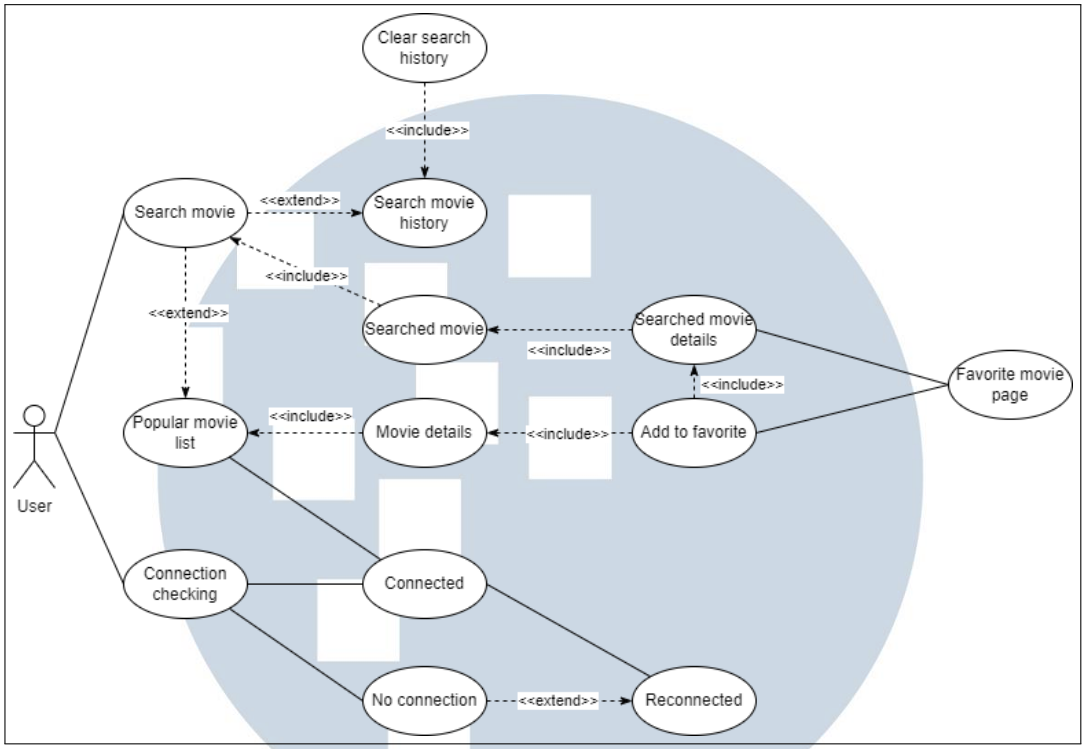
BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Selama melaksanakan kerja magang di PT Global Loyalty Indonesia lalu ditempatkan pada divisi *E-Commerce Application Development* sebagai *Android Developer Internship* yang dipimpin langsung oleh Bapak Erik Gunawan selaku *Android Developer Specialist*. Program magang yang dilakukan ini disebut sebagai ”GLI Academy” yang merupakan program pelatihan khusus untuk mengembangkan aplikasi *mobile* maupun *website*. Selama masa magang, komunikasi yang dilakukan secara langsung dan tidak langsung yang melalui aplikasi Telegram. Selama pelaksanaan magang, setiap partisipan diberikan *repository* pada BitBucket untuk mengumpulkan tugas mingguan. Posisi *Android Developer* sendiri bertugas untuk mengembangkan aplikasi Alfagift yang sudah ada sebelumnya, dalam pengembangan aplikasi ini, *Android Developer* harus mengetahui dan memperhatikan alur kerja dari aplikasi Alfagift, struktur folder aplikasi, *Android Architecture* yang digunakan serta mengedepankan *Clean Architecture* agar dapat lebih mudah di kelola oleh siapapun yang berada di *Android Developer Team*.

3.2 Tugas yang Dilakukan

Selama melaksanakan program kerja magang di PT Global Loyalty Indonesia, setiap partisipan diberikan tugas mingguan dengan tujuan pemahaman yang berbeda-beda, salah satunya merupakan *project* ”The Movie App” yang merupakan *project* yang saat ini dilakukan. Berikut merupakan Use Case Diagram dan penjelasan dari fitur-fitur dari aplikasi ”The Movie App”



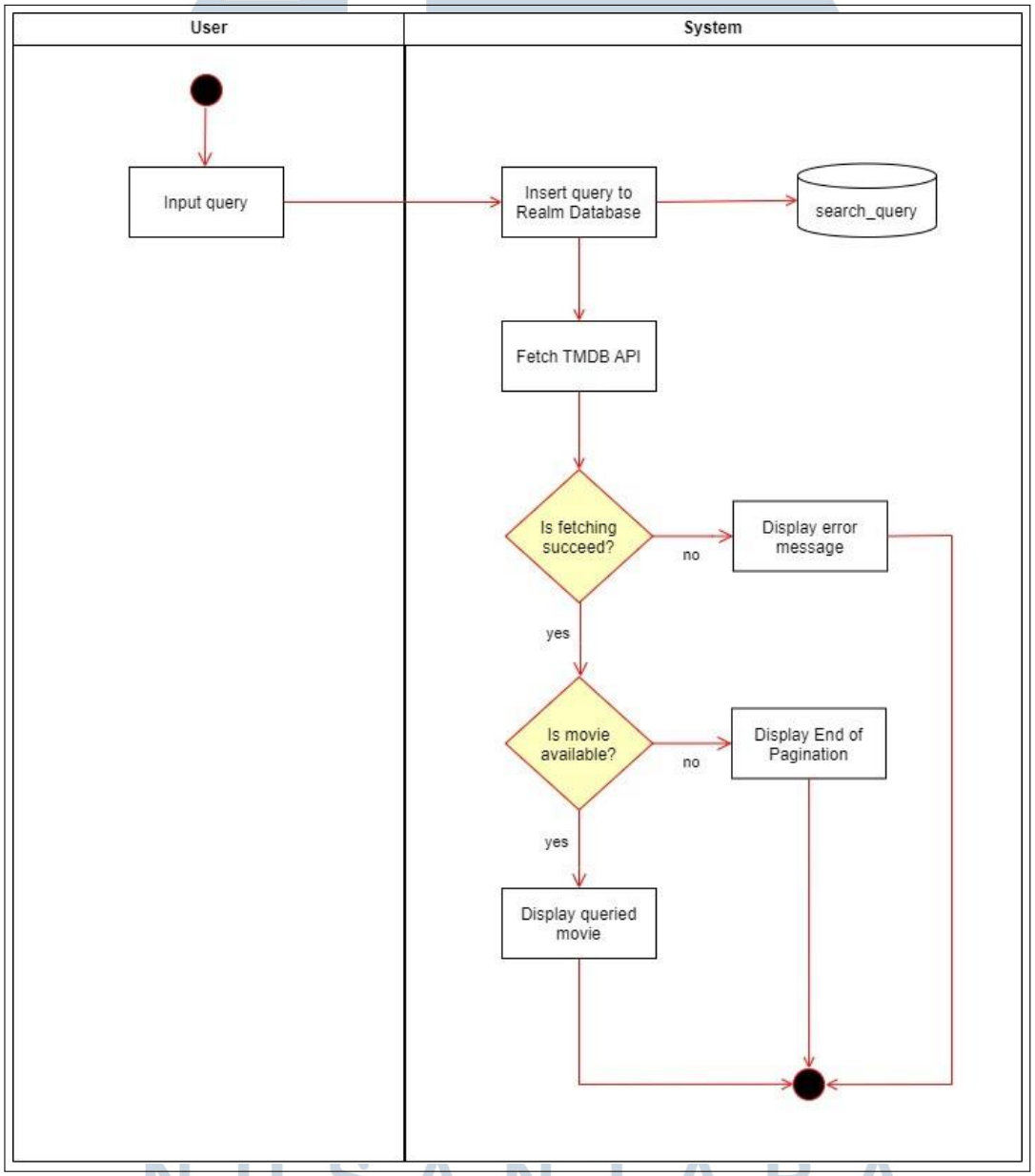
Gambar 3.1. *Use Case Diagram*

Use Case Diagram digunakan untuk menggambarkan bagaimana jalannya aplikasi ”The Movie App”. Pertama-tama ketika pengguna menjalankan aplikasi terdapat pengecekan koneksi, jika pengguna memiliki koneksi maka dapat langsung memunculkan list dari film populer, dan jika tidak terdapat koneksi maka pengguna dapat *reconnecting*, ketika sudah memiliki koneksi maka terdapat list populer film. Fitur search tetap dapat digunakan dalam keadaan ada atau tidak ada koneksi, namun jika tidak ada koneksi, list film yang dicari juga tidak dapat muncul dan pengguna harus *reconnecting* kembali. Ketika pengguna mengklik poster film baik dari halaman utama atau HomeFragment maupun halaman dari SearchMovie maka pengguna diarahkan untuk ke halaman MovieDetailActivity yang memunculkan detail setiap film yang user telah pilih atau klik sebelumnya, pada halaman MovieDetailActivity, pengguna dapat klik tombol favoritkan film yang nantinya film-film yang telah difavoritkan oleh pengguna dapat muncul pada halaman FavoriteFragment. Pada halaman FavoriteFragment pengguna juga dapat melihat detail film dengan mengklik salah satu poster dari film yang diinginkan, dan pengguna dibebaskan untuk dapat klik favorit atau hilangkan dari list film favoritnya jika sudah tidak diinginkan.

3.2.1 Home Page

A. Search Film

Berikut merupakan *activity diagram* untuk fitur *search* yang dapat dilihat pada Gambar 3.2.



Gambar 3.2. *Activity Diagram* Fitur *Search* Film

Berikut merupakan potongan kodenya:

```
private fun searchMovie(searchView: SearchView) {
        searchView.setOnQueryTextListener(object : SearchView.
    OnQueryTextListener {
            override fun onQueryTextSubmit(query: String?):
    Boolean {
                if (!query.isNullOrEmpty()) {
                    val currentQuery = binding.searchQueryTextView
    .text.toString().trim()
                    val newQuery = if (currentQuery.isNotEmpty())
    "$query | $currentQuery" else query
                    val limit = newQuery.take(40)
                    binding.searchQueryTextView.text = limit
                    observeQuerySubmitted()

                    movieNetworkViewModel.searchMovies(query)
                    searchBarViewModel.addQuery(query)

                    searchView.clearFocus()
                    return true
                } else {
                    searchView.clearFocus()
                    Toast.makeText(
                        requireContext(),
                        getString(R.string.enter_movie_title),
                        Toast.LENGTH_SHORT
                    ).show()
                }
                return true
            }

            override fun onQueryTextChange(newText: String?):
    Boolean {
                if (newText.isNullOrEmpty()) {
                    movieNetworkViewModel.loadMovies(
    requireContext())
                    collectLatestQueryChanged()
                }
                return true
            }
        })

        searchView.setOnCloseListener {
```

```
            searchView.clearFocus()
            false
        }

        binding.clearHistoryTextView.setOnClickListener {
            searchBarViewModel.clearSearchHistory()
            searchView.clearFocus()
        }
    }

    private fun observeQuerySubmitted() {
        binding.apply {
            viewObserveQueryVisible(searchQueryTextView)
            viewObserveQueryVisible(searchHistoryTextView)
            viewObserveQueryVisible(clearHistoryTextView)
        }
        binding.clearHistoryTextView.setOnClickListener {
            searchBarViewModel.clearSearchHistory()
            binding.apply {
                searchQueryTextView.text = ""
                viewObserveQueryGone(searchQueryTextView)
                viewObserveQueryGone(searchHistoryTextView)
                viewObserveQueryGone(clearHistoryTextView)
            }
        }
    }

    private fun collectLatestQueryChanged() {
        binding.apply {
            moviePosterRecyclerView.visibility = View.VISIBLE
            noMoviesLayout.visibility = View.GONE
            searchHistoryTextView.visibility = View.GONE
            searchQueryTextView.visibility = View.GONE
            clearHistoryTextView.visibility = View.GONE
        }
    }
```

Kode 3.1: HomeFragment

Pada fitur *search* film, *Search View* digunakan sebagai media pengguna dalam mencari film yang diinginkan, pertama-tama setelah pengguna selesai mencari film yang diinginkan maka sistem akan menyimpan setiap *query* pengguna ke dalam *Realm Database*, setelah sistem akan mengambil data dari TMDB API, jika proses pengambilan datanya gagal maka sistem akan memunculkan pesan errornya, jika sukses maka sistem akan mengecek kembali ketersediaan film dengan *query* tersebut di dalam TMDB API, jika tidak ditemukan maka akan menampilkan *layout End of Pagination*, namun jika ditemukan maka film dengan *query* tersebut muncul. Pada fitur *search movie* ini, *Realm Database* juga menampilkan *search history* sejumlah 10 buah *query* terbaru, dan pengguna dapat menghapus *search history* tersebut.

