Dokumentasi Uji Kom BNSP

Disusun oleh:

Nama : **DAI EL HIKAM**
Kelas : **Pemrograman Perangkat Lunak Mobile**
Asal : **Bogor**

PESANTREN TEKNOLOGI INFORMASI DAN KOMUNIKASI
Creats Future Skilled Professionals
2025

MainActivity.kt

```kotlin
package com.example.mylistsiswa.ui.main

import ...

class MainActivity : AppCompatActivity() {

    private var _activityMainBinding: ActivityMainBinding? = null
    private val binding get() = _activityMainBinding!!

    private lateinit var adapter: SiswaAdapter
    private lateinit var fullSiswaList: List<Siswa>

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

        adapter = SiswaAdapter()
        binding.rvSiswa.layoutManager = LinearLayoutManager( context: this)
        binding.rvSiswa.setHasFixedSize(true)
        binding.rvSiswa.adapter = adapter

        binding.fabAdd.setOnClickListener {
            val intent = Intent( packageContext: this@MainActivity, SiswaAddUpdateActivity::class.java)
            startActivity(intent)
        }

        val mainViewModel = obtainViewModel( activity: this@MainActivity)
        mainViewModel.getAllSiswa().observe( owner: this) { siswaList ->
            if (siswaList != null) {
                fullSiswaList = siswaList
                adapter.setListSiswa(siswaList)
            }
        }

        binding.searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener {
            override fun onQueryTextSubmit(query: String?): Boolean {
                return false
            }

            override fun onQueryTextChange(newText: String?): Boolean {
                val filteredList = fullSiswaList.filter {
                    it.namaLengkap?.contains( other: newText ?: "", ignoreCase = true) == true ||
                            it.nis?.contains( other: newText ?: "", ignoreCase = true) == true
                }
                adapter.setListSiswa(filteredList)
                return true
            }
        })
    }
```

```
    }

    Edit | Explain | Test | Document | Fix
    private fun obtainViewModel(activity: AppCompatActivity): MainViewModel {
        val factory = ViewModelFactory.getInstance(activity.application)
        return ViewModelProvider(activity, factory)[MainViewModel::class.java]
    }

    Edit | Explain | Test | Document | Fix
    override fun onDestroy() {
        super.onDestroy()
        _activityMainBinding = null
    }
}
```

1. `MainActivity.kt`

Fungsi:

Sebagai tampilan utama aplikasi yang menampilkan seluruh daftar (Siswa) dalam bentuk list.

Penjelasan:

- Menggunakan `RecyclerView` dengan `SiswaAdapter`.
- Mengamati data `LiveData` dari `MainViewModel`.
- Menerapkan fitur pencarian (SearchView) untuk memfilter siswa berdasarkan nama/nim.
- Navigasi ke `SiswaAddUpdateActivity` untuk menambah atau mengedit siswa.

MainViewModel.kt

```kotlin
package com.example.mylistsiswa.ui.main

import android.app.Application
import androidx.lifecycle.LiveData
import androidx.lifecycle.ViewModel
import com.example.mylist.database.Siswa
import com.example.mylistsiswa.repository.SiswaRepository

class MainViewModel(application: Application): ViewModel() {
    private val mSiswaRepository: SiswaRepository = SiswaRepository(application)
    fun getAllSiswa(): LiveData<List<Siswa>> = mSiswaRepository.getAllSiswa()
}
```

2. `MainViewModel.kt`

Fungsi:

ViewModel untuk `MainActivity`.

Penjelasan:

