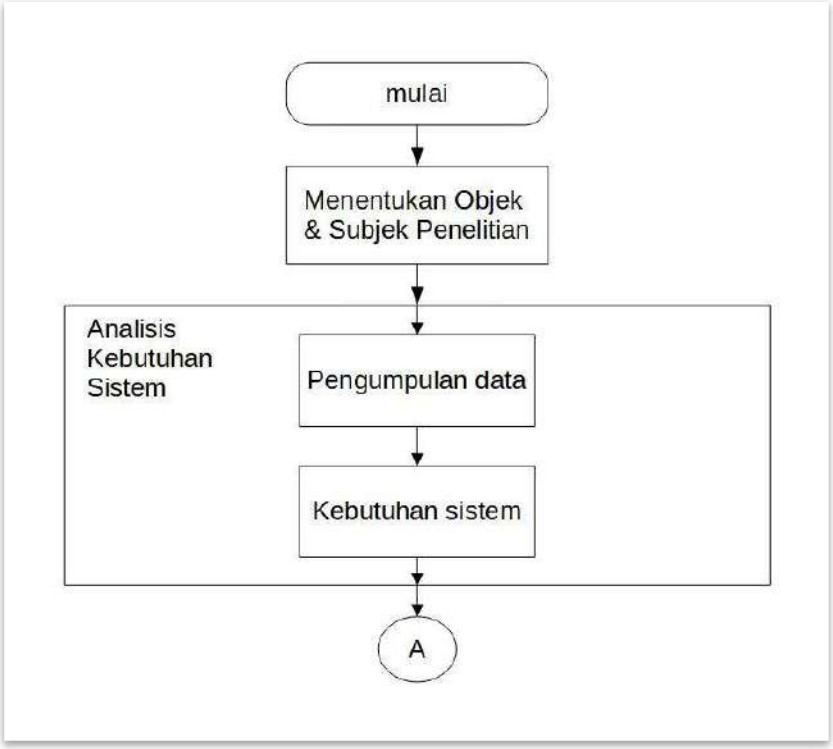
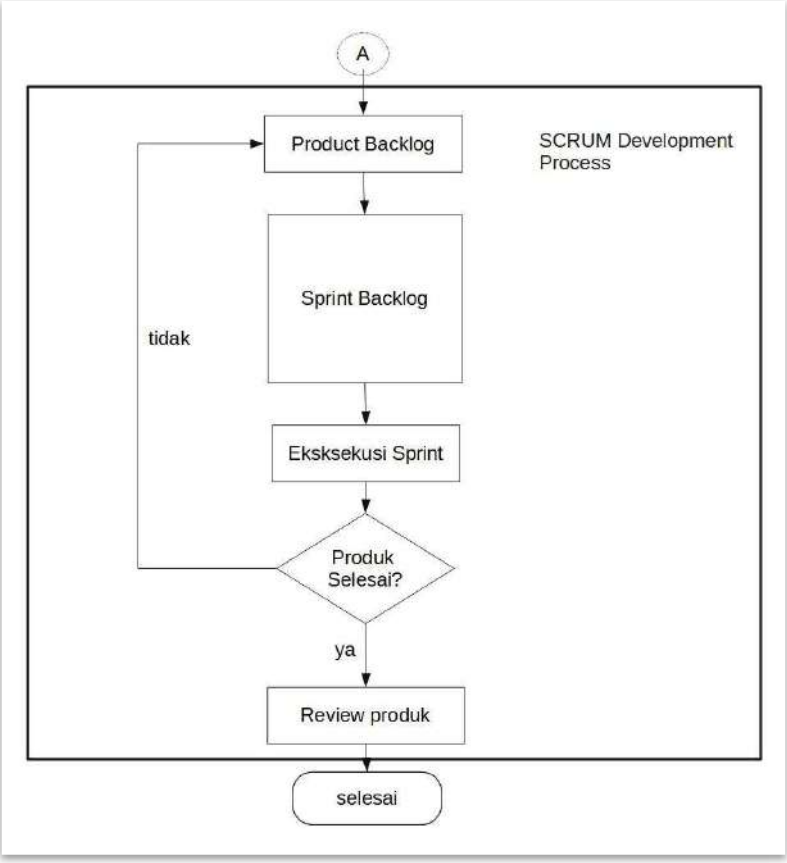
# BAB 3 METODE PENELITIAN

## 3.1 Diagram Alir Penelitian

Pada bab ini akan dibahas mengenai langkah penelitian yang dilakukan dalam Tugas Akhir ini. Bab ini juga akan membahas mengenai objek dan subjek penelitian




*Gambar 3.1 Alir Penelitian*




*Gambar 3.2 Alir Penelitian 2*

### 3.2 Objek Dan Subjek Penelitian

Berdasarkan latar belakang yang telah diuraikan pada Bab 1 diatas, objek pada penelitian ini adalah aplikasi mobile igracias IT Telkom Purwokerto. Dan subjek nya adalah mahasiswa terkhususnya angkatan di semester 5 keatas di IT Telkom Purwokerto

### 3.3 Analisis Kebutuhan Sistem

Pada tahap ini, dibahas mengenai kebutuhan desain dari aplikasi yang akan dibuat, mulai dari spesifikasi yang dibutuhkan, *desain interface*, hingga desain UML yang meliputi *use case*, *sequence*, dan *activity diagram.* Proses Analisa kebutuhan sistem diawali dengan proses pengumpulan data yang kemudian dilanjutkan dengan menganalisanya

### 3.3.1 Pengumpulan Data

Pada tahapan ini, Pengumpulan data dilakukan dengan 3 cara, pengumpulan data primer penulis lakukan dengan kuisioner ke mahasiswa dan wawancara dengan sisfo, beberapa laboran FIF dan studi literatur. Pengumpulan data sekunder penulis lakukan dengan studi literatur

a. Kuisioner

Penyebaran kuisioner dilakukan ke mahasiswa It Telkom Purwokerto dari tanggal 2 februari 2021 – 26 april 2021 dengan kriteria minimal semester 6 dan alumni IT Telkom Purwokerto, dimana kuisoner ini digunakan untuk menghimpun data awal penelitian

b. Wawancara

Wawancara dilakukan terhadap laboran FIF pada tanggal 21 mei 2021, Laboran FIF dipilih karena banyaknya jumalah mahasiswa dari FIF sehingga Seringnya Pihak Laboran FIF berinteraksi dengan mahasiswa. Narasumber wawancara diwakili oleh staff Laboran FIF dari laboratorium aplikasi yaitu Anggi Iskandar Aziz, a.Md dan Adinda Rahmi Saraswati, S.Kom. Adapun Hasil wawancara terdapat pada lampiran 1 dan 2

c. Studi Literatur

Data yang dibutuhkan penulis untuk membangun sistem ini adalah dengan membaca serta mempelajari referensi yang terdapat pada jurnal, buku dan skripsi. Selain itu, penulis juga mempelajari referensi lain melalui internet untuk mencari informasi serupa sesuai dengan penelitian. Teori-teori yang digunakan penulis berhubungan dengan A*ndroid, push notification, Scheduling*, SCRUM dan sistem notifikasi

### 3.3.2 Kebutuhan Sistem

a. Spesifikasi Aplikasi

Pada tahap ini, dibahas mengenai kebutuhan desain dari aplikasi yang akan dibuat, mulai dari spesifikasi yang dibutuhkan, desain *interface*, hingga desain *UML* yang meliputi *use case*, *sequence*, dan *activity diagram*.