Pada potongan kode diatas merupakan bagian dari fungsi searchMovie pada halaman HomeFragment yang digunakan untuk menangani Search View atau Search Bar ketika terdapat tindakan dari pengguna, jika pengguna mengklik Search View dan memasukan sebuah kalimat maka akan tersimpan sebagai query, query tersebut juga terdapat pengecekan, jika query tersebut tidak kosong maka akan tersimpan pada Realm Database dan ditampilkan dengan bentuk ”$query — $currentquery” yang dimana akan dibatasi sejumlah 10 query saja yang akan ditampilkan. Jika pengguna telah selesai menuliskan kalimat yang ingin di cari maka muncul film dengan judul yang relevan dengan kalimat yang dimasukkan pengguna / query, dan jika pengguna klik tombol x maka keyboard akan tertutup otomatis dan langsung kembali ke halaman Home dengan menggunakan searchView.clearFocus(), selain itu jika pengguna klik tulisan ”Clear” maka semua query yang telah pengguna tuliskan akan hilang semua dengan menggunakan searchBarViewModel.clearSearchHistory()

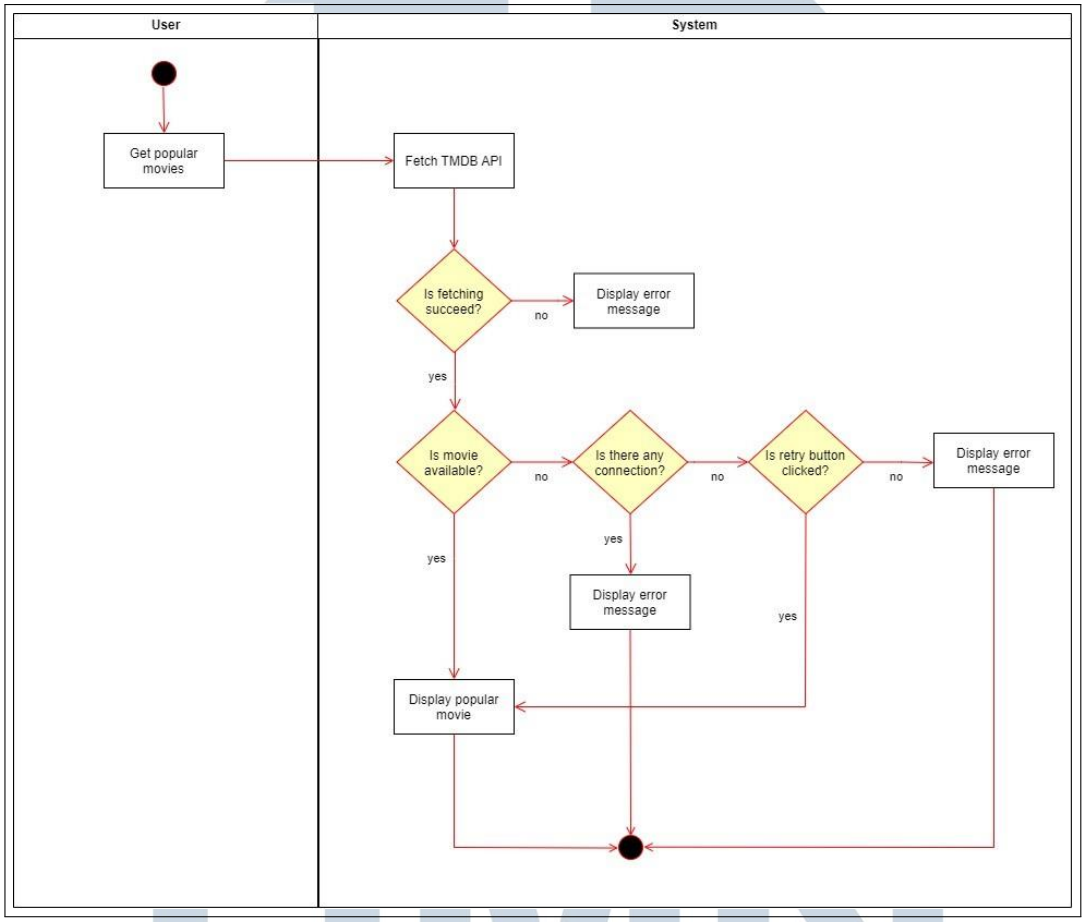
```
package com.example.moviedb.presentation.home

@HiltViewModel
class SearchBarViewModel @Inject constructor(
    private val searchUseCase: SearchUseCase
) : ViewModel() {

    private val maxHistorySize = 10
    private val searchHistory = mutableListOf<String>()

    fun clearSearchHistory() {
        viewModelScope.launch {
            searchUseCase.clearSearchHistory()
        }
    }

    fun addQuery(query: String) {
        viewModelScope.launch {
            searchUseCase.addSearchQuery(query)
            searchHistory.add(0, query)
            if (searchHistory.size > maxHistorySize) {
                val removedQuery = searchHistory.removeAt(
    maxHistorySize)
                searchUseCase.removeSearchQuery(removedQuery)
            }
        }
    }
}
```

Kode 3.2: SearchBarViewModel

Pada kode diatas menunjukan potongan kode dari SearchBarViewModel yang terdapat fungsi clearSearchHistory() yang dapat digunakan untuk menghapus seluruh query yang telah pengguna tuliskan, sementara fungsi addQuery(query: String) digunakan untuk menyimpan setiap query pengguna ke dalam Realm Database.

B. List Film Populer

Berikut merupakan *activity* diagram untuk fitur *search* yang dapat dilihat pada Gambar 3.3.



Gambar 3.3. *Activity* Diagram *List* Film Populer

Ketika pengguna pertama kali menggunakan aplikasi The Movie App maka terdapat *list* film yang sedang populer saat ini, *list* film populer didapatkan melalui TMDB API, setelah itu terdapat pengecekan untuk memastikan *list* film populer tersebut berhasil didapatkan, jika gagal maka maka terdapat informasi bahwa sistem gagal memunculkan *list* film populer dari TMDB API, jika berhasil maka terdapat pengecekan lagi untuk memastikan ketersediaan film dari TMDB API tersebut, jika tidak tersedia maka terdapat pengecekan lebih lanjut yaitu pengecekan terhadap koneksi internet pengguna, jika pengguna tidak memiliki koneksi internet maka pengguna akan di arahkan untuk klik tombol *retry*, jika pengguna telah memiliki koneksi internet dan sudah klik tombol *retry* maka *list* film populer akan

muncul, namun jika semua pengecekan tersebut gagal akan menampilkan informasi alasan mengapa terjadi kegagalan. *List* film populer ditampilkan menggunakan *Recycler View* pada *Home Fragment*, penggunaan *Home Fragment* dipilih karena untuk memperingan kinerja aplikasi dengan mengurangi perpindahan navigasi antar *Activity*, selain itu penggunaan *Fragment* juga dipilih dikarenakan terdapat fitur menggeserkan halaman antar *fragment* menggunakan *View Pager*.

Berikut merupakan potongan kodenya:

```
1  package com.example.moviedb.presentation.home
2
3
4  @AndroidEntryPoint
5  class HomeFragment : Fragment(), MoviePosterAdapter.
       OnItemClickListener {
6      private lateinit var binding: FragmentHomeBinding
7      private val movieNetworkViewModel: MovieNetworkViewModel by
       viewModels()
8      private lateinit var moviePosterAdapter: MoviePosterAdapter
9      private lateinit var loadingStateAdapter: LoadingStateAdapter
10     private val searchBarViewModel: SearchBarViewModel by
       viewModels()
11
12     override fun onItemClick(item: MovieItem) {
13         val intent = Intent(requireContext(), MovieDetailsActivity
       ::class.java)
14         intent.putExtra(MovieDetailsActivity.EXTRA_MOVIE, item)
15         startActivity(intent)
16     }
17
18     override fun onCreateView(
19         inflater: LayoutInflater, container: ViewGroup?,
20         savedInstanceState: Bundle?
21     ): View {
22         binding = FragmentHomeBinding.inflate(inflater, container,
       false)
23
24         movieNetworkViewModel.loadMovies(context = requireContext
       ())
25
26         return binding.root
27     }
28
29     override fun onViewCreated(view: View, savedInstanceState:
```

```
Bundle?) {
30      super.onViewCreated(view, savedInstanceState)
31
32      endOfPagination()
33      observeViewModel()
34      initAdapter()
35      initRetryButton()
36      searchMovie(searchView = binding.searchView)
37  }
38
39  private fun endOfPagination() {
40      moviePosterAdapter = MoviePosterAdapter(this)
41      binding.moviePosterRecyclerView.apply {
42          adapter = moviePosterAdapter
43          layoutManager = GridLayoutManager(requireContext(), 2)
.also {
44              it.spanSizeLookup = object : GridLayoutManager.
SpanSizeLookup() {
45                  override fun getSpanSize(position: Int): Int {
46                      return if (position == moviePosterAdapter.
itemCount) 2 else 1
47                  }
48              }
49          }
50      }
51  }
52
53  private fun observeViewModel() {
54      movieNetworkViewModel.errorMessage.observe(
viewLifecycleOwner) { errorMessage ->
55          binding.noInternetLayout.noInternetLayout.visibility =
56              if (errorMessage != null) View.VISIBLE else View.
GONE
57      }
58
59      movieNetworkViewModel.movieData.observe(viewLifecycleOwner
) { pagingData ->
60          lifecycleScope.launch {
61              pagingData?.collectLatest { pagingData ->
62                  moviePosterAdapter.submitData(pagingData)
63              }
64          }
65      }
```

```
66      }
67
68      private fun initAdapter() {
69          loadingStateAdapter = LoadingStateAdapter {
        moviePosterAdapter.retry() }
70          binding.moviePosterRecyclerView.adapter =
        moviePosterAdapter.withLoadStateFooter(
71              footer = loadingStateAdapter
72          )
73
74          lifecycleScope.launch {
75              moviePosterAdapter.loadStateFlow.collect {
76                  when (val loadState = it.source.refresh) {
77                      is LoadState.Loading -> {
78                          recyclerVisible(View.VISIBLE)
79                          noInternetVisible(View.GONE)
80                      }
81
82                      is LoadState.Error -> {
83                          when (loadState.error) {
84                              is HttpException, is
        UnknownHostException -> {
85                                  recyclerVisible(View.GONE)
86                                  noInternetVisible(View.VISIBLE)
87                              }
88
89                              else -> {
90                                  recyclerVisible(View.VISIBLE)
91                                  noInternetVisible(View.GONE)
92                              }
93                          }
94                      }
95
96                      else -> {
97                          recyclerVisible(View.VISIBLE)
98                          noInternetVisible(View.GONE)
99                      }
100                 }
101             }
102         }
103     }
104
105     private fun initRetryButton() {
```

```
106         binding.noInternetLayout.retryButton.setOnClickListener {
107             moviePosterAdapter.retry()
108         }
109     }
110 
111     private fun recyclerVisible(visibility: Int) {
112         binding.moviePosterRecyclerView.visibility = visibility
113     }
114 
115     private fun noInternetVisible(visibility: Int) {
116         binding.noInternetLayout.noInternetLayout.visibility =
    visibility
117     }
118 
119     private fun searchMovie(searchView: SearchView) {
120         searchView.setOnQueryTextListener(object : SearchView.
    OnQueryTextListener {
121             override fun onQueryTextSubmit(query: String?):
    Boolean {
122                 if (!query.isNullOrEmpty()) {
123                     val currentQuery = binding.searchQueryTextView
    .text.toString().trim()
124                     val newQuery = if (currentQuery.isNotEmpty())
    "$query | $currentQuery" else query
125                     val limit = newQuery.take(40)
126                     binding.searchQueryTextView.text = limit
127                     observeQuerySubmitted()
128 
129                     movieNetworkViewModel.searchMovies(query)
130                     searchBarViewModel.addQuery(query)
131 
132                     searchView.clearFocus()
133                     return true
134                 } else {
135                     searchView.clearFocus()
136                     Toast.makeText(
137                         requireContext(),
138                         getString(R.string.enter_movie_title),
139                         Toast.LENGTH_SHORT
140                     ).show()
141                 }
142                 return true
143             }
```

```
144
145         override fun onQueryTextChange(newText: String?):
    Boolean {
146             if (newText.isNullOrEmpty()) {
147                 movieNetworkViewModel.loadMovies(
    requireContext())
148                 collectLatestQueryChanged()
149             }
150             return true
151         }
152     })
153
154     searchView.setOnCloseListener {
155         searchView.clearFocus()
156         false
157     }
158
159     binding.clearHistoryTextView.setOnClickListener {
160         searchBarViewModel.clearSearchHistory()
161         searchView.clearFocus()
162     }
163 }
164
165 private fun observeQuerySubmitted() {
166     binding.apply {
167         viewObserveQueryVisible(searchQueryTextView)
168         viewObserveQueryVisible(searchHistoryTextView)
169         viewObserveQueryVisible(clearHistoryTextView)
170     }
171     binding.clearHistoryTextView.setOnClickListener {
172         searchBarViewModel.clearSearchHistory()
173         binding.apply {
174             searchQueryTextView.text = ""
175             viewObserveQueryGone(searchQueryTextView)
176             viewObserveQueryGone(searchHistoryTextView)
177             viewObserveQueryGone(clearHistoryTextView)
178         }
179     }
180 }
181
182 private fun collectLatestQueryChanged() {
183     binding.apply {
184         moviePosterRecyclerView.visibility = View.VISIBLE
```

```
185                 noMoviesLayout.visibility = View.GONE
186                 searchHistoryTextView.visibility = View.GONE
187                 searchQueryTextView.visibility = View.GONE
188                 clearHistoryTextView.visibility = View.GONE
189             }
190         }
191 }
```