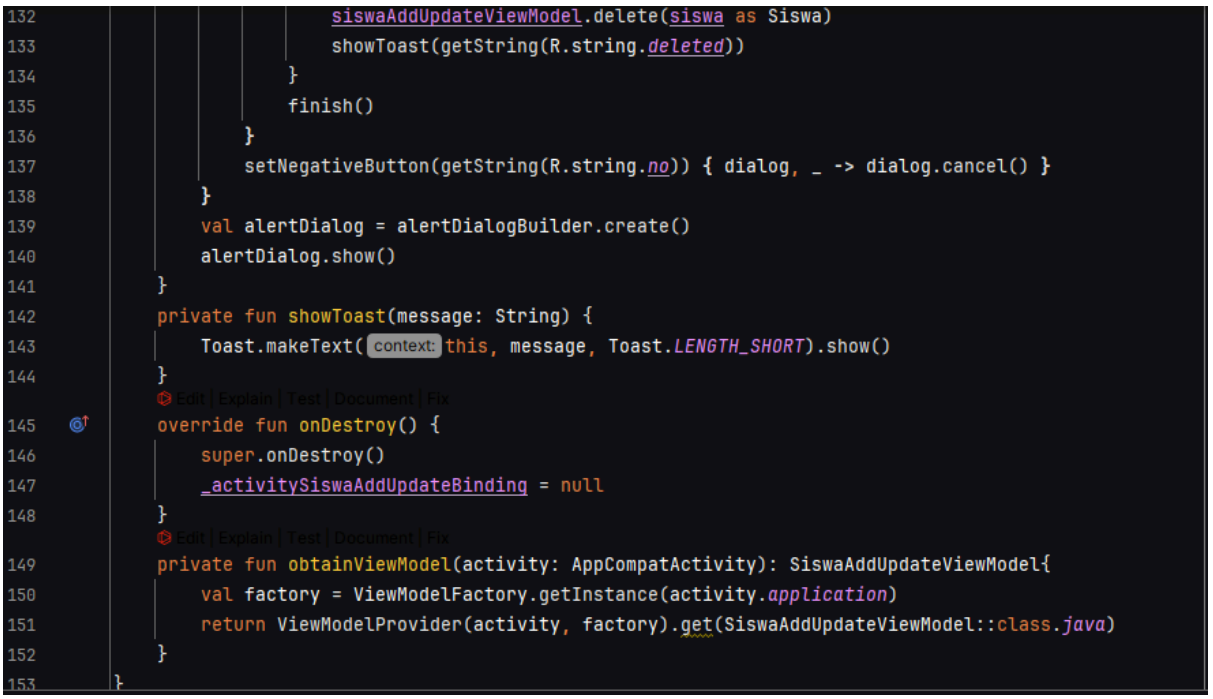
- Mengambil semua data List dari `SiswaRepository`.
- Menyediakan `LiveData<List<Siswa>>` agar UI bisa mengamati perubahan data secara otomatis.