1) Spesifikasi Kebutuhan Perangkat Keras

Perangkat lunak yang digunakan dalam pengambangan aplikasi *mobile* ini dapat dilihat pada Tabel 3.1 dibawah

*Tabel 3.1 Spesifikasi Perangkat Keras*

| no | kebutuhan | Spesifikasi |
|---|---|---|
| 1 | Processor | Intel Core i3 7100u |
| 2 | RAM | 4GB |
| 3 | HardDisk | 1 TB |
| 4 | Mouse | Optical |
| 5 | Keyboard | Qwerty |

2) Spesifikasi Kebutuhan Perangkat Lunak

Perangkat lunak yang digunakan dalam pengambangan aplikasi mobile ini dapat dilihat pada Tabel 3.2 dibawah :

*Tabel 3.2 Spesifikasi Perangkat Lunak*

| no | kebutuhan | Keterangan | fungsi |
|---|---|---|---|
| 1 | Sistem operasi | Ubuntu 20.04 | Sistem operasi yag digunakan |
| 2 | Aplikasi | Android Studio | Membangun Aplikasi |
| | | Github | Github adalah *code hosting* untuk version control project |

3) Spesifikasi Kebutuhan Android

Perangkat yang digunakan dalam pengembangan aplikasi adalah smartphone xiaomi redmi 6A dengen spesifikasi yang dapat dilihat pada Tabel 3.3 dibawah :

*Tabel 3.3 Spesifikasi Perangkat Xiaomi  Redmi 6A*

| No | Kebutuhan | Spesifikasi |
|---|---|---|
| 1 | Prosesor | Mediatek Helio A22 Octa-core 2.0 |
| 2 | Baterai | 3.000 mAh |

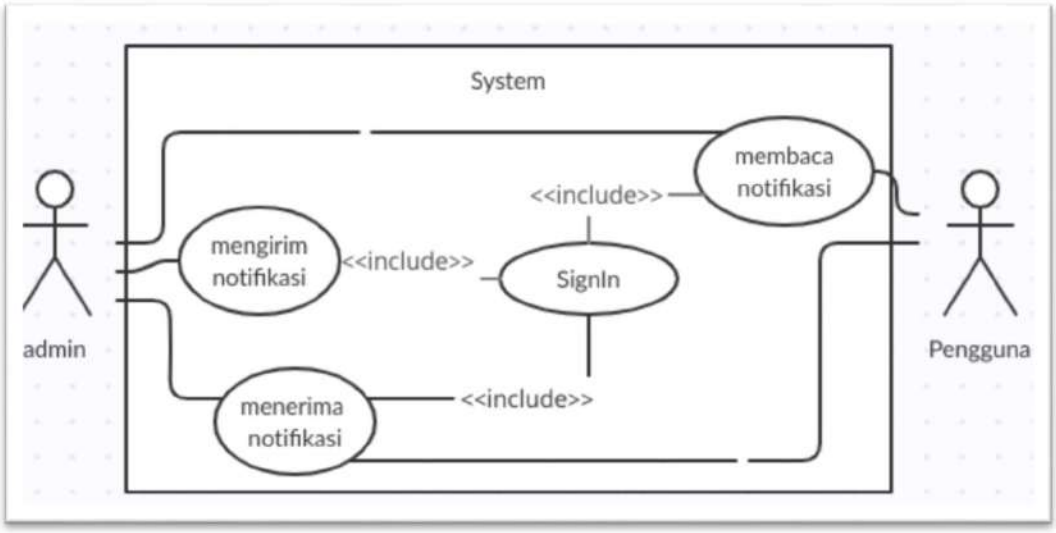
| | | |
|---|---|---|
| 3 | RAM | 2GB |
| 4 | Sistem Operasi | Android Pie 9 |

b. Desain Diagram UML

Pada subbab ini, merancang diagram perilaku yang terdiri dari *use case diagram*, *sequence diagram*, dan *activity diagram* sehingga dapat memberikan gambaran dari aplikasi yang akan dibangun

1) Use case diagram

Tahapan perancangan *diagram* menggunakan *diagram* UML, diantaranya terdiri dari *use case diagram*, *sequence diagram*, dan *activity diagram*