Kode 3.3: HomeFragment

Pada HomeFragment terdapat fungsi ketika poster film di klik maka terdapat penanganan untuk membuat pengguna pindah ke halaman MovieDetailActivity yang dapat memunculkan detail film yang pengguna pilih atau klik, setelah itu terdapat fungsi endOfPagination() yang digunakan sebagai penanda bahwa pengguna telah mencapai halaman terakhir dari list film tersebut, lalu terdapat fungsi observeViewModel() yang digunakan untuk mengecek koneksi pengguna, jika pengguna tidak memiliki koneksi maka terdapat tombol ”Retry” yang dapat digunakan untuk meraih data film kembali ketika pengguna telah memiliki koneksi internet, selain itu terdapat fungsi initAdapter() yang berfungsi untuk memuat kembali film ketika pengguna telah memiliki koneksi internet dan sudah klik tombol ”Retry”, terdapat fungsi initRetryButton() yang berfungsi untuk memunculkan poster film tersebut. Fungsi recyclerVisible dibuat karena terdapat beberapa kali pemanggilan fungsi yang sama untuk memunculkan poster film, hal ini dilakukan untuk mengefisiensikan kode sehingga terlihat lebih rapi dan lebih *reusable*, hal ini juga berlaku untuk fungsi noInternetVisible(). Terdapat fungsi searchMovie yang digunakan untuk menangani tindakan pengguna terhadap penggunaan fitur Search Movie yang telah disebutkan pada potongan kode sebelumnya.

```
1  package com.example.moviedb.presentation.home.loadstate.adapter
2
3  class LoadingStateAdapter(
4      private val retry: () -> Unit
5  ) : LoadStateAdapter<LoadingStateViewHolder>() {
6      override fun onCreateViewHolder(parent: ViewGroup, loadState:
       LoadState) : LoadingStateViewHolder {
7          return LoadingStateViewHolder.create(parent, retry)
8      }
9
10     override fun onBindViewHolder(holder: LoadingStateViewHolder,
       loadState: LoadState) {
```

```
        if (loadState is LoadState.Error) {
            holder.bind(loadState.error)
        } else {
            holder.bind(null)
        }
    }
}
```

Kode 3.4: LoadStateAdapter

```
package com.example.moviedb.presentation.home

@HiltViewModel
class MovieNetworkViewModel @Inject constructor(private val
    repository: MovieNetworkRepository) : ViewModel() {
    private val _movieData = MutableLiveData<Flow<PagingData<
    MovieItem>>>()
    val movieData: LiveData<Flow<PagingData<MovieItem>>?> get() =
    _movieData

    private val _errorMessage = MutableLiveData<String>()
    val errorMessage: LiveData<String> get() = _errorMessage

    private val _isOfflineMode = MutableLiveData<Boolean>()

    fun loadMovies(context: Context) {
        viewModelScope.launch {
            try {
                _movieData.value = repository.getMovies().cachedIn
    (viewModelScope)
                updateConnectionStatus(false)
            } catch (e: Exception) {
                updateConnectionStatus(true)
                _errorMessage.value = context.getString(R.string.
    no_internet_connection)
            }
        }
    }

    private fun updateConnectionStatus(isOffline: Boolean) {
        _isOfflineMode.value = isOffline
    }

    fun searchMovies(query: String) : Flow<PagingData<MovieItem>>
```

```
    {
        viewModelScope.launch {
            try {
                _movieData.value = repository.searchMovies(query).
    cachedIn(viewModelScope)
                updateConnectionStatus(false)
            } catch (e: Exception) {
                updateConnectionStatus(true)
                _errorMessage.value = e.message
            }
        }
        return repository.getMovies().cachedIn(viewModelScope)
    }
}
```

Kode 3.5: MovieNetworkViewModel

LoadStateAdapter dan MovieNetworkViewModel berfungsi untuk memunculkan seluruh poster film ketika tombol ”Retry” sudah di klik oleh pengguna dan sudah memiliki koneksi.

```
package com.example.moviedb.data.network.repository

private const val PAGE_SIZE = 20
private const val MAX_PAGES = 5

class MovieNetworkRepository @Inject constructor(private val
    movieApi: MovieApi) {
    fun getMovies(): Flow<PagingData<MovieItem>> {
        return Pager(
            config = PagingConfig(
                pageSize = PAGE_SIZE,
                maxSize = PAGE_SIZE * MAX_PAGES,
                enablePlaceholders = false
            ),
            pagingSourceFactory = { MoviePagingSource(movieApi,
    MAX_PAGES) }
        ).flow
    }

    fun searchMovies(query: String): Flow<PagingData<MovieItem>> {
        return Pager(
            config = PagingConfig(
```

```
                pageSize = PAGE_SIZE,
                maxSize = PAGE_SIZE * MAX_PAGES,
                enablePlaceholders = false
            ),
            pagingSourceFactory = { MoviePagingSource(movieApi,
    MAX_PAGES, query) }
        ).flow
    }
}
```

Kode 3.6: MovieNetworkRepository

Pada MovieNetworkRepository terdapat fungsi getMovies() yang berfungsi untuk menampilkan seluruh poster film yang dibatasi dengan pageSize yaitu 20, poster film tersebut di dapatkan dari MoviePagingSource yang dipanggil dengan pagingSourceFactory. Selain itu terdapat fungsi searchMovies yang dapat menyimpan query sebagai String yang dapat memunculkan film dengan query yang sesuai dengan apa yang pengguna masukkan lalu diberikan pageSize sebesar 20 halaman sebagai batas munculnya film.

```
package com.example.moviedb.presentation.movieposter

@Suppress("DEPRECATION")
class MoviePosterAdapter(private val listener: OnItemClickListener
    )
    : PagingDataAdapter<MovieItem, MoviePosterAdapter.
    MoviePosterViewHolder>(DiffCallback()) {

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
    Int): MoviePosterViewHolder {
        val binding = ItemMoviesBinding.inflate(LayoutInflater.
    from(parent.context), parent, false)
        return MoviePosterViewHolder(binding)
    }

    override fun onBindViewHolder(holder: MoviePosterViewHolder,
    position: Int) {
        val currentItem = getItem(position)
        if (currentItem != null) {
            holder.bind(currentItem)
        }

```

```
        holder.itemView.setOnClickListener {
            if (currentItem != null) {
                listener.onItemClick(currentItem)
            }
        }
    }

    inner class MoviePosterViewHolder(private val binding:
ItemMoviesBinding) : RecyclerView.ViewHolder(binding.root) {
        init {
            binding.root.setOnClickListener {
                val position = adapterPosition
                if (position != RecyclerView.NO_POSITION) {
                    val item = getItem(position)
                    if (item != null) {
                        listener.onItemClick(item)
                    }
                }
            }
        }

        fun bind(item: MovieItem) {
            val imageUrl = Constants.IMG_URL + item.posterPath
            Glide.with(binding.root)
                .load(imageUrl)
                .into(binding.moviePosterImageView)
        }
    }

    class DiffCallback : DiffUtil.ItemCallback<MovieItem>() {
        override fun areItemsTheSame(oldItem: MovieItem, newItem:
MovieItem): Boolean {
            return oldItem.id == newItem.id
        }

        override fun areContentsTheSame(oldItem: MovieItem,
newItem: MovieItem): Boolean {
            return oldItem == newItem
        }
    }

    interface OnItemClickListener {
        fun onItemClick(item: MovieItem)
```

```
    }
}
```

Kode 3.7: MoviePosterAdapter

MoviePosterAdapter merupakan adapter RecyclerView yang memanfaatkan library Paging 3 untuk menampilkan poster film dalam daftar yang dipaginasi. Adapter ini mempunyai kelas internal MoviePosterViewHolder yang berfungsi untuk mengikat data film ke tampilan, termasuk menampilkan gambar poster menggunakan Glide. Untuk efisiensi, terdapat penggunaan DiffCallback yang membandingkan item film berdasarkan ID setiap film. Selain itu, terdapat interface OnItemClickListener untuk berinteraksi dengan setiap item film. Setelah itu terdapat penanganan klik pada item baik di metode onBindViewHolder maupun di inisialisasi view holder, sehingga interaksi pengguna bisa tertangkap dengan baik.

```
package com.example.moviedb.data.pagingsource

private const val STARTING_PAGE_INDEX = 1

class MoviePagingSource(
    private val service: MovieApi,
    private val maxPages: Int,
    private val query: String? = null
) : PagingSource<Int, MovieItem>() {
    override suspend fun load(params: LoadParams<Int>): LoadResult
    <Int, MovieItem> {
        val page = params.key ?: STARTING_PAGE_INDEX

        return try {
            val response =
            if (!query.isNullOrEmpty()) {
                service.searchMovies(query, page)
            } else {
                service.getPopularMovies(page)
            }

            LoadResult.Page(
                data = response.results.map { MovieItemResponse.
    toMovieItem(it) },
                prevKey = if (page == STARTING_PAGE_INDEX) null
    else page - 1,
                nextKey = if (page < maxPages) page + 1 else throw
```

```
    IOException("No more pages")
                )
            } catch (e: SocketTimeoutException) {
                LoadResult.Error(e)
            } catch (e: HttpException) {
                LoadResult.Error(e)
            } catch (e: IOException) {
                LoadResult.Error(e)
            } catch (e: Exception) {
                LoadResult.Error(e)
            }
        }

        override fun getRefreshKey(state: PagingState<Int, MovieItem>)
        : Int? {
            return state.anchorPosition
        }
}
```