SiswaAddUpdateActivity.kt

```kotlin
package com.example.mylistsiswa.ui.insert

import ...

class SiswaAddUpdateActivity : AppCompatActivity() {
    companion object{
        const val EXTRA_SISWA = "extra_siswa"
        const val ALERT_DIALOG_CLOSE = 10
        const val ALERT_DIALOG_DELETE = 20
    }
    private var isEdit = false
    private var siswa: Siswa? = null

    private lateinit var siswaAddUpdateViewModel: SiswaAddUpdateViewModel
    private var _activitySiswaAddUpdateBinding: ActivityNoteAddUpdateBinding? = null
    private val binding get() = _activitySiswaAddUpdateBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        _activitySiswaAddUpdateBinding = ActivityNoteAddUpdateBinding.inflate(layoutInflater)
        setContentView(binding?.root)
        siswaAddUpdateViewModel = obtainViewModel( activity: this@SiswaAddUpdateActivity)

        siswa = intent.getParcelableExtra(EXTRA_SISWA)
        if (siswa != null){
            isEdit = true
```

```
        }else{
            siswa = Siswa()
        }
        val actionBarTitle: String
        val btnTitle: String
        if (isEdit){
            actionBarTitle = getString(R.string.change)
            btnTitle = getString(R.string.update)
            if (siswa != null){
                siswa?.let { note ->
                    binding?.edtTitle?.setText(note.namaLengkap)
                    binding?.edtDescription?.setText(note.nis)

                }
            }
        }else{
            actionBarTitle= getString(R.string.add)
            btnTitle = getString(R.string.save)
        }
        supportActionBar?.title = actionBarTitle
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        binding?.btnSubmit?.text = btnTitle

        binding?.btnSubmit?.setOnClickListener{
            val title = binding?.edtTitle?.text.toString().trim()
            val description = binding?.edtDescription?.text.toString().trim()
            when{
                title.isEmpty() -> {
                    binding?.edtTitle?.error = getString(R.string.empty)
                }
                description.isEmpty() -> {
                    binding?.edtDescription?.error = getString(R.string.empty)
                }
                else -> {
                    siswa.let { note ->
                        note?.namaLengkap = title
                        note?.nis = description
                    }
                    if (isEdit){
                        siswaAddUpdateViewModel.update(siswa as Siswa)
                        showToast(getString(R.string.changed))
                    }else{
                        siswa.let { note ->
                            note?.date = DateHelper.getCurrentDate()
                        }
                        siswaAddUpdateViewModel.insert(siswa as Siswa)
                        showToast(getString(R.string.added))
                    }
                    finish()
```

```
88                }
89
90            }
91        }
92        onBackPressedDispatcher.addCallback( owner: this, object : OnBackPressedCallback( enabled: true){
93            override fun handleOnBackPressed() {
94                showAlertDialog(ALERT_DIALOG_CLOSE)
95            }
96        })
97    }
98
99
        Edit | Explain | Test | Document | Fix
100   override fun onCreateOptionsMenu(menu: Menu?): Boolean {
101       if (isEdit){
102           menuInflater.inflate(R.menu.menu,menu)
103       }
104       return super.onCreateOptionsMenu(menu)
105   }
106
        Edit | Explain | Test | Document | Fix
107   override fun onOptionsItemSelected(item: MenuItem): Boolean {
108       when (item.itemId){
109           R.id.action_delete -> showAlertDialog(ALERT_DIALOG_DELETE)
110           android.R.id.home -> showAlertDialog(ALERT_DIALOG_CLOSE)
111       }
112       return super.onOptionsItemSelected(item)
113   }
114
        Edit | Explain | Test | Document | Fix
115   private fun showAlertDialog(type: Int){
116       val isDialogClose = type == ALERT_DIALOG_CLOSE
117       val dialogTitle: String
118       val dialogMessage: String
119       if (isDialogClose){
120           dialogTitle = getString(R.string.cancel)
121           dialogMessage = getString(R.string.message_cancel)
122       }else{
123           dialogMessage = getString(R.string.message_delete)
124           dialogTitle = getString(R.string.delete)
125       }
126       val alertDialogBuilder = AlertDialog.Builder( context: this)
127       with(alertDialogBuilder){
128           setTitle(dialogTitle)
129           setMessage(dialogMessage)
130           setCancelable(false)
131           setPositiveButton(getString(R.string.yes)) { _, _ ->
132               if (!isDialogClose){
```



```
                siswaAddUpdateViewModel.delete(siswa as Siswa)
                showToast(getString(R.string.deleted))
            }
            finish()
        }
        setNegativeButton(getString(R.string.no)) { dialog, _ -> dialog.cancel() }
    }
    val alertDialog = alertDialogBuilder.create()
    alertDialog.show()
}
private fun showToast(message: String) {
    Toast.makeText( context: this, message, Toast.LENGTH_SHORT).show()
}
Edit | Explain | Test | Document | Fix
override fun onDestroy() {
    super.onDestroy()
    _activitySiswaAddUpdateBinding = null
}
Edit | Explain | Test | Document | Fix
private fun obtainViewModel(activity: AppCompatActivity): SiswaAddUpdateViewModel{
    val factory = ViewModelFactory.getInstance(activity.application)
    return ViewModelProvider(activity, factory).get(SiswaAddUpdateViewModel::class.java)
}
}
```

3. `SiswaAddUpdateActivity.kt`

Fungsi:

Menampilkan form untuk menambah atau mengedit catatan.

Penjelasan:

- Menentukan apakah aktivitas dalam mode tambah (`add`) atau ubah (`edit`) berdasarkan `Intent`.
- Validasi input sebelum menyimpan.
- Menampilkan dialog konfirmasi saat keluar atau menghapus.
- Berinteraksi dengan `SiswaAddUpdateViewModel` untuk insert, update, atau delete.