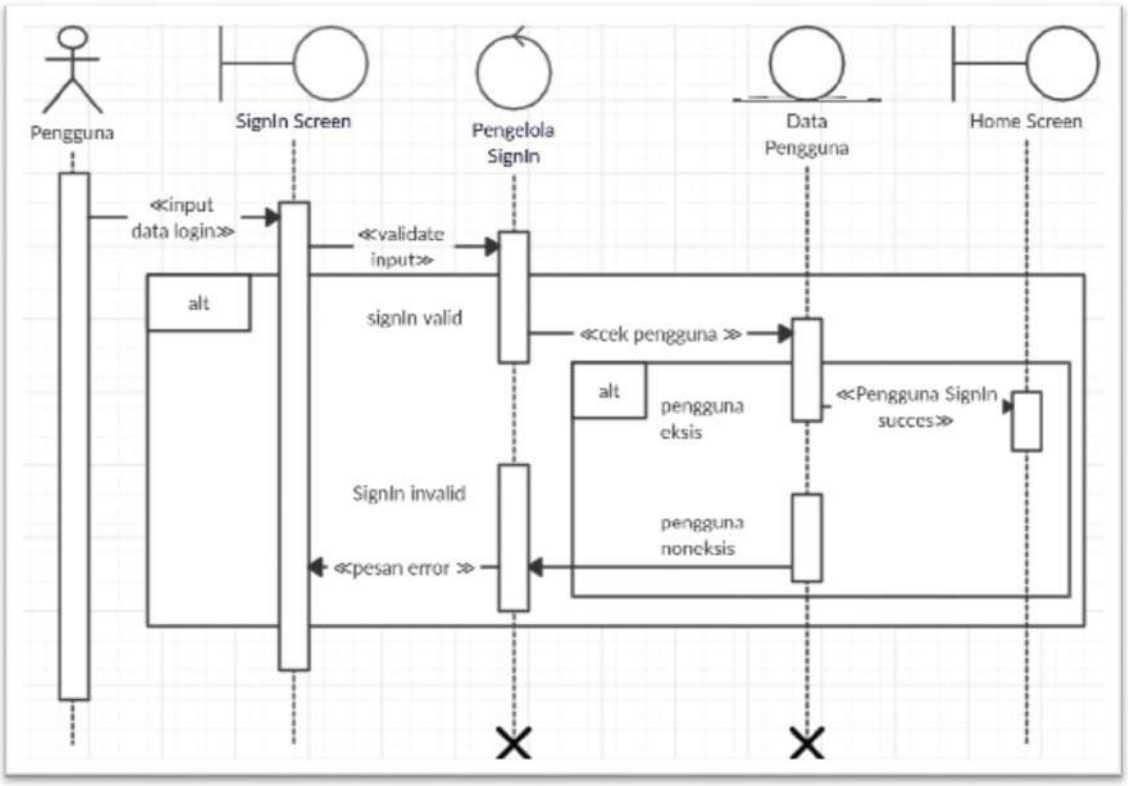



*Gambar 3.3 use case Aplikasi*

Gambar 3.3 merupakan *use case diagram* yang menunujukan kegiatan apa saja yang pengguna dan admin dapat dilakukan, untuk pengguna dapat membaca dan menerima notifikasi dari aplikasi, untuk admin kegiatan yang dapat dilakukan sama seperti pengguna dengan tambahan admin mampu untuk mem-*broadcast* notifikasi ke *pengguna*. Baik admin maupun pengguna diwajibkan untuk login di aplikasi menggunakan email IT Telkom Purwokerto

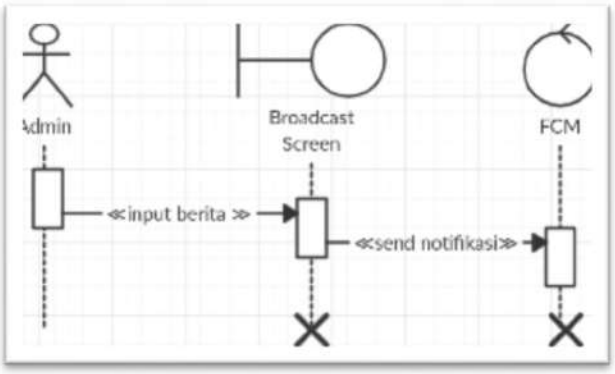
2) Sequence diagram

*Sequence diagram* menggambarkan interaksi antar objek di dalam sistem pada urutan rangkaian waktu. Dibawah ini adalah *sequence* diagram yang ada dalam aplikasi



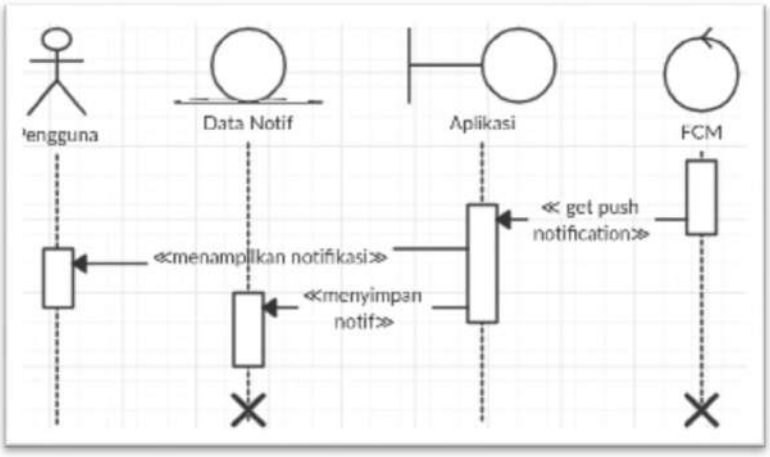

*Gambar 3.4 Sequence SignIn*

Gambar 3.4 menjelaskan tentang proses pengguna dalam melakukan *signIn* ke aplikasi. Data akun login akan di validasi apakah sudah sesuai dengan standar atau belum jika valid, maka akan dilanjutkan ke *database* untuk dilakukan cek *pengguna*, jika pengguna eksis di *database*, pengguna diarahkan menuju halaman *home*, jika pengguna tidak eksis di db / input yang dilakukan tidak *valid*, sistem akan memberikan pesan error sesuai yang ditimbulkan ke *pengguna*.



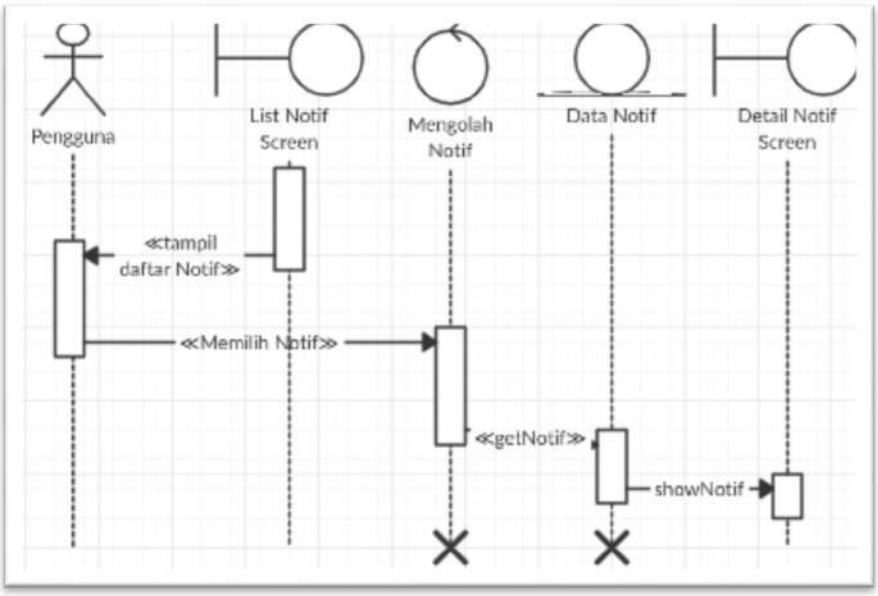

*Gambar 3.5 Sequence Mengirim Notifikasi*

Gambar 3.5 menjelaskan menganai bagaimana admin melakukan *broadcast* notifikasi ke smartphone pengguna yang menginstall aplikasi. Admin melakukan input berita kedalam form yang disediakan sesuai kebutuhan, lalu data berita tersebut dikirim ke *firebase* untuk selanjutnya diproses oleh *firebase cloud* messaging agar sampai ke smartphone pengguna melalui *push notification*




*Gambar 3.6 Sequence Menerima Notifikasi*

Gambar 3.6 menjelaskan proses bagaimana aplikasi dapat menerima *push notification* yang dikirim oleh *server firebase*. Selain menerima notifikasi aplikasi juga menyimpan data notifikasi yang dimuat oleh notifikasi tersebut

