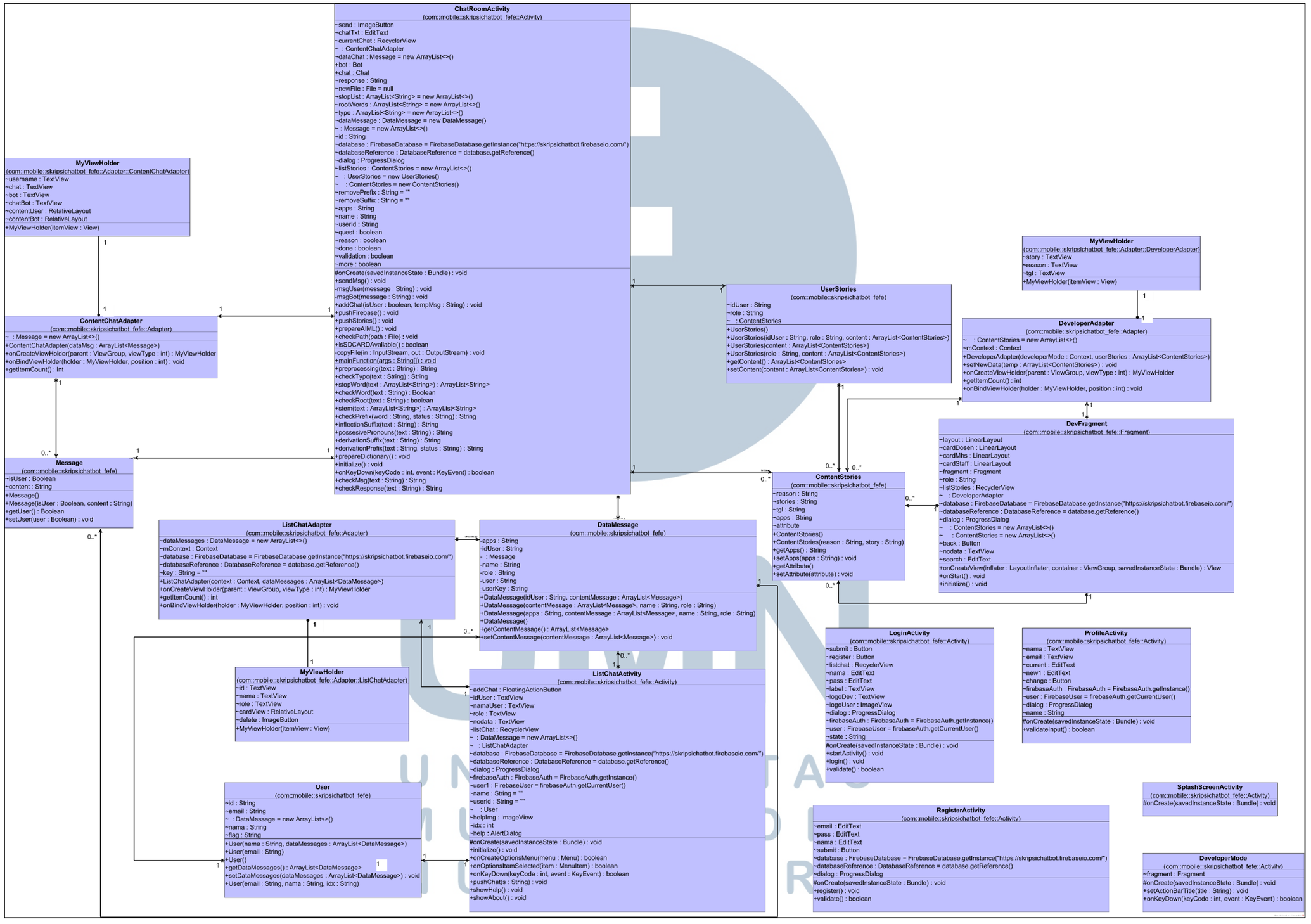
pushStories berfungsi untuk memasukkan data *user story* ke Firebase. Setelah selesai maka akan dilakukan *return*.



Gambar 3.12 *Sequence Diagram* CheckUserStories

3.2.4 Class Diagram

Class diagram untuk *chat bot* dapat dilihat pada Gambar 3.13. Terdapat beberapa *class* yang dibuat pada aplikasi *chat bot* ini. Untuk *class* ChatRoomActivity, *class* DeveloperMode, class *ListChatActivity*, class LoginActivity, *class* ProfileActivity, *class* RegisterActivity dan *class* SplashScreenActivity berada pada *package/folder activity*. Untuk *class* ContentChatAdapter, *class* DeveloperAdapter, dan *class* ListChatAdapter berada pada *folder adapter* karena merupakan *class adapter* yang digunakan untuk menampilkan data pada objek recyclerView yang ada di dalam beberapa *activity*. *Class* devFragment digunakan sebagai *fragment* yang menempel pada *class* DeveloperMode. Untuk *class* ContentStories, *class* DataMessage, *class* Message, *class User*, dan *class User*Stories digunakan sebagai model untuk menampung data-data tertentu.