SiswaAddUpdateViewModel.kt

```kotlin
package com.example.mylistsiswa.ui.insert

import android.app.Application
import androidx.lifecycle.ViewModel
import com.example.mylist.database.Siswa
import com.example.mylistsiswa.repository.SiswaRepository

Edit | Explain | Test | Document | Fix
class SiswaAddUpdateViewModel(application: Application): ViewModel() {
    private val mSiswaRepository: SiswaRepository = SiswaRepository(application)
    fun insert(siswa: Siswa){
        mSiswaRepository.insert(siswa)
    }
    fun update(siswa: Siswa){
        mSiswaRepository.update(siswa)
    }
    fun delete(siswa: Siswa) {
        mSiswaRepository.delete(siswa)
    }
}
```

4. `SiswaAddUpdateViewModel.kt`

Fungsi:

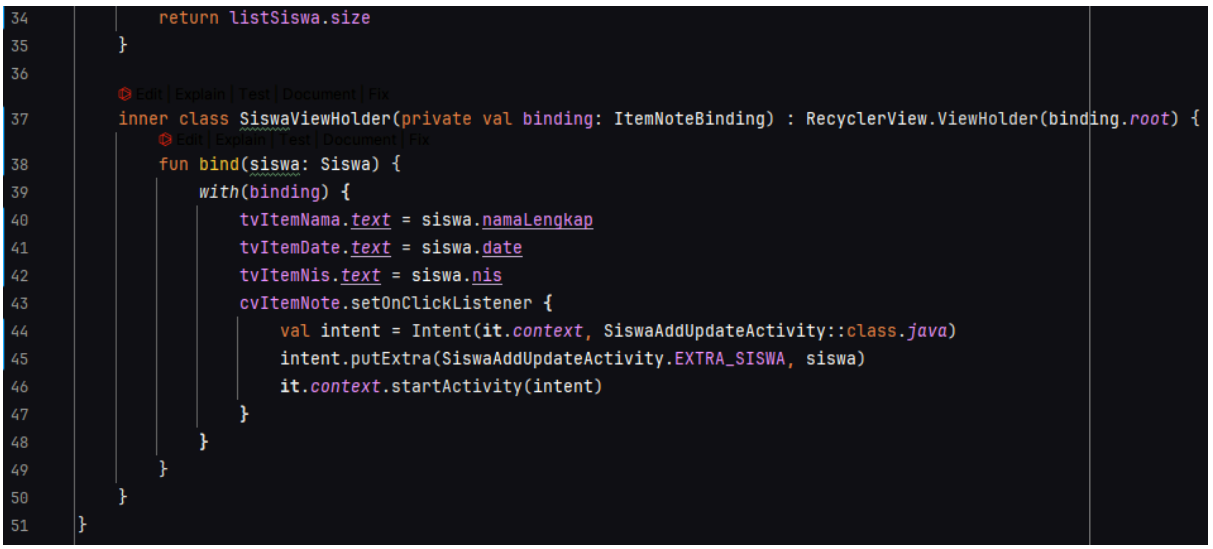
ViewModel untuk `SiswaAddUpdateActivity`.

Penjelasan:

- Bertanggung jawab mengelola data list dari repository untuk operasi **insert**, **update**, dan **delete**.
- Memastikan logika bisnis tidak berada di dalam `Activity`.

SiswaAdapter.kt

```kotlin
package com.example.mylistsiswa.ui.main

import ...

class SiswaAdapter : RecyclerView.Adapter<SiswaAdapter.SiswaViewHolder>() {
    private val listSiswa = ArrayList<Siswa>()

    fun setListSiswa(listSiswa: List<Siswa>) {
        val diffCallback = SiswaDiffCallback(this.listSiswa, listSiswa)
        val diffResult = DiffUtil.calculateDiff(diffCallback)
        this.listSiswa.clear()
        this.listSiswa.addAll(listSiswa)
        diffResult.dispatchUpdatesTo( adapter: this)
    }

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

    override fun onBindViewHolder(holder: SiswaViewHolder, position: Int) {
        holder.bind(listSiswa[position])
    }

    override fun getItemCount(): Int {
```



```
34          return listSiswa.size
35      }
36
        Edit | Explain | Test | Document | Fix
37      inner class SiswaViewHolder(private val binding: ItemNoteBinding) : RecyclerView.ViewHolder(binding.root) {
            Edit | Explain | Test | Document | Fix
38          fun bind(siswa: Siswa) {
39              with(binding) {
40                  tvItemNama.text = siswa.namaLengkap
41                  tvItemDate.text = siswa.date
42                  tvItemNis.text = siswa.nis
43                  cvItemNote.setOnClickListener {
44                      val intent = Intent(it.context, SiswaAddUpdateActivity::class.java)
45                      intent.putExtra(SiswaAddUpdateActivity.EXTRA_SISWA, siswa)
46                      it.context.startActivity(intent)
47                  }
48              }
49          }
50      }
51  }
```