Kode 3.8: MoviePagingSource

MoviePagingSource berfungsi untuk mengambil data film secara bertahap dari API, menggunakan PagingSource dari library Paging 3. Jika terdapat query pencarian, film akan dicari berdasarkan query tersebut sedangkan jika tidak ada, maka akan mengambil film populer. Jika permintaan ke API berhasil, hasilnya diubah menjadi MovieItem dan halaman berikutnya akan dimuat sampai batas maksimal halaman tercapai. Terdapat metode getRefreshKey yang digunakan untuk mengembalikan posisi yang membantu RecyclerView mengingat posisi terakhir yang dilihat pengguna.

```
package com.example.moviedb.data.datasource

interface MovieApi {
    @GET("movie/popular")
    suspend fun getPopularMovies(
        @Query("page") page: Int
    ): MovieResponse

    @GET("search/movie")
    suspend fun searchMovies(
        @Query("query") query: String?,
        @Query("page") page: Int
```

```
    ): MovieResponse
}
```

Kode 3.9: MovieApi

Interface MovieApi berfungsi untuk mengakses API film dengan dua metode utama. Metode pertama, getPopularMovies, digunakan untuk mengambil daftar film populer dan membutuhkan parameter halaman (page) sebagai query. Metode kedua, searchMovies, digunakan untuk mencari film berdasarkan query pencarian (query) yang diberikan dan juga memerlukan parameter halaman (page). Kedua metode ini ditandai dengan suspend, yang berarti mereka dijalankan secara asinkron dan mengembalikan MovieResponse, yaitu hasil dari permintaan API tersebut.

```
package com.example.moviedb.data.datasource.dto

data class MovieResponse(
    @field:SerializedName("page")
    val page: Int,

    @field:SerializedName("results")
    val results: List<MovieItemResponse>
)
```

Kode 3.10: MovieResponse

Data class MovieResponse digunakan untuk mengambil respons dari API film. Properti ”page” digunakan untuk menyimpan nomor halaman dari respons tersebut, yang berguna untuk navigasi atau paginasi data. Properti ”results” berisi daftar objek ”MovieItemResponse”, yang merupakan representasi data dari film-film yang ditemukan sesuai dengan permintaan API. Setiap ”MovieItemResponse” menyimpan informasi seperti judul film, gambar poster, deskripsi, dan atribut lainnya yang relevan dengan informasi film tersebut. Dengan menggunakan ”MovieResponse”, kita dapat dengan mudah mengakses dan mengelola data film yang diterima dari server melalui API.

```kotlin
package com.example.moviedb.data.datasource.dto

@Parcelize
data class MovieItemResponse(

    @field:SerializedName("overview")
    val overview: String,

    @field:SerializedName("original_language")
    val originalLanguage: String,

    @field:SerializedName("original_title")
    val originalTitle: String,

    @field:SerializedName("video")
    val video: Boolean,

    @field:SerializedName("title")
    val title: String,

    @field:SerializedName("genre_ids")
    val genreIds: List<Int>,

    @field:SerializedName("poster_path")
    val posterPath: String,

    @field:SerializedName("backdrop_path")
    val backdropPath: String,

    @field:SerializedName("release_date")
    val releaseDate: String,

    @field:SerializedName("popularity")
    val popularity: Double,

    @field:SerializedName("vote_average")
    val voteAverage: Double,

    @field:SerializedName("id")
    val id: Int,

    @field:SerializedName("vote_count")
    val voteCount: Int,
```

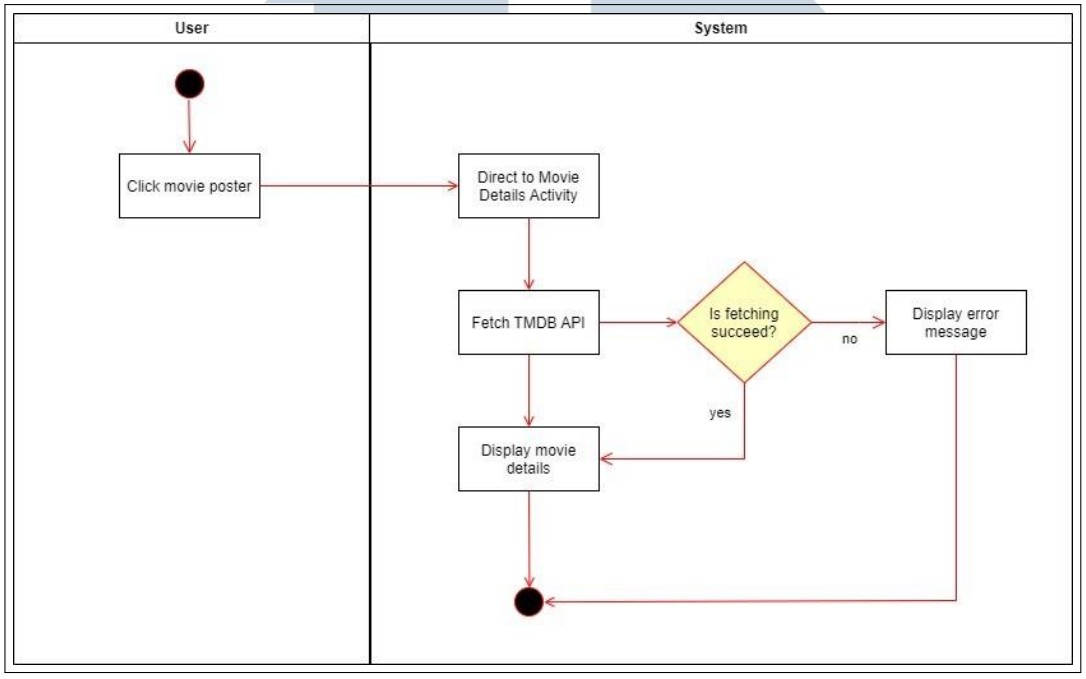
```
44
45      var isFavorite: Boolean
46  ) : Parcelable {
47      companion object {
48          fun toMovieItem(movieItemResponse: MovieItemResponse):
        MovieItem {
49              return MovieItem(
50                  overview = movieItemResponse.overview,
51                  title = movieItemResponse.title,
52                  posterPath = movieItemResponse.posterPath,
53                  releaseDate = movieItemResponse.releaseDate,
54                  voteAverage = movieItemResponse.voteAverage,
55                  id = movieItemResponse.id,
56                  isFavorite = movieItemResponse.isFavorite
57              )
58          }
59      }
60  }
```

Kode 3.11: MovieItemResponse

Data class MovieItemResponse digunakan untuk mendapatkan informasi dari film yang diterima dari API. Properti-properti seperti overview, originalLanguage, originalTitle, video, title, genreIds, posterPath, backdropPath, releaseDate, popularity, voteAverage, id, voteCount, dan isFavorite digunakan untuk menyimpan detail dari film tersebut. Metode toMovieItem di companion object digunakan untuk mengonversi objek MovieItemResponse menjadi objek MovieItem, dengan memilih properti yang relevan seperti judul, gambar poster, tanggal rilis, dan lainnya.

3.2.2 Movie Detail Page

Berikut merupakan *activity diagram* untuk fitur *search* yang dapat dilihat pada Gambar 3.4.



Gambar 3.4. *Activity Diagram* Fitur Detail Film

Pada fitur sebelumnya yaitu *list* film populer, jika pengguna klik salah satu dari poster film tersebut pengguna akan diarahkan ke halaman detail film, pada halaman detail film sistem akan menampilkan detail *movie* terkait pada TMDB API, antara lain: poster, *rating*, tanggal rilis, dan *overview*. Jika pengambilan detail *movie* tersebut gagal maka terdapat pesan informasi bahwa pengambilan detail film tersebut gagal, namun jika berhasil maka detail *movie* tersebut akan muncul. Pada halaman detail film juga terdapat tombol *favorite* sehingga pengguna dapat bebas memilih film favoritnya.

Berikut merupakan potongan kodenya:

```
package com.example.moviedb.presentation.detail

@AndroidEntryPoint
class MovieDetailsActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMovieDetailsBinding
    private lateinit var movieItem: MovieItem
    private val favoriteViewModel: FavoriteViewModel by viewModels
    ()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMovieDetailsBinding.inflate(
    layoutInflater)
        setContentView(binding.root)

        setSupportActionBar(binding.toolbar)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowTitleEnabled(false)
        supportActionBar?.setHomeAsUpIndicator(R.drawable.back)

        movieItem = if (Build.VERSION.SDK_INT >= Build.
    VERSION_CODES.TIRAMISU) {
            intent.getParcelableExtra(EXTRA_MOVIE, MovieItem::
    class.java) ?: return
        } else {
            @Suppress("DEPRECATION")
            intent.getParcelableExtra(EXTRA_MOVIE) ?: return
        }

        displayMovieDetails(movieItem)

        checkDatabase()

        favoriteStatusCollect()
        favoriteStatusRemove()

        binding.favoritesButton.setOnClickListener {
            try {
                if (movieItem.isFavorite) {
                    deleteFavorite(movieItem.toFavoriteMovie())
                } else {
                    addToFavorite(movieItem.toFavoriteMovie())
```

```
                }

                checkDatabase()
            } catch (e: Exception) {
                Toast.makeText(
                    this,
                    getString(R.string.error) + e.message,
                    Toast.LENGTH_SHORT
                ).show()
            }
        }
    }

    private fun favoriteStatusRemove() {
        lifecycleScope.launch {
            favoriteViewModel.statusCodeRemove.collectLatest {
                when (it) {
                    is DatabaseState.Loading -> {
                        CoroutineScope(Dispatchers.Main).launch {
                            Toast.makeText(
                                this@MovieDetailsActivity,
                                getString(R.string.loading),
                                Toast.LENGTH_LONG
                            ).show()
                        }
                    }

                    is DatabaseState.Success -> {
                        updateFavoriteButtonUI(isFavorite = false)
                        Toast.makeText(
                            this@MovieDetailsActivity,
                            getString(R.string.
removed_from_favorite),
                            Toast.LENGTH_LONG
                        ).show()
                    }

                    is DatabaseState.Error -> {
                        CoroutineScope(Dispatchers.Main).launch {
                            Toast.makeText(
                                this@MovieDetailsActivity,
                                getString(R.string.error) + it.
exception.message,
```

```
                                Toast.LENGTH_LONG
                            ).show()
                        }
                    }
                    else -> Unit
                }
            }
        }
    }

    private fun favoriteStatusCollect(){
        lifecycleScope.launch {
            favoriteViewModel.statusCode.collectLatest {
                when (it) {
                    is DatabaseState.Loading -> {
                        CoroutineScope(Dispatchers.Main).launch {
                            Toast.makeText(
                                this@MovieDetailsActivity,
                                getString(R.string.loading),
                                Toast.LENGTH_SHORT
                            ).show()
                        }
                    }

                    is DatabaseState.Success -> {
                        updateFavoriteButtonUI(isFavorite = true)
                        Toast.makeText(
                            this@MovieDetailsActivity,
                            getString(R.string.added_to_favorite),
                            Toast.LENGTH_LONG
                        ).show()
                    }

                    is DatabaseState.Error -> {
                        CoroutineScope(Dispatchers.Main).launch {
                            Toast.makeText(
                                this@MovieDetailsActivity,
                                getString(R.string.error) + it.
exception.message,
                                Toast.LENGTH_SHORT
                            ).show()
                        }
                    }
```

```
122
123                     else -> Unit
124                 }
125             }
126         }
127     }
128
129     private fun deleteFavorite(favoriteMovie: FavoriteMovie) {
130         favoriteViewModel.deleteFavorite(favoriteMovie)
131     }
132
133     private fun addToFavorite(favoriteMovie: FavoriteMovie) {
134         favoriteViewModel.addFavorite(favoriteMovie)
135     }
136
137     private fun displayMovieDetails(movieItem: MovieItem) {
138         binding.apply {
139             val imageUrl = Constants.IMG_URL + movieItem.
    posterPath
140             Glide.with(this@MovieDetailsActivity)
141                 .load(imageUrl)
142                 .into(moviePosterImageView)
143
144             movieTitleTextView.text = movieItem.title
145             movieRatingValueTextView.text = movieItem.voteAverage.
    toString()
146             movieOverviewTextView.text = movieItem.overview
147             movieReleaseDateTextView.text = movieItem.releaseDate
148         }
149     }
150
151     private fun checkDatabase() {
152         favoriteViewModel.isFavorite(movieItem.id) { isFavorite ->
153             movieItem.isFavorite = isFavorite
154             updateFavoriteButtonUI(movieItem.isFavorite)
155         }
156     }
157
158     private fun updateFavoriteButtonUI(isFavorite: Boolean) {
159         val favoriteDrawableRes = if (isFavorite) {
160             R.drawable.ic_favorite
161         } else {
162             R.drawable.ic_unfavorite
```