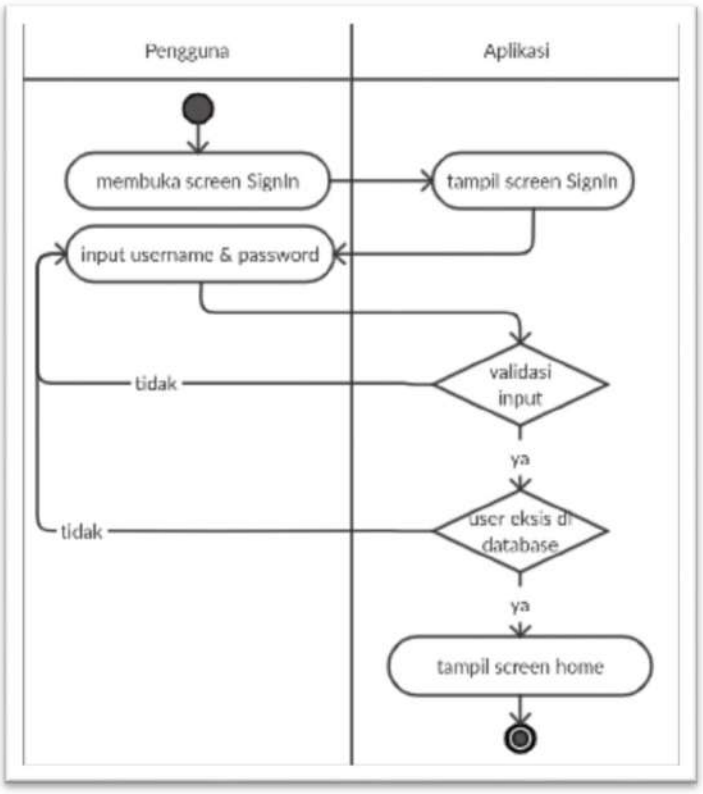



*Gambar 3.7 Sequence Membaca Notifikasi*

Gambar 3.7 menjelaskan proses dari pengguna yang dilakukan agar dapat membaca notifikasi yang tersedia di device smartphonenya. pengguna mengklik salah satu daftar yang ditampilkan, klik yang dilakukan tersebut membawa id dari item tersebut agar sistem dapat melakukan getnotif yang sesuai untuk ditampilkan di halaman detail.

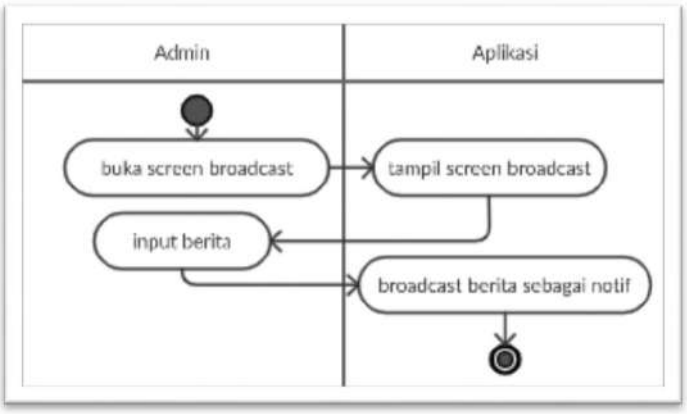
3) Activity diagram

*Activity Diagram* menjabarkan aliran aktifitas dari sistem yang sedang dirancang. *activity diagram* juga dapat menjabarkan proses paralel yang dapat terjadi pada beberapa aktifitas yang berjalan



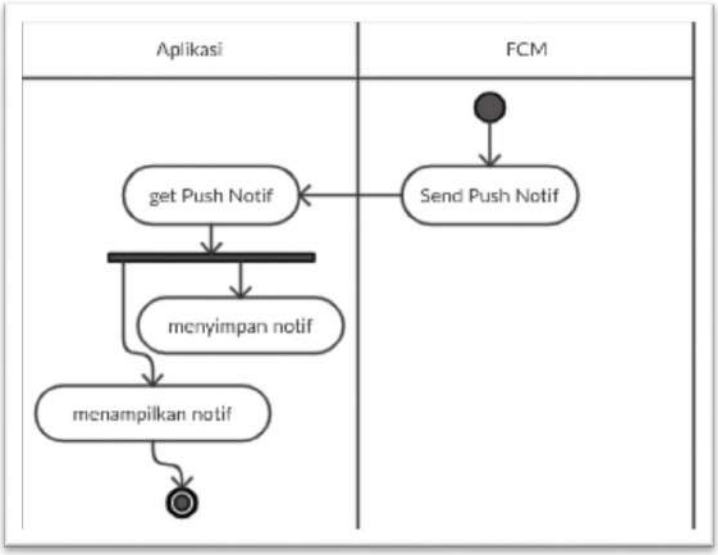

*Gambar 3.8 Activity SignIn*

Gambar 3.8 menjabarkan proses *signIn*, pengguna membuka halaman login lalu melakukan input username dan password. Sebelum data pengguna dikirim, dilakukan proses validasi mengenai input dari pengguna apakah sudah sesuai dengan standar atau belum, seperti minimal panjang password. Jika data pengguna valid, selanjutnya dilakukan pengecekan mengenai ada tidaknya data pengguna yang diinputkan, jika ada, pengguna dapat melanjutkan ke langkah selanjutnya, jika tidak aplikasi akan memberitahukan respon yg sesuai ke *pengguna*



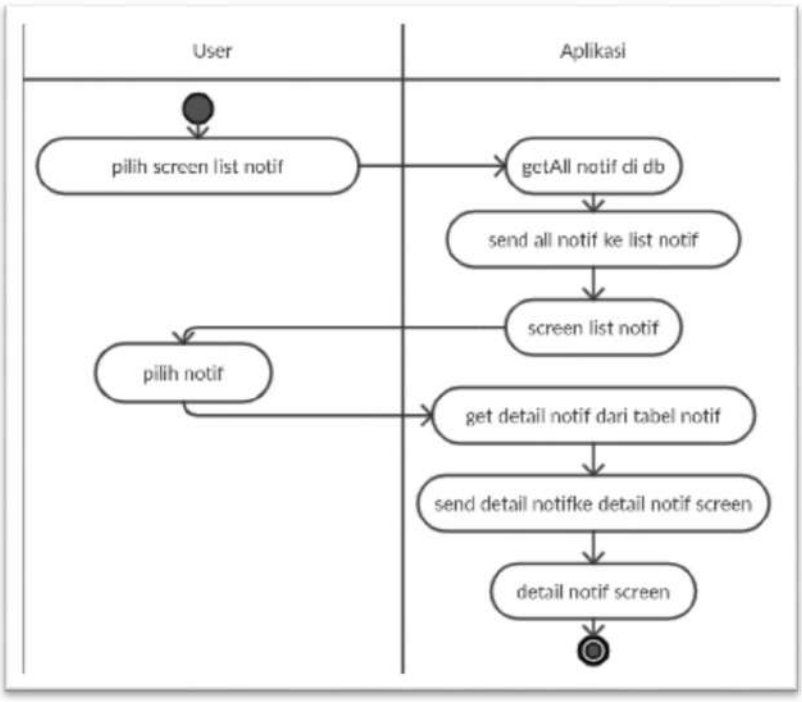

*Gambar 3.9 Activity Mengirim Notifikasi*

Gambar 3.9 menjabarkan bagaimana proses *broadcast* berjalan, admin membuka halaman *broadcast* dan melakukan input berita untuk selanjutnya sistem mengirim berita tersebut sebagai *push notification*