Gambar 3.13 *Class* Diagram

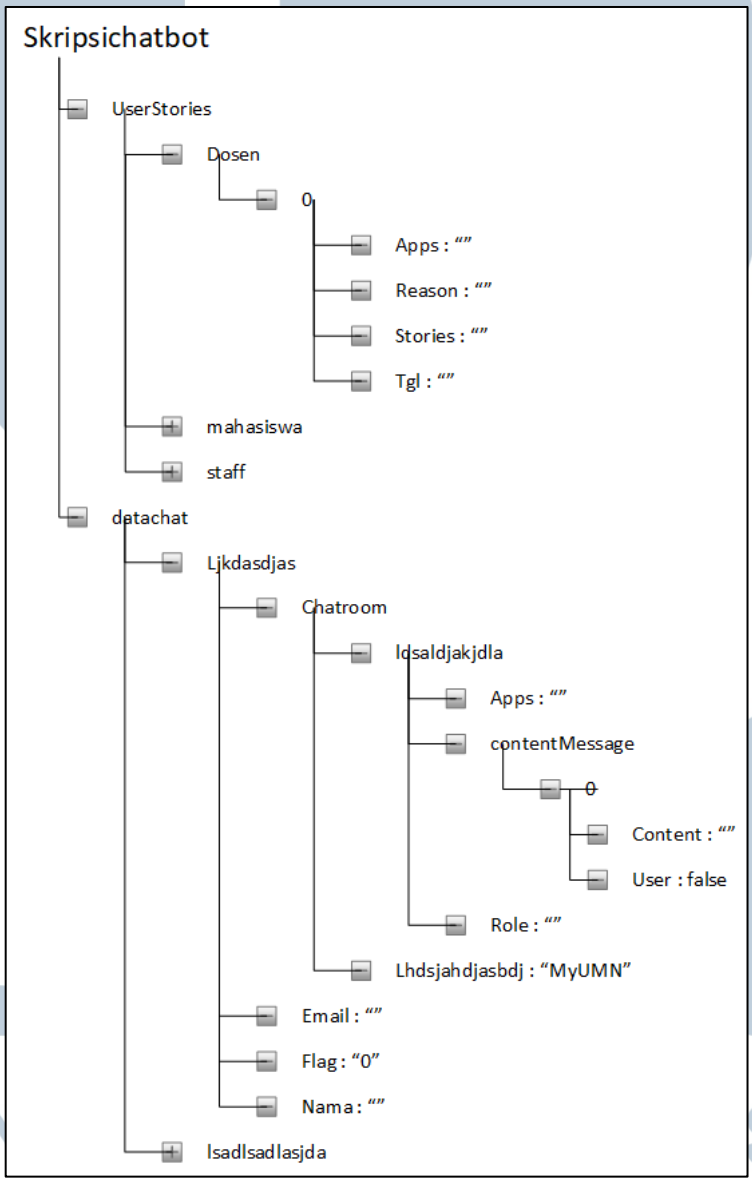
3.2.5 Firebase Database

Database yang digunakan dalam aplikasi ini adalah Firebase. Gambaran struktur *Collection* dari *database* dapat dilihat pada Gambar 3.14. Pada *collection* tersebut terdapat 2 *child* utama yaitu *child* UserStories dan *child* datachat.

*Child User*Stories merupakan *child* yang menampung data *user stories* baik dari dosen, mahasiswa atau staff. Setiap *child* dosen, mahasiswa, maupun staff bisa memiliki beberapa atau banyak *child* nantinya yang namanya merupakan angka *increment* mulai dari 0 dan *child* tersebut berisikan data *user stories* yaitu *apps* yang merupakan nama dari aplikasi, *reason* yang merupakan alasan dari *story* yang telah diberikan, *stories* yang merupakan *stories* dari pengguna dan tanggal *user story* tersebut diberikan.

Child datachat memiliki beberapa *child* dibawahnya dimana nama *child* tersebut merupakan kombinasi angka dan huruf yang di-*generate* secara acak ketika data ditambahkan. *Child* tersebut merupakan *child* yang menampung data *user*. Data *user* terdiri dari data diri seperti chatroom, email, flag, dan nama. *Child* chatroom merupakan *child* yang menampung setiap data *chat room* pengguna dan chatroom nantinya bisa memiliki beberapa atau banyak *child*. Nama dari *child* chatroom juga merupakan kombinasi angka dan huruf yang di-*generate* secara acak ketika data ditambahkan. Ketika pertama kali *child* dari chatroom bertambah maka *child* hanya berisikan nama *apps*. Jika sudah tedapat data maka child dari chatroom yang sebelumnya hanya berisikan nama *apps* akan ter-*update child*-nya dan akan memiliki *child* lain seperti contentMessage dan *role*. *Child* contentMessage merupakan *child* yang menampung data *chat* setiap chatroom. Setiap memiliki data baru maka *child* dari contentMessage akan bertambah. Nama

child adalah angka hasil *increment* mulai dari 0. Setiap *child* tersebut memiliki 2 *child* yaitu *child content* yang merupakan *content* dari pesan dan *child user* yang berisikan *Boolean true* atau *false*, dimana jika true maka *content* tersebut merupakan *chat user* sedangkan jika *false* maka *content* tersebut merupakan *child* dari *bot*.



Gambar 3.14 Gambaran Struktur *Collection* Firebase