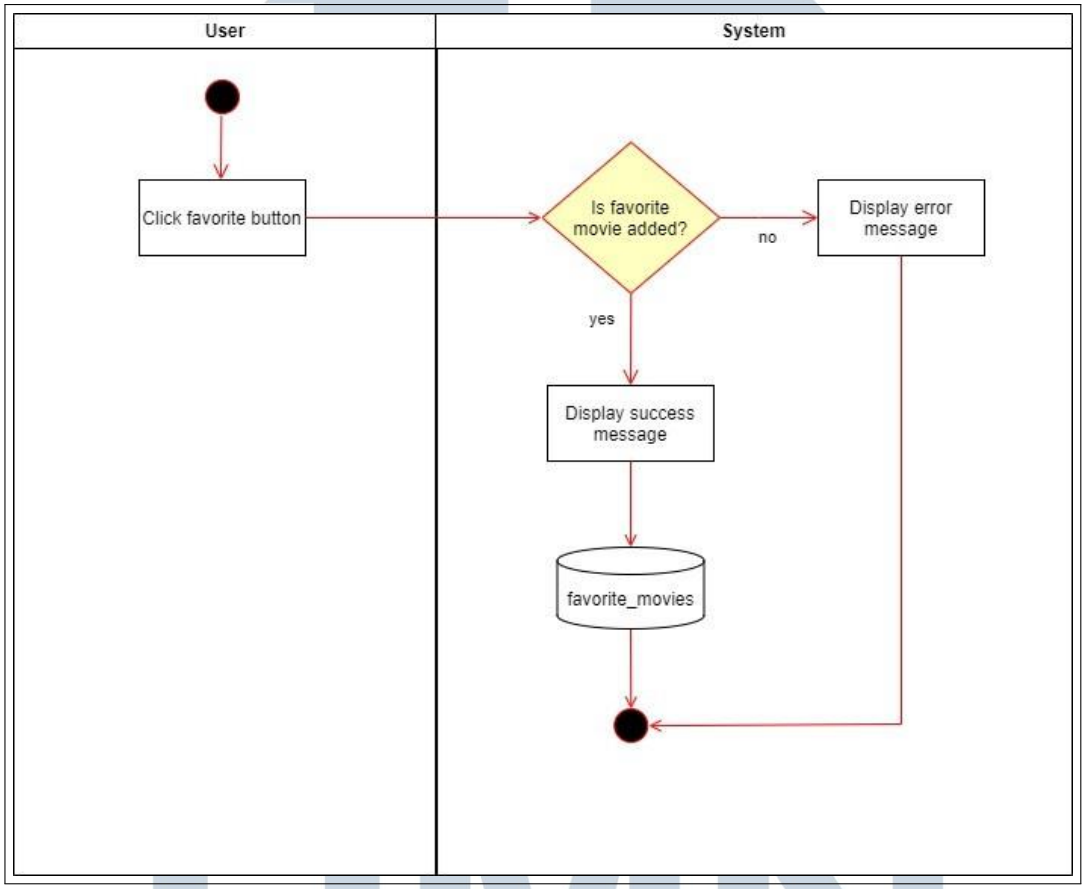
```
163         }
164         binding.favoritesButton.setImageResource(
    favoriteDrawableRes)
165         movieItem.isFavorite = isFavorite
166     }
167
168     override fun onSupportNavigateUp(): Boolean {
169         onBackPressedDispatcher.onBackPressed()
170         return true
171     }
172
173     companion object {
174         const val EXTRA_MOVIE = "extra_movie"
175     }
176 }
```

Kode 3.12: MovieDetailsActivity

Pada MovieDetailsActivity terdapat fungsi untuk menampilkan detail dari sebuah film, data film dari Intent diambil dan ditampilkan menggunakan binding. Selain itu terdapat fungsi untuk menambahkan dan menghapus film dari daftar favorit, dengan tombol favorit yang diperbarui sesuai dengan status film dalam database favorit. Metode checkDatabase digunakan untuk memeriksa apakah film sudah tersimpan dalam database favorit atau tidak, sedangkan metode updateFavoriteButtonUI digunakan untuk mengubah tampilan tombol favorit sesuai dengan status film yang ditampilkan. Selain itu, terdapat penanganan respons dari database favorit melalui favoriteViewModel dan menampilkan informasi sesuai dengan status operasi yang dilakukan, seperti penambahan atau penghapusan film dari daftar favorit.

3.2.3 Favorite Page

Berikut merupakan *activity diagram* untuk fitur *search* yang dapat dilihat pada Gambar 3.5.



Gambar 3.5. *Activity Diagram* Fitur *Favorite Movie*

Pada halaman *movie details*, pengguna dibebaskan untuk memilih film yang ingin difavoritkan, setelah pengguna menekan tombol favorit maka sistem akan melakukan pengecekan terhadap keberhasilannya, jika sistem mendeteksi bahwa gagal memasukan film ke dalam *list* film favorit maka muncul informasi bahwa terdapat kegagalan, jika sukses maka terdapat informasi bahwa film tersebut telah berhasil ditandai sebagai film *favorite* pengguna, film *favorite* yang telah dipilih tersebut akan dimasukan ke dalam *Room Database* dan akan di tampilkan ke dalam *Favorite Movie Fragment* dengan berbentuk *List View* menggunakan *Recycler View*. Jika pengguna ingin menghapus salah satu film tersebut dari *list* favoritnya, maka pengguna dapat menekan tombol *favorite* kembali dan sistem langsung menghapus film tersebut dari *list* favoritnya.

Berikut merupakan potongan kodenya:

```
package com.example.moviedb.presentation.favorite

@AndroidEntryPoint
class FavoriteFragment : Fragment() {
    private val favoriteViewModel: FavoriteViewModel by viewModels()
    private val favoriteMovieAdapter: FavoriteMovieAdapter by lazy { FavoriteMovieAdapter() }
    private lateinit var binding: FragmentFavoriteBinding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentFavoriteBinding.inflate(inflater, container, false)

        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        setupRecyclerView()

        favoriteViewModel.getAllFavoriteMovies()

        observeViewModel()
    }

    private fun observeViewModel() {
        favoriteViewModel.favoriteMovies.observe(viewLifecycleOwner) {
            favoriteMovieAdapter.setFavoriteMovies(it)
            binding.emptyMoviesTextView.visibility = if (it.isEmpty()) View.VISIBLE else View.GONE
        }
    }

    private fun setupRecyclerView() {
```

```
37          favoriteMovieAdapter.setOnItemClickListener { _: Int,
    favoriteMovie: FavoriteMovie ->
38              val movieItem = MovieItem(
39                  id = favoriteMovie.id,
40                  title = favoriteMovie.title.toString(),
41                  overview = favoriteMovie.overview.toString(),
42                  posterPath = favoriteMovie.posterPath.toString(),
43                  releaseDate = favoriteMovie.releaseDate.toString()
    ,
44                  voteAverage = favoriteMovie.voteAverage,
45                  isFavorite = favoriteMovie.isFavorite
46              )
47              val intent = Intent(requireContext(),
    MovieDetailsActivity::class.java).apply {
48                  putExtra(EXTRA_MOVIE, movieItem)
49              }
50              startActivity(intent)
51          }
52          binding.moviePosterRecyclerView.adapter =
    favoriteMovieAdapter
53      }
54  }
```

Kode 3.13: FavoriteFragment

FavoriteFragment berfungsi untuk menampilkan daftar film yang disukai pengguna. Terdapat setupRecyclerView yang digunakan untuk menampilkan daftar film favorit. Selain itu terdapat fungsi untuk membuka detail film ketika salah satu item di RecyclerView diklik, dengan membuat objek MovieItem dari data film favorit yang dipilih dan memulai Activity MovieDetailsActivity untuk menampilkan detail film tersebut. Detail film yang ditampilkan antara lain id film, judul film, overview dari film tersebut, posterPath yang digunakan untuk menampilkan poster film tersebut, tanggal rilis dari film tersebut, rata-rata vote dari penonton film tersebut, serta isFavorite yang digunakan untuk menjadi penanda keadaan film sudah difavoritkan pengguna atau belum. Namun tidak semua detail tersebut di tampilkan, karena yang hanya di tampilkan cukup poster film, judul film, rating atau rata-rata vote, overview, dan tanggal rilis film tersebut.

```
package com.example.moviedb.presentation.favorite

@HiltViewModel
class FavoriteViewModel @Inject constructor(private val
    favoriteRepository: FavoriteMovieRepository) :
    ViewModel() {
     private val _favoriteMovies: MutableLiveData<List<
    FavoriteMovie>> = MutableLiveData()
     val favoriteMovies: LiveData<List<FavoriteMovie>> get() =
    _favoriteMovies

     val statusCode = MutableStateFlow<DatabaseState<Long>?>(null)
    val statusCodeRemove = MutableStateFlow<DatabaseState<Int>?>(
    null)

     fun addFavorite(favoriteMovie: FavoriteMovie) {
         viewModelScope.launch {
             try {
                 favoriteRepository.addFavoriteMovie(favoriteMovie)
                 statusCode.value = DatabaseState.Success(1)
             } catch (e: Exception) {
                 statusCode.value = DatabaseState.Error(e)
             }
         }
     }

     fun deleteFavorite(favoriteMovie: FavoriteMovie) {
         viewModelScope.launch {
             try {
                 favoriteRepository.removeFavoriteMovie(
    favoriteMovie)
                 statusCodeRemove.value = DatabaseState.Success(1)
             } catch (e: Exception) {
                 statusCodeRemove.value = DatabaseState.Error(e)
             }
         }
     }

     fun getAllFavoriteMovies() {
         viewModelScope.launch {
             favoriteRepository.getFavoriteMovies().observeForever
    {
                 _favoriteMovies.value = it
```

```
            }
        }
    }

    fun isFavorite(id: Int, callback: (Boolean) -> Unit) {
        viewModelScope.launch(Dispatchers.IO) {
            val isFav = favoriteRepository.getFavoriteMovieById(id
    )
            withContext(Dispatchers.Main) {
                isFav?.let { callback(it) }
            }
        }
    }
}
```