*Gambar 3.10 Activity Menerima Notifikasi*

Gambar 3.10 menjabarkan proses di smartphone pengguna jika ada *push notification* masuk. Fcm mem-*broadcast* pesan dalam bentuk *push* notiification ke pengguna yang dituju, lalu sistem yang menerima *push notification* menampilkan notif tersebut ke *pengguna*, selain itu sistem juga menyimpan notifikasi tersebut kedalam *Room database* aplikasi.



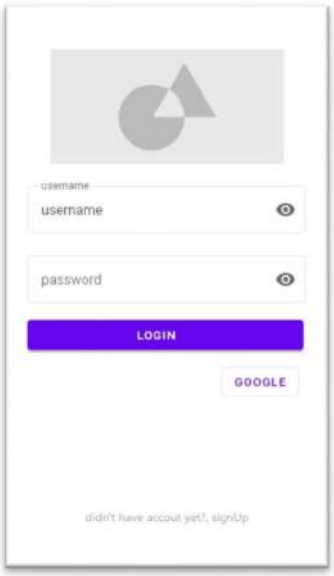

*Gambar 3.11 Activity Membaca Notifikasi*

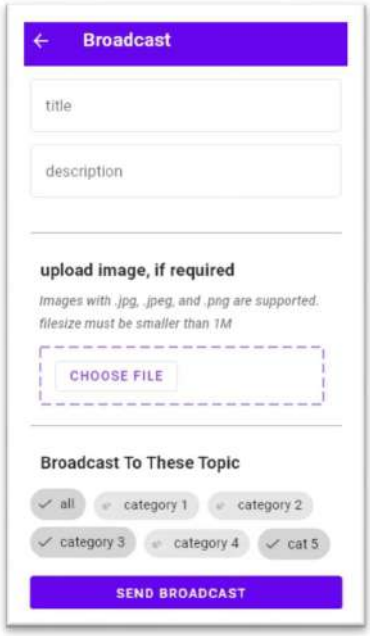
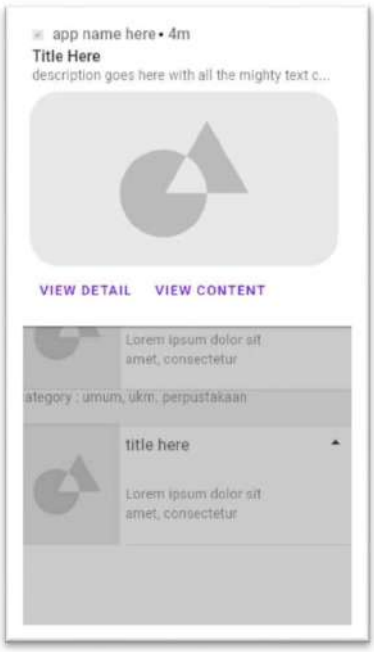
Gambar 3.11 menjabarkan proses dari pengguna dalam membaca pesan yang diterima, pengguna membua halaman home yang berisi daftar notif yang diterima, sistem merespon dengan memproses data notifikasi yang ada di *Room database* dan menampilkan di halsaman *home*, dari sekian notifikasi yang ditampilkan, pengguna dapat memilih salah satu untuk membaca isi dari keseluruhan notifikasi di halaman detail.

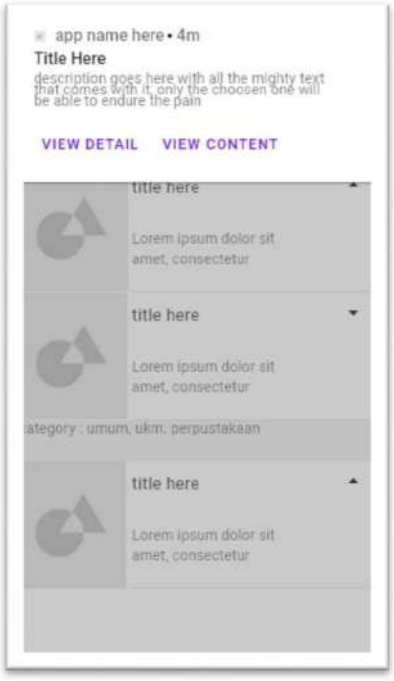
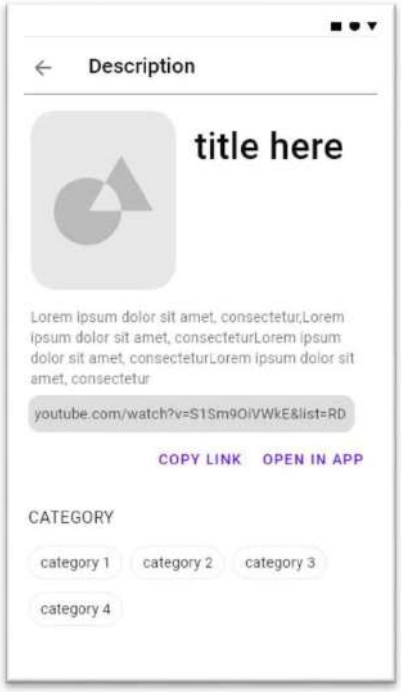
c. Desain Interface

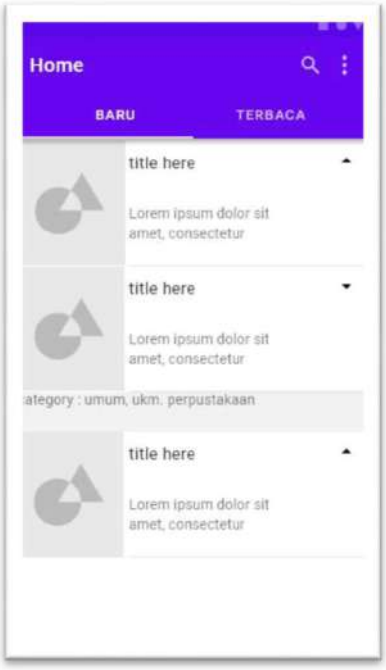
Desain Interface aplikasi notifikasi dapat dilihat pada Tabel 3.4 dibawah ini