5. `SiswaAdapter.kt`

Fungsi:

Adapter untuk `RecyclerView` yang menampilkan daftar list.

Penjelasan:

- Mengatur binding setiap item List (`Siswa`) ke dalam tampilan list.
- Menggunakan `DiffUtil` untuk optimalisasi performa saat data berubah.
- Menangani event klik item untuk membuka `SiswaAddUpdateActivity` (edit mode).

ViewModelVactory.kt

```kotlin
package com.example.mylistsiswa.helper

import android.app.Application
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import com.example.mylistsiswa.ui.insert.SiswaAddUpdateViewModel
import com.example.mylistsiswa.ui.main.MainViewModel

Edit | Explain | Test | Document | Fix
class ViewModelFactory private constructor(private val mApplication: Application):
    ViewModelProvider.NewInstanceFactory(){

    companion object{
        @Volatile
        private var INSTANCE: ViewModelFactory? = null
        Edit | Explain | Test | Document | Fix
        @JvmStatic
        fun getInstance(application: Application): ViewModelFactory{
            if (INSTANCE == null){
                synchronized(ViewModelFactory::class.java)
                {
                    INSTANCE = ViewModelFactory(application)
                }
            }
            return INSTANCE as ViewModelFactory
        }
    }
```

6. `ViewModelFactory.kt`

Fungsi:

Membantu pembuatan `ViewModel` yang memerlukan parameter `Application`.

Penjelasan:

- Menggunakan pola Singleton agar `ViewModelFactory` hanya dibuat satu kali.
- Diperlukan karena `ViewModel` menerima `Application` di konstruktor, bukan default constructor.

SiswaRepository.kt

```kotlin
package com.example.mylistsiswa.repository

import ...

class SiswaRepository(application: Application) {
    private val mSiswaDao: SiswaDao
    private val executorService: ExecutorService = Executors.newSingleThreadExecutor()
    init {
        val db = ListSiswaDatabase.getDatabase(application)
        mSiswaDao = db.siswaDao()
    }
    fun getAllSiswa(): LiveData<List<Siswa>> = mSiswaDao.getAllSiswa()
    fun insert(siswa: Siswa){
        executorService.execute { mSiswaDao.insert(siswa) }
    }
    fun delete(siswa: Siswa){
        executorService.execute { mSiswaDao.delete(siswa) }
    }
    fun update(siswa: Siswa){
        executorService.execute { mSiswaDao.update(siswa) }
    }
}
```