Kode 3.14: FavoriteViewModel

Pada FavoriteViewModel, terdapat beberapa fungsi antara lain addFavorite yang digunakan untuk menambahkan film ke daftar favorit. Saat fungsi ini dipanggil, ViewModel akan meluncurkan sebuah coroutine untuk memanggil metode addFavoriteMovie dari favoriteRepository. Jika operasi berhasil, statusCode akan diubah menjadi DatabaseState.Success(1) yang menunjukkan penambahan sukses. Namun, jika terjadi kesalahan saat menambahkan film, statusCode akan menjadi DatabaseState.Error yang akan menampilkan pesan kesalahan terkait. Selain itu, fungsi deleteFavorite bertugas menghapus film dari daftar favorit. Sama seperti fungsi sebelumnya, coroutine digunakan untuk memanggil metode removeFavoriteMovie dari favoriteRepository. Jika operasi berhasil, statusCodeRemove akan menjadi DatabaseState.Success(1), sedangkan jika terjadi kesalahan, statusCodeRemove akan menampilkan DatabaseState.Error dengan pesan kesalahan yang sesuai. Fungsi getAllFavoriteMovies digunakan untuk mengambil semua film favorit dari repository. Ketika fungsi ini dipanggil, ViewModel akan meluncurkan coroutine untuk mengamati perubahan pada daftar film favorit yang diperoleh dari favoriteRepository. Setiap kali ada perubahan, LiveData pada favoriteMovies akan diperbarui dengan nilai yang baru. Terdapat fungsi isFavorite digunakan untuk memeriksa apakah suatu film sudah ada dalam daftar favorit berdasarkan ID-nya. Fungsi ini menerima sebuah callback yang akan dipanggil dengan parameter boolean yang menunjukkan status film tersebut dalam daftar favorit. Coroutine digunakan untuk menjalankan operasi ini secara asinkron di latar belakang menggunakan dispatcher IO, sehingga tidak memblokir antarmuka pengguna.

```
1  package com.example.moviedb.presentation.favorite.adapter
2
3  @Suppress("UNCHECKED_CAST")
4  class FavoriteMovieAdapter : ListAdapter<FavoriteMovie,
       FavoriteMovieAdapter.FavoriteMovieViewHolder>(
5      FavoriteMovieDiffCallback()
6  ) {
7      private var onItemClick: ((Int, FavoriteMovie) -> Unit)? =
       null
8
9      override fun onCreateViewHolder(parent: ViewGroup, viewType:
       Int): FavoriteMovieViewHolder {
10         val binding = ItemMoviesBinding.inflate(LayoutInflater.
       from(parent.context), parent, false)
11         return FavoriteMovieViewHolder(binding)
12     }
13
14     override fun onBindViewHolder(holder: FavoriteMovieViewHolder,
        position: Int) {
15         val favoriteMovie = getItem(position)
16         holder.bind(favoriteMovie)
17         holder.itemView.setOnClickListener {
18             onItemClick?.invoke(position, favoriteMovie)
19         }
20     }
21
22     fun setFavoriteMovies(listFavoriteMovie: List<FavoriteMovie>)
       {
23         submitList(listFavoriteMovie)
24     }
25
26     fun setOnItemClickListener(listener: Any) {
27         this.onItemClick = listener as ((Int, FavoriteMovie) ->
       Unit)
28     }
29
30     inner class FavoriteMovieViewHolder(private val binding:
       ItemMoviesBinding) :
31         RecyclerView.ViewHolder(binding.root) {
32         fun bind(favoriteMovie: FavoriteMovie) {
33             with(binding) {
34                 Glide.with(root.context)
35                     .load(Constants.IMG_URL + favoriteMovie.
```

```
    posterPath)
                    .into(moviePosterImageView)
            }
        }
    }

    class FavoriteMovieDiffCallback : DiffUtil.ItemCallback<
    FavoriteMovie>() {
        override fun areItemsTheSame(oldItem: FavoriteMovie,
    newItem: FavoriteMovie): Boolean {
            return oldItem.id == newItem.id
        }
        override fun areContentsTheSame(oldItem: FavoriteMovie,
    newItem: FavoriteMovie): Boolean {
            return oldItem == newItem
        }
    }
}
```

Kode 3.15: FavoriteMovieAdapter

Adapter FavoriteMovieAdapter berfungsi untuk mengatur tampilan daftar film favorit dalam RecyclerView. Fungsi onCreateViewHolder digunakan untuk membuat ViewHolder baru saat RecyclerView memerlukan tampilan item, dengan menginflate tata letak item menggunakan ItemMoviesBinding. Data dari objek FavoriteMovie disimpan ke dalam ViewHolder, termasuk memuat gambar poster film favorit dengan Glide dan menetapkan OnClickListener untuk menangani klik pada item. Fungsi setFavoriteMovies mengatur daftar film favorit yang akan ditampilkan, sedangkan setOnItemClickListener menetapkan listener untuk menangani klik item RecyclerView. FavoriteMovieViewHolder mewakili tampilan ViewHolder untuk item dalam RecyclerView, dengan fungsi bind yang menyimpan data dari objek FavoriteMovie ke tampilan ViewHolder. Selain itu, terdapat FavoriteMovieDiffCallback yang mengimplementasikan DiffUtil.ItemCallback untuk membandingkan item-item dalam daftar film favorit, memeriksa keberadaan item dengan ID yang sama.

3.3 Uraian Pelaksanaan Magang

Berikut adalah penjelasan mengenai uraian pekerjaan yang telah dilakukan dan sebutkan pada Tabel 3.1 yang tercantum sebelumnya:

3.3.1 Minggu 1

Pelaksanaan *training softskill* dengan materi *Presentation Skill*, *training softskill* untuk minggu pertama ini banyak mengajarkan untuk dapat memberikan presentasi di depan para audiens dengan lebih baik dan lebih formal namun tetap tidak kaku, ketika *training softskill*, selain itu mempelajari mengenai gestur dan pelafalan yang baik ketika presentasi. Selanjutnya mempelajari hal-hal yang termasuk dalam *Kotlin Basic* meliputi *Basic Syntax*, *Data Types*, *Operators*, *Functions*, *Control Flow*, *If Else Expressions*, *When Expression*, *For Loop*, *While Loop*, *Break*, dan *Continue*. Mempelajari *Kotlin Function* yang meliputi *Function Parameter*, *Function Return Type*, *Function Named and Default Argument*, *Single-Expression Function*, *Function Varargs*, *Tail Recursive Function*, *Lambda Expression*, *High Order Function*, *Anonymous Function*, *Inline Function*, *Extension Function*, *Infix Notation*. Serta mengerjakan tugas untuk mengimplementasikan materi tersebut dengan membuat *function* untuk menampilkan informasi produk berupa nama produk, stok produk, dan harga satuan produk. Pada *function* tersebut pengguna dapat meng*input* berupa *quantity* pembelian dan nominal pembayaran, *output* yang diharapkan bahwa partisipan dapat menampilkan informasi produk yang di-*input*, menampilkan total harga produk, menampilkan diskon produk dengan ketentuan yang telah diberikan, menampilkan total bayar jika terdapat diskon, pengendalian jika stok produk yang kurang dibandingkan permintaan pembeli, pengendalian jika nominal pembayaran pembeli kurang dari total harga, pengendalian jika *input* pengguna kosong atau 0.

3.3.2 Minggu 2

Pelaksanaan *training softskill* dengan materi *Effective Communication*, dengan materi tersebut dapat mengetahui dan memahami cara untuk dapat berkomunikasi dengan baik, efektif, serta efisien dengan memperhatikan intonasi dan pelafalan yang baik, selain itu juga mempelajari hal-hal yang termasuk dalam *Kotlin Data Classes and Collection* yang meliputi *Data Classes*, *Collection*, *list*, *Sets*, *Pair*, *Triple*, *Maps*, *Collection Operations*. Mempelajari *Kotlin OOP*

yang meliputi *Abstract Classes*, *Class and Objects*, *Constructor*, *Delegation*, *Destructuring Declarations*, *Exception Handling*, *Inheritance*, *Interface*, *Sealed Class*, *Visibility Control*. Mempelajari *Android Development* yang meliputi *Hello World Project*, *Activity*, *Resources*, *View and View Group*, *ScrollView*. Serta mengerjakan tugas untuk mengimplementasikan materi tersebut yaitu degan membuat aplikasi untuk menghitung volume bangun ruang yang telah di tentukan, membuat aplikasi kalkulator dengan fungsi tertentu, dan aplikasi *login* dengan ketentuan yang telah diberikan. Terdapat persyaratan yang harus diperhatikan yaitu *function* harus memiliki parameter *username* dan *password*, *username* terdiri dari 6 karakter hingga 15 karakter dengan kombinasi huruf dan angka, *password* terdiri dari minimal 8 karakter hingga 20 karakter dengan kombinasi simbol, huruf, dan angka, jika *login* gagal maka terdapat pesan bahwa *login* gagal beserta penyebabnya, dan jika *login* berhasil tampilkan pesan bahwa *login* berhasil.

3.3.3 Minggu 3

Pada minggu ketiga, terdapat *training softskill* yang berfokus pada *Work Ethics* yang ada di PT Global Loyalty Indonesia, pada *training softskill* pada minggu ini diharapkan dapat memiliki visi dan misi yang sama dengan PT Global Loyalty Indonesia. Mempelajari *Style and Theme*, *Toolbar and Menu*, *Intent*, *Kotlin Parcelable*, *Localization*, *Debugging*, *Nested ScrollView*, *RecyclerView*, *Fragment*, *TabLayout*, *BottomNavigationView*, *Object and Companion Object*, dan *Solid Principles*. Implementasi materi-materi tersebut dalam penugasan yang berupa pembuatan aplikasi yang serupa dengan aplikasi Alfagift dengan menampilkan halaman produk berdasarkan kategori, penanganan jika produk di klik maka akan masuk ke halaman detail produk, menampilkan banner, produk dengan penawaran terbaik, dan *official store*. Pada bagian menampilkan banner harus menampilkan *list* data banner dan jika banner di klik maka akan menampilkan *list* produk dengan banner yang sesuai di atasnya. Pada bagian produk dengan penawaran terbaik maka dapat menampilkan *list* produk dan jika produk tersebut di klik maka akan masuk ke halaman detail produk. Pada bagian *official store* maka dapat menampilkan *list official store* dan jika logo dari *official store* tersebut di klik maka akan menampilkan detail *official store* dengan menampilkan *list* produk.

3.3.4 Minggu 4

Pada minggu keempat ini, pelaksanaan *training softskill* yang berfokus pada *Team Work,* di materi *softskill* minggu ini mempelajari bagaimana cara untuk dapat bekerja dalam tim dengan baik, dan bagaimana cara agar seuatu tim dapat mencapai tujuan bersama dengan baik. Mempelajari penggunaan *SnackBar*, *BottomSheet*, *AlertDialog, DialogFragment, ViewBinding, BackgroundThread*, *Networking*, *Modern Android Developement*, dan *Android Architecture Component* yang dilanjutkan dengan pengerjaan tugas mingguan yang berupa membuat aplikasi menggunakan API TMDB dengan membuat tampilan halaman menggunakan *Fragment* dan *View Pager* untuk pengendalian halaman sehingga dapat lebih mudah digunakan oleh pengguna, juga menggunakan *View Binding* untuk *layout-layout* pada aplikasi tersebut.

3.3.5 Minggu 5

Penggunaan *training softskill* yang berfokus pada *Result Oriented*, mendapatkan pemahaman mengenai bagaimana caranya untuk mendapatkan hasil dari kinerja tim yang baik dan cara mempertahankan kinerja baik tersebut. Pada *training softskill* minggu ini juga terdapat pengetahuan bahwa pentingnya memiliki semangat yang tinggi untuk menggapai tujuan tersebut. Dilanjutkan untuk mempelajari *Background Task*, *Notification*, *Permission*, dan *Local Data*. Pengerjaan tugas untuk membuat aplikasi ”The Movie App” dengan mengimplementasikan materi pada minggu ini, terdapat penambahan fitur antara lain penambahan penanganan ketika pengguna tidak memiliki akses internet, menyimpanan *list* film yang disukai pengguna ke dalam *Local Database Room* yang ditampilkan pada halaman *”Favorite Movies”* serta menambahkan tombol favoritnya.

3.3.6 Minggu 6

Pelaksanaan *training softskill* yang berfokus pada *Analytical Thinking*, mendapatkan pemahaman yang baik bagaimana caranya untuk memiliki pemikiran yang lebih analitik dan krisis untuk menghadapi segala tantangan dalam bekerja, selain itu juga mempelajari bahwa untuk menyelesaikan masalah-masalah mendatang dapat menggunakan beberapa metode diantaranya metode *Fish Bone*,

5 Whys, dll. Mempelajari *Clean Architecture, MVVM, Coroutines, Dependecy Injection, Jetpack Paging 3, dan Jetpack Compose.*

3.3.7 Minggu 7

Implementasi dari *Clean Architecture, MVVM, Coroutines, Dependecy Injection, Jetpack Paging 3, dan Jetpack Compose* pada project lainnya agar dapat lebih mengerti dan memahaminya.