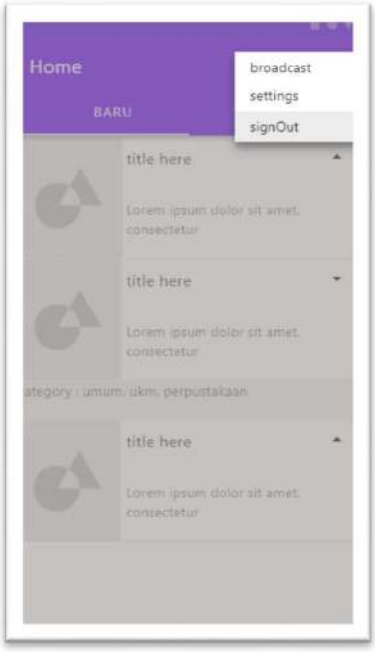
*Tabel 3.4 Desain Interface*

| No | *Interface* | Deskripsi |
|---|---|---|
| 1 |  *Gambar 3.12 Halaman Awal* | Gambar 3.12 menunjukan tampilan awal dari aplikasi, halaman ini adalah halaman yang pertama pengguna lihat ketika pengguna membuka aplikasi untuk pertama kali, halaman seperti ini sering juga disebut dengan *splash screen* |
| 2 |  *Gambar 3.13 Halaman SignIn* | Gambar 3.13 menampilkan halaman untuk *signIn*, di halaman ini terdapat *editText* untuk username dan *editText* untuk password, *button signIn* dan *signIn with Google*, selain melakukan *signIn* secara manual, pengguna juga dapat melakukan *signIn* secara otomotasi menggunakan button *signIn with Google* yang terintegrasi dengan *firebase Authentication* |

| No | *Interface* | Deskripsi |
|---|---|---|
| 3 |  *Gambar 3.14 Halaman Broadcast* | Gambar 3.14 menampilkan halaman *broadcast* yang digunakan untuk mengirim *push notification*. *Form* yang ada Halaman *broadcast* terdiri dari judul berita, deskripsi singkat notifikasi, form optional untuk manambahkan gambar, url *web* yang akan di-*load* sebagai konten di detail, *chip topic* untuk menargetkan pengguna tertentu dari *push notification* dan *button* kirim |
| 4 |  *Gambar 3.15 Notifikasi Dengan Gambar* | Gambar 3.15 menampilkan ilustrasi saat *smartphone* menerima *push notification* yang dikirim oleh admin melalu halaman *broadcast*. Jenis *Notification* pada Gambar 3.15 berupa jenis notifikasi dengan gambar. Terdiri dari judul, deskripsi, gambar yang dimuat, *button view detail* dan *view content*. *View detail* akan mengarahkan *user* ke halaman detail notifikasi seperti yang diilustrasikan pada Gambar 3.17. |

| No | *Interface* | Deskripsi |
|---|---|---|
| 5 |  <br>*Gambar 3.16 Notifikasi Tanpa Gambar* | Gambar 3.16 menampilkan ilustrasi saat *smartphone* menerima *push notification* yang dikirim oleh admin melalu halaman *broadcast*. Jenis *Notification* pada Gambar 3.16 berupa notifikasi tanpa gambar. Terdiri dari judul, deskripsi, gambar yang dimuat, *button view detail* dan *view content*. |
| 6 |  <br>*Gambar 3.17 Halaman Deskripsi Notifikasi* | Gambar 3.17 menampilkan deskripsi secara mendetail dari suatu item. *Url* yang ada juga akan terlihat terdapat opsi untuk menyalin url atau melihatnya melalui aplikasi. Kategori dari aplikasi juga akan terlihat dibagian bawah halaman |

| No | *Interface* | Deskripsi |
|---|---|---|
| 7 |  <br>*Gambar 3.18 Halaman Home* | Gambar 3.20 menampilkan halaman *home* yang terdiri dari dua tab. Tab baru dan tab terbaca, setiap item notifikasi terdiri dari judul, gambar dan deskripsi, dalam item notifikasi ditampilkan juga *category spesifik* dari item tersebut. |
| 8 |  <br>*Gambar 3.19 Halaman Settings* | Gambar 3.21 menampilkan halaman *settings.* Halaman setting memiliki *section notification*. *Section notification* digunakan untuk mem-*filter* notifikasi apa saja yang ditampilkan oleh sistem secara sementara. Sehingga pengguna dapat lebih cepat mencari beberapa notifikasi dalam kategori *topic* tertentu |

| No | *Interface* | Deskripsi |
|---|---|---|
| 9 |  <br>*Gambar 3.20 Halaman Home Dengan Menu Terbuka* | Gambar 3.22 menampilkan halaman *home* dengan opsi menu ter-*click. M*enampilkan pilihan *broadcast*, settings dan signOut menu *broadcast* hanya tersedia untuk *admin*, dan tidak akan muncul untuk pengguna. *Settings* digunakan untuk menuju halaman *settings*, *SignOut* digunakan untuk mengeluarkan / berganti akun pengguna saat ini |

## 3.4 SCRUM Development Process

### 3.4.1 *Product Backlog*

Tahapan pertama adalah menentukan *product backlog*, *product backlog* adalah daftar terurut *becklog item* yang berkemungkinan dibutuhkan dalam produk. *Backlog item* didapat berdasarkan *requierment* yang didapat dari kuisioner, wawancara dari berbagai narasumber.

*Requirement* pada *product backlog* bersifat dinamis sehingga akan terus bertambah apabila mendapatkan *feedback* dari pengguna yang didapat pada masa *review* dan *demo*. Setiap *requirement* yang didapat dijadikan menjadi *becklog item* dan diberikan derajat kepentingan dalam pengerjaannya.

Lalu daftar *backlog item* tersebut ditambahkan kedalam *product backlog* dan diberikan estimasi waktu berapa lama *backlog item* tersebut dapat diselesaikan

### 3.4.2 *Sprint Backlog*

*Sprint backlog* merupakan sekumpulan *product backlog item* yang telah dipilih untuk dikerjakan pada fase *sprint* nanti. Tahapan awal di *sprint backlog* adalah menentukan berapa lama sekali *sprint* berlangsung, estimasi waktu umum antara 1-4 minggu tergantung hasil kesepakatan. Kemudian ditentukan pula berapa banyak *backlog item* yang akan dikerjakan dalam sekali *sprint*.

### 3.4.3 Eksekusi *Sprint*

Tahapan selanjutnya adalah eksekusi *sprint*, dimana pada tahapan ini dapat memakan waktu antara 1-4 minggu per sekali *sprintnya*. Pada fase ini, *developer* mengerjakan setiap *backlog item* yang telah ditentukan pada *sprint backlog* menggunakan aplikasi android studio berdasarkan desain sistem yang sudah dibentuk sebelumnya.