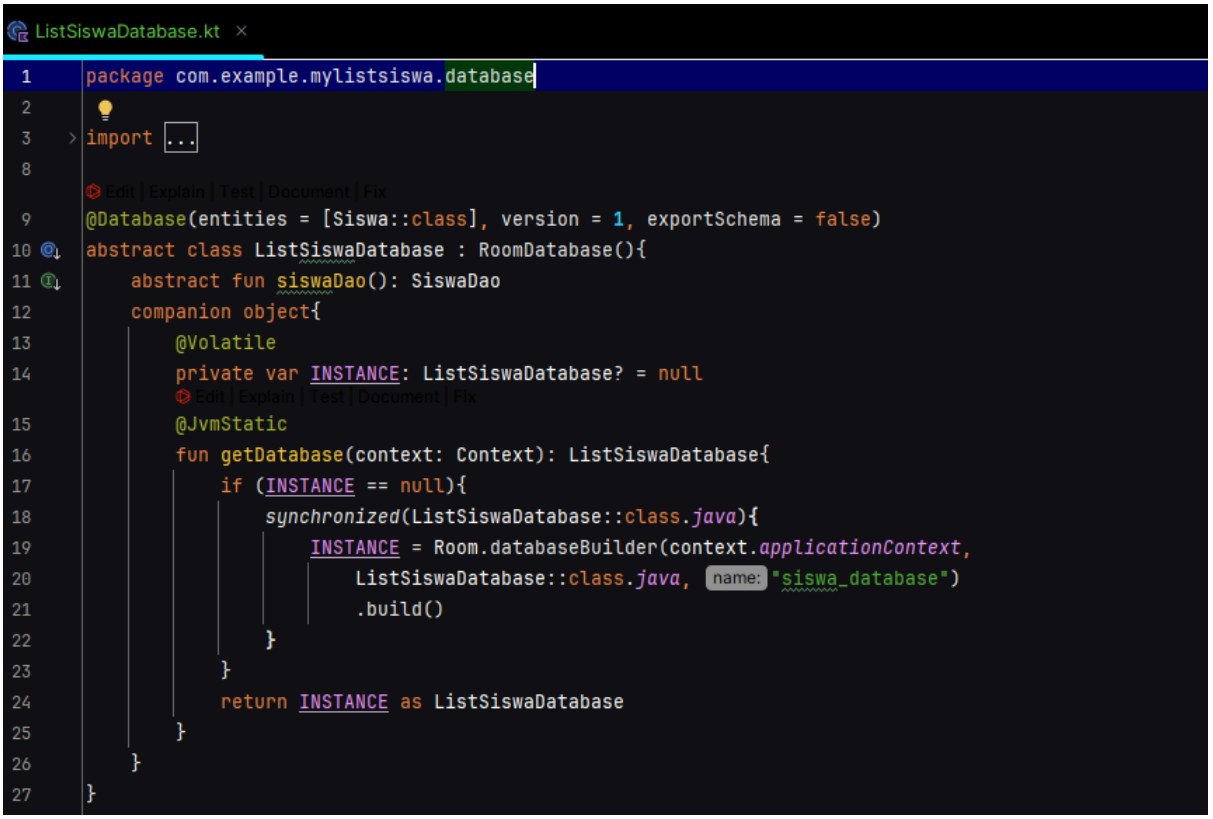
7. `SiswaRepository.kt`

Fungsi:

Lapisan abstraksi antara `ViewModel` dan Room.

Penjelasan:

- Menyediakan method untuk operasi CRUD: `insert`, `update`, `delete`, dan `getAllSiswa`.
- Menggunakan `ExecutorService` agar operasi database berjalan di background thread (bukan UI thread).



8. `ListSiswaDatabase.kt`

Fungsi:

Membangun dan menyediakan instance `RoomDatabase`.

Penjelasan:

- Singleton untuk instance `Room`.
- Berisi method `SiswaDao()` untuk akses DAO.

SiswaDao.kt

```kotlin
package com.example.mylistsiswa.database

import androidx.lifecycle.LiveData
import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.Update
import com.example.mylist.database.Siswa

Edit | Explain | Test | Document | Fix
@Dao
interface SiswaDao {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    fun insert(siswa: Siswa)
    @Update
    fun update(siswa: Siswa)
    @Delete
    fun delete(siswa: Siswa)


    @Query("SELECT * from siswa ORDER BY id ASC")
    fun getAllSiswa(): LiveData<List<Siswa>>
}
```