3.3.8 Minggu 8

Pada minggu kedelapan ini pelaksanaan *training softskill* untuk terakhir kalinya dengan materi pembahasan terhadap *Awareness*, pada materi kali ini mendapatkan ilmu bagaimana caranya untuk menjadi pribadi yang lebih baik terhadap lingkungan sekitarnya, dari materi ini diharapkan dapat lebih peka terhadap sekitar sehingga jika terdapat orang-orang yang terindikasi akan menggangu maka dapat lebih mudah menyikapinya dan menghindari untuk memiliki sifat-sifat mengganggu tersebut pada diri sendiri. Melanjutkan pengerjaan aplikasi ”The Movie App” dengan menambahkan penanganan ketika pengguna melakukan *scroll* terhadap *list* film yang tersedia yang dimana jika sudah sampai pada halaman terakhir terdapat informasi bahwa tidak ada film yang dapat di tampilkan dengan menggunakan *End Of Pagination* dari *Jetpack Paging 3*, selain itu juga sudah mengimplementasikan penggunaan *MVVM* sehingga menanganan terhadap setiap folder, fitur, dan kode dapat lebih mudah, menggunakan *Dependency Injection* terhadap tiap *View Model* dan *Activity* atau *Fragment* yang digunakan.

3.3.9 Minggu 9

Membuat aplikasi ”The Movie App” untuk menampilkan *list* dari film yang sedang populer dengan implementasi dari *Paging 3* dan penggunaan *Dependency Injection*. Serta menyelesaikan setiap komentar dari para *Mentor* di Bit Bucket sehingga pengembangan aplikasi tersebut lebih baik dan optimal hingga disetujui, setelah itu dilakukan *Weekly Review* yang bertujuan untuk meninjau ulang apa yang sudah di pelajari dan sejauh mana memahami materi tersebut. Selain menguji kemampuan, saat *Weekly Review* para *Mentor* juga menanyakan kesulitan apa saja

yang dihadapi sehingga kedepannya antara *Mentor* dan *Mentee* dapat lebih baik lagi.

3.3.10 Minggu 10

Melanjutkan pembuatan aplikasi ”The Movie App” dengan menambahkan beberapa fitur berupa *save* di *Local Database* menggunakan *Room* dan *Realm* untuk fitur *Favorite* dan *Search* pada aplikasi *Movie* tersebut, setelah itu memperbaiki setiap komentar dari *Mentor* di Bit Bucket agar aplikasi dapat berjalan lebih baik dan lebih optimal, setelah itu di lakukan *Weekly Review*.

3.3.11 Minggu 11

Membuat aplikasi ”My Nearest Gas Station” yaitu aplikasi yang dapat digunakan untuk mencari dan menampilkan pom bensin terdekat berdasarkan keberadaan pengguna dan lokasi yang diinginkan pengguna dengan mengimplementasikan Google Maps API, pengendalian perizinan terutama *negative case* atau keadaan ketika pengguna tidak mengizinkan penggunaan lokasi dan GPS untuk aplikasi tersebut.

3.3.12 Minggu 12

Melanjutkan pembuatan aplikasi ”My Nearest Gas Station” dengan menambahkan pengendalian pergerakan pengguna di *Map View* tersebut agar ketika pengguna menggerakannya maka pom bensin terdekat yang muncul dalam *list* mengikuti *pin point* lokasi yang terbaru, mengendalikan pergerakan ketika pengguna berjalan maka *pin point* juga akan ikut berjalan. Setelah itu memperbaiki dan meninjau ulang komentar yang diberikan oleh *Mentor* pada Bit Bucket agar aplikasi dapat berjalan lebih baik dan optimal lagi terutama pada bagian bagian yang sensitif (penanganan perizinan lokasi), melakukan *Weekly Review* bersama para *Mentor*.

3.3.13 Minggu 13

Membuat aplikasi kamera yang dapat digunakan untuk mengambil foto dan menyimpannya di galeri aplikasi dan galeri *device* dengan mengimplementasikan *Local Database Room*, serta terdapat fitur scan dengan mengimplementasikan ML

Kit dan fitur *QR Code Generator* berdasarkan input pengguna berupa *string* dengan mengimplementasikan ZXing. Setelah itu memperbaiki aplikasi berdasarkan komentar yang diberikan oleh *Mentor* agar lebih optimal dan melakukan *Weekly Review*.

3.3.14 Minggu 14

Melanjutkan pengembangan aplikasi ”The Movie App” sebelumnya dari MVVM atau *Model-View-View Model* menjadi MVVM dan *Clean Architecture*. Dengan adanya refaktorisasi ini diharapkan partisipan dapat lebih memahami dan terbiasa dengan penggunaan *Clean Architecture* kedepannya. Setelah itu memperbaiki aplikasi berdasarkan komentar yang diberikan oleh *Mentor* agar lebih optimal dan melakukan *Weekly Review*.

3.4 Kendala dan Solusi yang Ditemukan

Kendala yang dihadapi selama praktik kerja magang di PT Global Loyalty Indonesia adalah sebagai berikut.

1. Kesulitan dalam beradaptasi dengan lingkungan kerja yang sangat jauh berbeda dengan lingkungan ketika kuliah sebelumnya, di lingkungan kerja harus dapat bersikap profesional dan tangguh terhadap apapun baik sikap maupun perkataan dari orang lain. Kesadaran bahwa pentingnya membuka diri terhadap relasi baru salah satunya tidak malu untuk bertanya dan membicarakan hal yang belum diketahui.
2. Kesulitan dalam memahami beberapa materi, hal ini terjadi karena sebelumnya tidak mendapatkan materi pembelajaran tersebut di masa perkuliahaan.
3. Kesulitan untuk mengakses ilmu, saat awal memulai praktik kerja magang PT Global Loyalty memberikan Wifi khusus yang ternyata tidak dapat digunakan untuk membuka situs Youtube, sehingga harus menggunakan internet pribadi untuk mengakses dan belajar berdasarkan video pembelajaran yang terdapat dalam Youtube sehingga lebih mudah untuk memahaminya.

Solusi yang ditemukan dan telah di lakukan antara lain:

1. Berusaha untuk membuka diri terhadap semua orang yang ada di PT Global Loyalty Indonesia dan mengurangi sifat malu yang terdapat dalam diri sendiri memberanikan diri untuk bertanya, menggali ilmu sedalam-dalamnya dan mengupayakan pengerjaan tugas semaksimal mungkin, meminimalisir komentar dari para *Mentor* dengan cara memperhatikan setiap detail dari tugas yang diberikan, mengeksplor ilmu sehingga tidak merasa tertinggal dari rekan-rekan lainnya.
2. Mempelajari ilmu-ilmu baru dengan lebih giat dengan cara eksplorasi secara mandiri, berdiskusi kepada siapapun yang dirasa mampu untuk diajak berdiskusi, memperbanyak dan membiasakan diri untuk lebih banyak membaca dan memahami, mencari tahu *golden time* atau waktu terbaik untuk mempelajari ilmu baru dan mengimplementasikannya, mencari cara dan gaya belajar yang sesuai, efektif, dan efisien.
3. Menyediakan internet lebih banyak dari sebelumnya sehingga tidak keterbatasan untuk menonton video pembelajaran dari situs Youtube maupun situs pendukung pembelajaran lainnya, memberi tahu kepada *Mentor* mengenai keterbatasan penggunaan Wifi tersebut.

3.5 Perangkat Penunjang

Selama melaksanakan kerja magang di PT Global Loyalty Indonesia, terdapat beberapa perangkat lunak dan perangkat keras yang digunakan untuk mengerjakan tugas setiap harinya. Berikut merupakan daftar perangkat lunak yang digunakan, antara lain:

1. Android Studio Hedgehog 2023.1.1 Patch 2
2. BitBucket
3. Search Engine:
    - Google Chrome
    - Microsoft Edge
4. Postman

5. Telegram

Terdapat spesifikasi khusus untuk perangkat keras yang digunakan, yaitu:

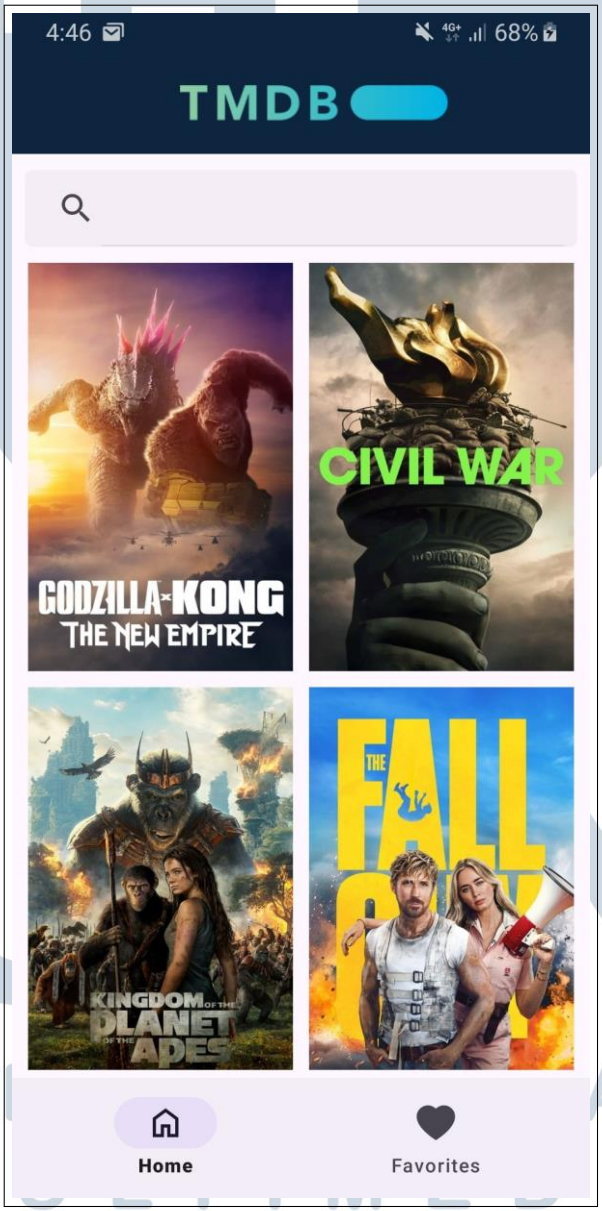
1. Operating System: Windows
2. RAM: 8 GB
3. Storage: 512 GB SSD
4. Processor: 10th Generation Intel Core i7

3.6 Implementasi

Aplikasi ini dibangun menggunakan bahasa pemrograman Kotlin dan XML, dan menerapkan *Clean Architecture* untuk memudahkan pengelolaan kode. *Clean Architecture* membagi kode menjadi beberapa lapisan seperti domain, data, dan presentasi, yang berfungsi untuk menjaga struktur kode agar tetap rapi, mudah untuk dipelihara, dan dikembangkan. Pendekatan *MVVM (Model-View-ViewModel)* diterapkan untuk memisahkan logika bisnis, data, dan tampilan (UI). Model menangani logika bisnis dan struktur data, *View* bertanggung jawab atas interaksi pengguna dan penyajian data, sedangkan *ViewModel* mengelola data untuk *View* serta menangani logika UI yang kompleks. Dengan pemisahan ini, aplikasi menjadi lebih modular dan fleksibel, sehingga perubahan pada satu komponen tidak memengaruhi komponen lainnya, yang memudahkan pemeliharaan dan pengembangan kedepannya. Terdapat penggunaan *Room Database* yang digunakan untuk menyimpan list film favorit pengguna, *Room Database* dipilih untuk menyimpan *list* film favorit pengguna karena kemampuannya dalam menyediakan API yang kuat untuk SQLite, termasuk dukungan untuk operasi basis data relasional dan migrasi skema yang lebih mudah. Sedangkan *Realm Database* digunakan untuk menyimpan *query* pencarian film pengguna karena performanya yang tinggi dalam menangani data yang sering berubah dan kemampuan untuk melakukan operasi baca/tulis dengan cepat.