### 3.4.4 Pengujian Aplikasi

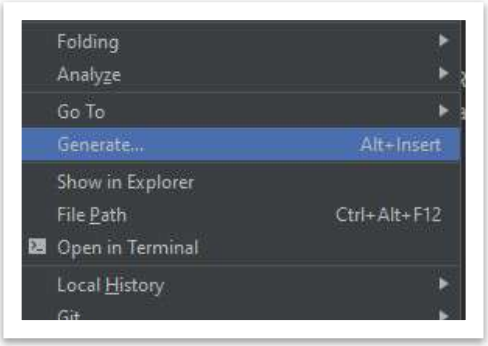
Setiap selesai satu *cycle sprint* dilakukan *test*ing pada aplikasi yang sedang dibangun, tahapan-tahapan *test*ing terdiri dari *small test, medium test,* dan *large test*

a. *Small test*ing

*Small test* ditulis berfokus pada *unit test* yang secara menyeluruh membutuhkan validasi fungsionalitas dan kontrak dari setiap *class* dalam aplikasi. Setiap adanya penambahan atau perubahan *method* dari suatu *class*, buat dan jalankan *unit test* ke *method* tersebut. *Test* yang bergantung dengan *framework* android menggunakan API androidx.*test* . Jika *method* / *class* yang di *test* membutuhkan *resources* android*, maka* aktifkan opsi *includeAndroidResource pada build.gradle(app) agar Unit test dapat menggunakan resource android.*

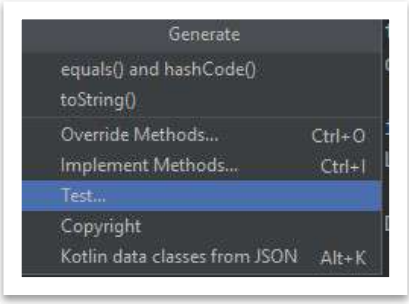
Untuk menguji sebuah *class* / *method* secara *unit* dapat dilakukan dengan cara berikut:

1) Klik kanan pada *class* yang akan dilakukan *test*, lalu pilih generate, ditunjukan seperti pada Gambar 3.21 dibawah:



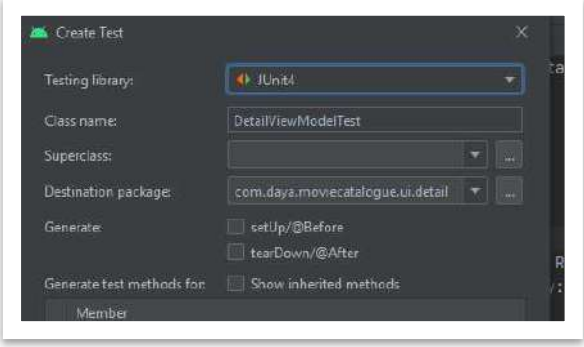

*Gambar 3.21 klik kanan class*

2) Akan muncul dialog generate, Pilih opsi *test*, seperti yang ditunjukan pada Gambar 3.22 dibawah:



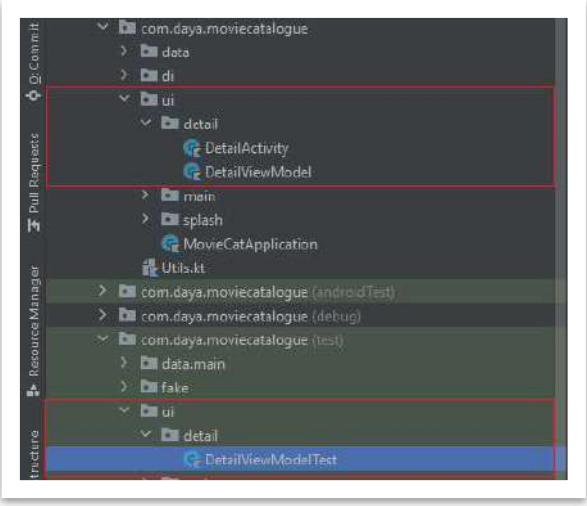

*Gambar 3.22 dialog generate*

3) Akan muncul dialog '*create test*' pasting *test*ing *library* menggunakan J*unit*4, lalu klik *ok,* seperti yang ditunjukan pada Gambar 3.23 dibawah:




*Gambar 3.23 dialog create test*

4) *class test* akan tergenerate pada folder dengan imbuhan '*test*. struktur folder juga akan mencerminkan *class* yang di*test*, hasil dari *generated class test* berupa nama *class* asli+'*test*' dibelakangnya. Seperti yang ditunjukan pada Gambar 3.24 dibawah




*Gambar 3.24 generated test class*

5) jika *class* yang ingin di-*test* membutuhkan *class* lain saat inisialisasi, lakukan *mock* terhadapt *class-class* tersebut sebelum di-*consume* oleh *class* yang ingin di-*test.* Dicontohkan pada kode dibawah:

```
Private lateinit var viewModel :DetailViewModel
Private lateinit var remoteMainRepository : RemoteMainRepository
Private lateinit var localPersistRepository : LocalPersistRepository

@Before
fun setup(){
    remoteMainRepository = mock()
    localPersistRepository = mock()
    viewModel = DetailViewModel(remoteMainRepository, localPersistRepository)
}
```

6) pada proses *testing*, jika *method* tersebut secara internal memanggil *method* lain pada *class* yang dilakukan *mock*, suplai *value return* dengan menggunakan '*whenever'*. Pengujian sukses jika hasil dari *method* yang di*test* sama dengan nilai yang diharapkan. '*assertThat'* digunakan untuk membandingkan hasil dari *method* dan nilai yang diharapkan, 'assertThat' akan bernilai true jika hasil dan nilai yang diharapkan sama. Untuk menguji *method* internal yang *class*-nya dilakukan *mock*, gunakan '*verify'* untuk memastikan bahwa *method internal* tersebut benar-benar tereksekusi, Dicontohkan pada kode dibawah:

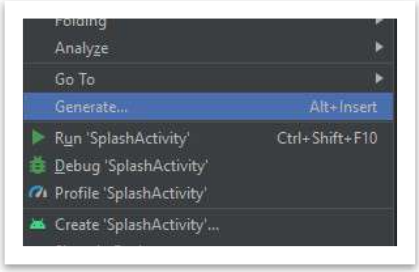
```
@Test
fun `foo`(){
    whenever(remoteMainRepository.getDetailMovie(dummyMovie.id).thenReturn(dummyMovie)
    viewModel.submitMovie(dummyMovie.id)

    assertThat(viewModel.observeMovie.getOrAwaitValue()).isEqualTo(Resource.succes(dummyMovie)
}
```