3.6.1 Home Page

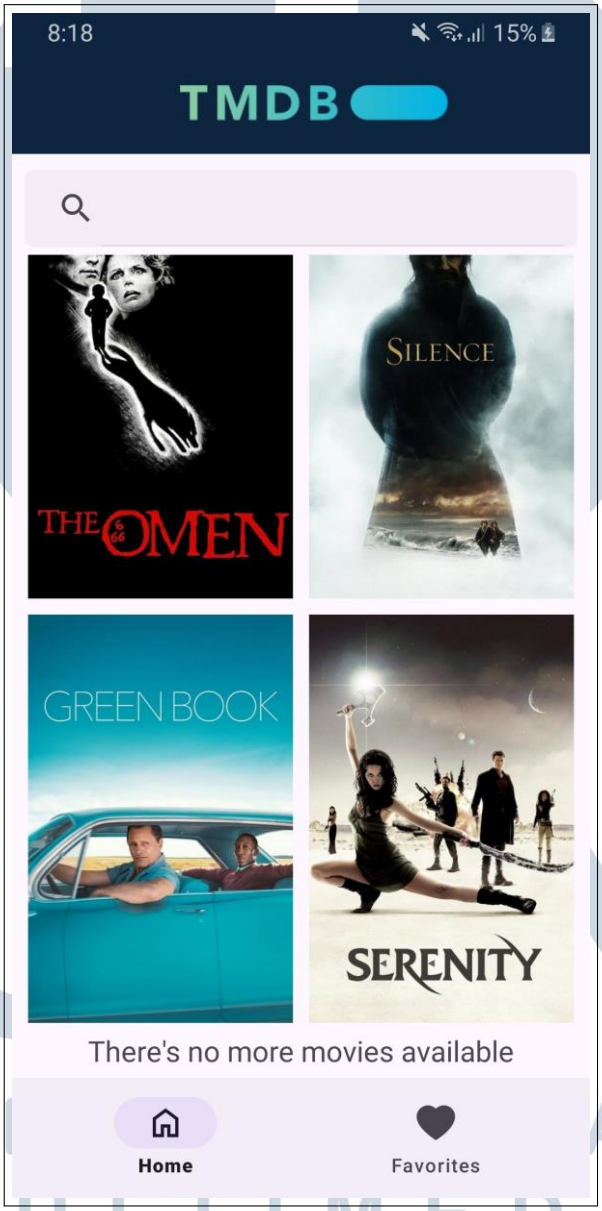
Bagian Home Page merupakan halaman pertama yang dapat dilihat oleh pengguna ketika pengguna membuka aplikasi The Movie App. Berikut merupakan tampilan dari Home Page yang dapat dilihat melalui Gambar 3.6.



Gambar 3.6. *Home Page*

Pada Gambar 3.6. *list* dari film populer didapatkan dari pemanggilan TMDB API dengan *query* ”https://api.themoviedb.org/3/movie/popular”, terdapat fitur *End of Pagination* yang berfungsi sebagai pemberi tanda bahwa *list* film sudah

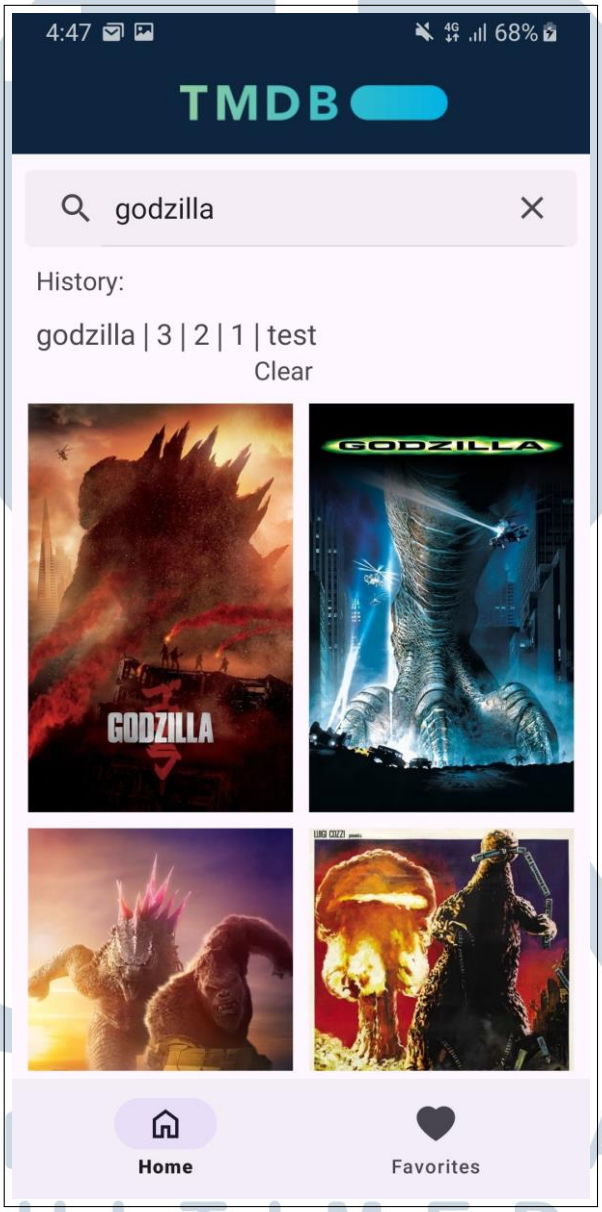
ditampilkan semuanya. Pada Gambar 3.7. merupakan tampilan ketika pengguna telah mencapai halaman terakhirnya, terdapat kalimat *"There's no more movies available"* untuk memberikan informasi kepada pengguna bahwa *list* film populer sudah dimuat seluruhnya.



Gambar 3.7. *End of Pagination*

A. Search Film

Fitur Search Film dapat digunakan pengguna untuk mencari film yang diinginkan. Berikut merupakan tampilan jika pengguna menggunakan fitur search



Gambar 3.8. Fitur *Search*

Pada fitur search film, pengguna dapat mencari seluruh judul film yang diinginkan pada Search View dan semua judul film yang telah pengguna cari sebelumnya akan tersimpan pada Realm Database, namun pengguna juga dapat menghapus seluruh riwayat pencariannya dengan menekan teks ”Clear”. Setiap

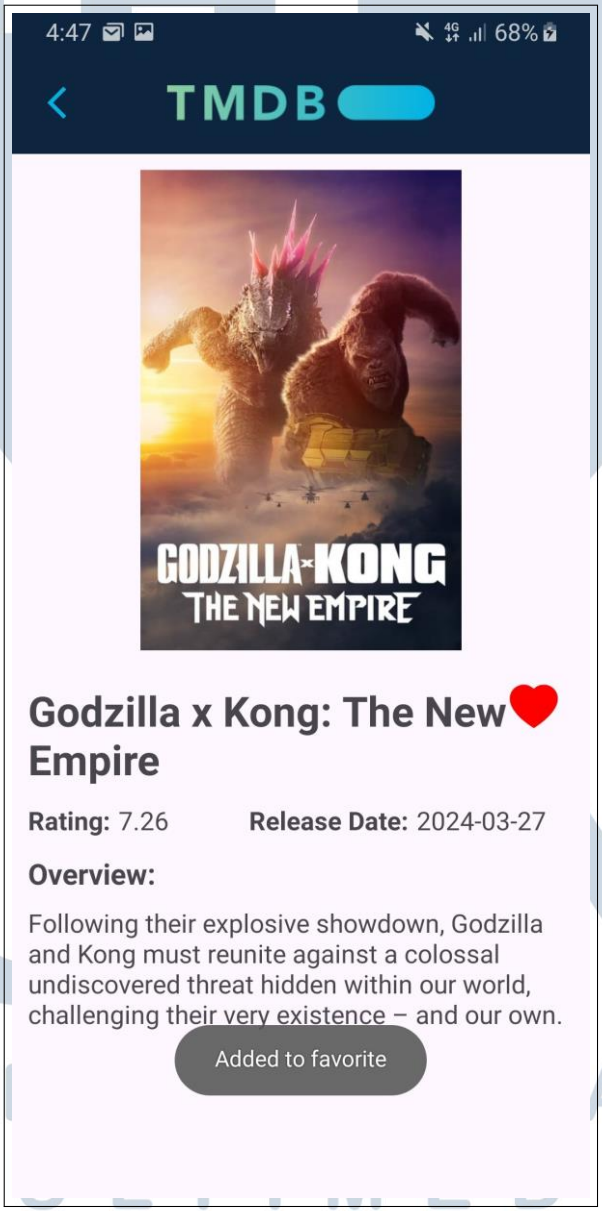
riwayat pencarian pengguna juga sudah diurutkan dari riwayat pencarian terlama hingga terbaru. Jika ketika pengguna mencari film yang diinginkan namun tidak ada dalam TMDB API maka muncul End of Pagination sebagai informasi bahwa film yang dicari oleh pengguna tidak ditemukan.

Pada Gambar 3.9. merupakan tampilan dengan kondisi ketika film yang dicari oleh pengguna tidak ditemukan

Gambar 3.9. *Search Empty*

3.6.2 Movie Detail Page

Ketika pengguna klik poster pada list film populer maka pengguna akan masuk ke halaman movie detail, berikut merupakan tampilan dari movie detail page yang dapat dilihat melalui Gambar 3.10.

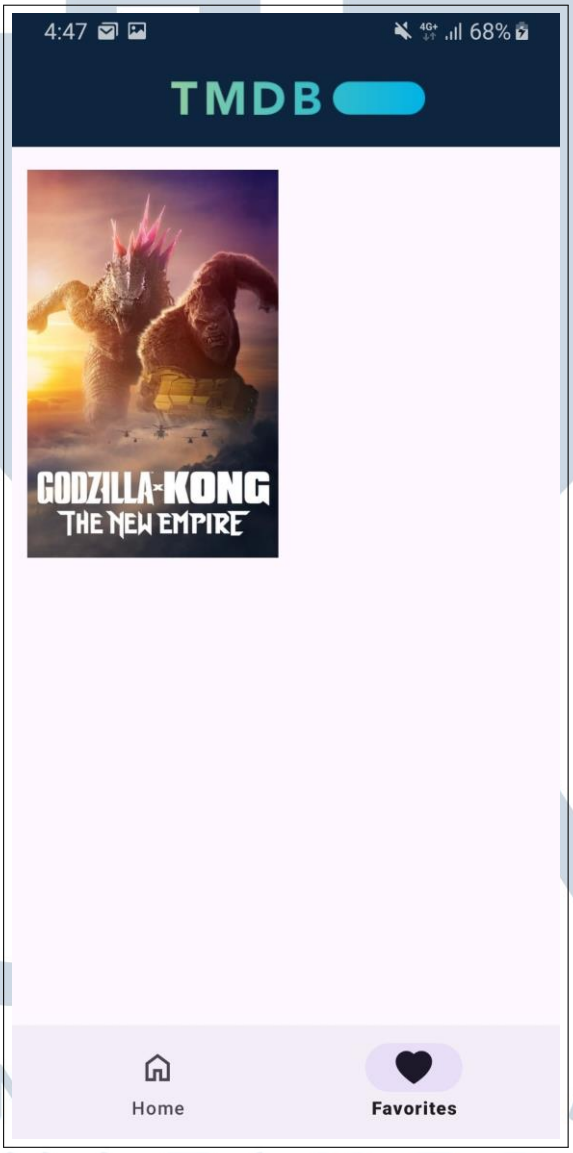


Gambar 3.10. *Movie Detail Page*

Pada halaman movie detail terdapat informasi mengenai filmnya, yaitu judul film, rating, tanggal rilis, dan overviewnya. Terdapat tombol favorite yang dapat digunakan untuk menandai film tersebut sebagai film favorit pengguna

3.6.3 Favorite Page

Jika pengguna telah klik *button favorite* maka film yang difavoritkan *user* akan tersimpan melalui *Room Database* dan muncul di halaman *favorite*. Berikut merupakan tampilan ketika user telah klik *button favorite*



Gambar 3.11. *Favorite Movie Page*

Ketika pengguna tidak ingin film tersebut berada di *list favorite*, pengguna diberikan kebebasan untuk tidak memfavoritkan kembali film tersebut dengan cara menekan poster film pada *list* favorite yang akan mengarahkan ke halaman *Movie Details Activity*.