Hasil dari *small testing* berupa sebuah *class* didalam folder *test*, *class testing* tersebut akan memiliki berbagai *method* beranotasi *test* dimana *method-method* tersebut menguji *method* dari *class* yang diuji

b. *Medium test*ing

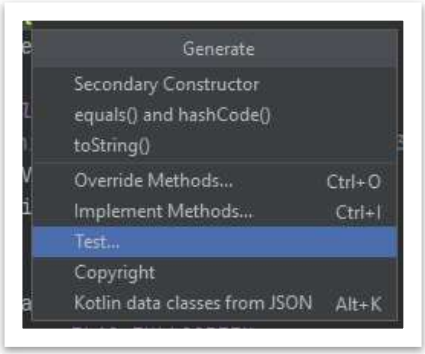
Sebagai tambahan *unit test* dari aplikasi dengan menjalankan *small test. Medium test* ditulis dalam bentuk *scenario* yang merepresentasikan langkah-langkah atau tindakan yang dapat dilakukan di aplikasi. Untuk pengujian secara *medium* dapat dilakukan dengan cara berikut:

1) Klik kanan pada *UI* yang akan dibuat diuji, lalu pilih *generate,* seperti halnya ditunjukan pada Gambar 3.25 dibawah:



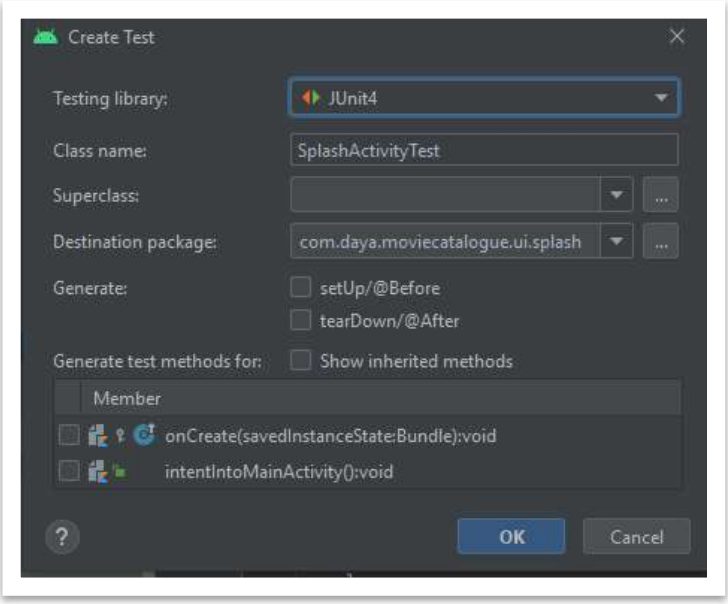

*Gambar 3.25 klik kanan pada activity*

2) Pilih opsi *test*, seperti pada Gambar 3.26 dibawah



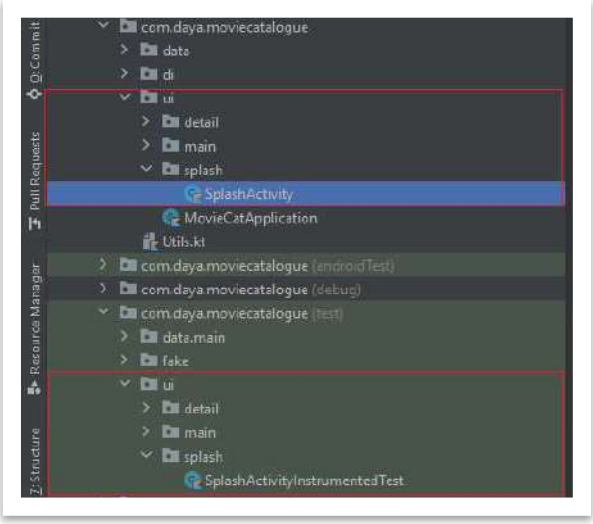

*Gambar 3.26 dialog generate*

3) Akan muncul dialog '*create test*' pastikan *testing library* menggunakan J*unit*4, lalu klik ok, seperti yang ditunjukan pada Gambar 3.27 dibawah




*Gambar 3.27 dialog create test*

4) *Class test* akan ter-*generate* pada folder *androidTest* dengan imbuhan '*test'*. struktur folder juga akan mencerminkan *class* yang di-*test*, hasil dari *generated class test* berupa nama *class* asli ditambah dengan kata '*test*' dibelakangnya. Seperti yang ditunjukan pada Gambar 3.28 dibawah:




*Gambar 3.28 generated test class untuk splashActivity*

5) *Testing* mencerminkan suatu langkah dari aplikasi, pada kasus dibawah, *test* memastikan bahwa *supportActionBar* tidak terlihat, dicontohkan pada codingan dibawah:

```
@RunWith(AndroidJunit4::class)
Class bazz{
    @get:Rule
    Val activityRule = activityScenarioRule<SplashActivity>()

    @Test
    fun `foo`(){
        val scenario = activityRule.scenario
        scenario.onActivity{
            val toolbar = it.supportActionBar
            assertThat(toolbar).isNotNull()
            assertThat(toolbar).isNotEqualTo(!toolbar!!.isShowing)
        }
    }
}
```

Hasil dari *medium test*ing berupa sebuah *class* didalam folder *androidTest*, *class testing* tersebut akan memiliki berbagai *method* beranotasi *test* dimana *method-method* tersebut memvalidasi suatu tindakan kecil dari aplikasi

c. *Large test*ing

Dalam *large test*, *test*ing dilakukan dengan bantuan manusia dalam bentuk *black box test*ing, *black box* ini digunakan untuk menguji kepuasan pengguna dalam mengggunakan metode kusioner. Hasil *test* ini akan menunjukan apakah aplikasi dapat dikatakan baik atau belum, berdasarkan nilai akhir dari perhitungan bobot interval likert sangat baik, baik, dan cukup dikategorikan sebagai aplikasi berjalan dengan baik. Dan interval sangat tidak baik dan tidak baik dikategorikan sebagai aplikasi belum berjalan dengan balik

Selanjutnya dilakukan *review* apakah ada masukan berupa *feedback* atas *functional requirement* dari aplikasi yang didemokan. *feedback* pada suatu fungsi maka feedback tersebut akan dipertimbangkan untuk dimasukan kedalam *backlog* dan akan ditambahkan pada *cycle* selanjutnya dari *sprint*. Pertimbangan dimasukan tidaknya suatu *feedback* diputuskan berdasarkan banyak tidaknya feedback tersebut dikemukakan oleh responden

### 3.4.5 Analisis Scrum

Analisis scrum dilakukan berdasarkan *burndown chart* yang dibuat selama fase *sprint* berlangsung. Selama fase *sprint*, pengembang, mengisi tabel *burndown chart* dimana setiap data pekerjaan yang selesai dilakukan didokumentasikan untuk perhitungan *burndown chart*.