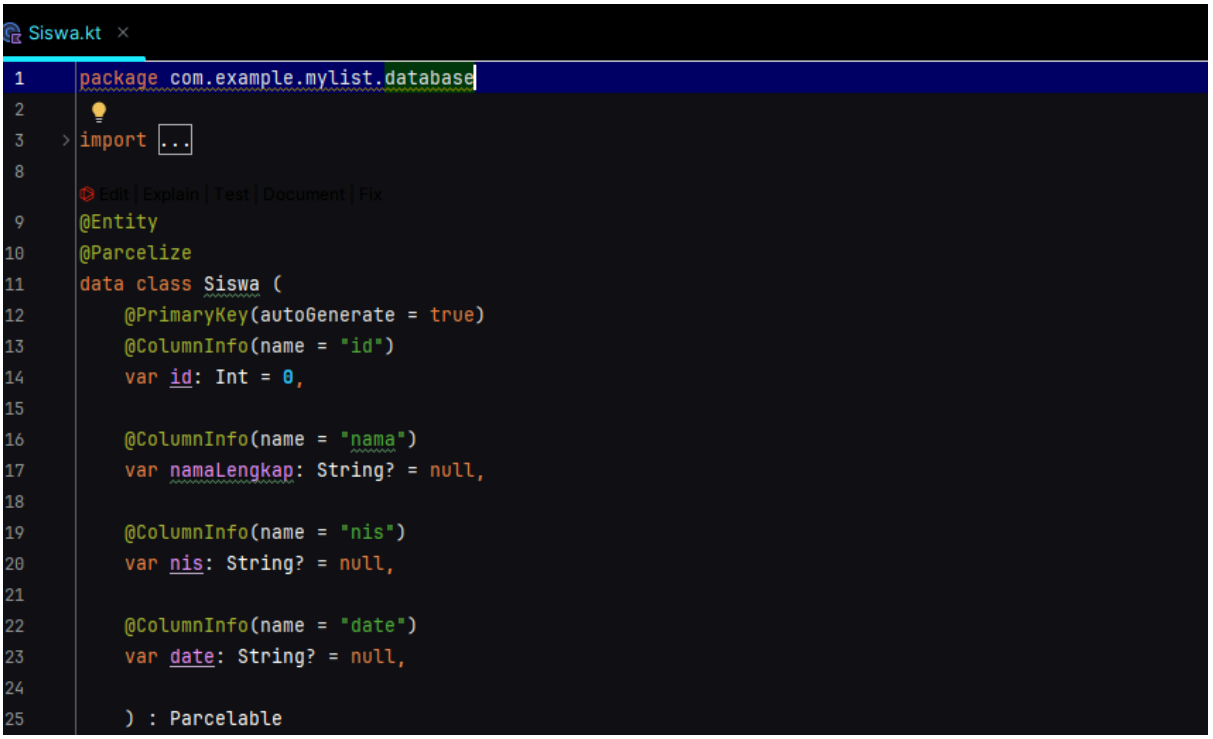
9. `SiswaDao.kt`

Fungsi:

Interface DAO (Data Access Object) untuk database catatan.

Penjelasan:

- Menyediakan query SQL untuk `insert`, `update`, `delete`, dan `getAllSiswa()`.
- `getAllSiswa()` mengembalikan `LiveData<List<Siswa>>` agar bisa di-observe dari UI.



10. `Siswa.kt`

Fungsi:

Model (Entity) untuk entitas `Siswa` dalam database.

Penjelasan:

- Menggunakan anotasi Room (`@Entity`, `@PrimaryKey`) agar bisa disimpan ke database.
- Implementasi `Parcelable` agar `Siswa` bisa dikirim lewat `Intent`.

DateHelper.kt

```kotlin
package com.example.mylistsiswa.helper

import java.text.SimpleDateFormat
import java.util.Date
import java.util.Locale

object DateHelper {
    Edit | Explain | Test | Document | Fix
    fun getCurrentDate(): String{
        val dateFormat = SimpleDateFormat( pattern: "yyyy/MM/dd HH:mm:ss", Locale.getDefault())
        val date = Date()
        return dateFormat.format(date)

    }
}
```

11. DateHelper.kt

Fungsi:

Menyediakan utilitas untuk mengambil tanggal saat ini.

Penjelasan:

- Mengembalikan tanggal dengan format `yyyy/MM/dd` untuk ditampilkan atau disimpan ke list.

SiswaDiffCallback.kt

```kotlin
package com.example.mylistsiswa.helper

import androidx.recyclerview.widget.DiffUtil
import com.example.mylist.database.Siswa

class SiswaDiffCallback(private val oldSiswaList: List<Siswa>, private val newSiswaList: List<Siswa>) :
    DiffUtil.Callback() {
    override fun getOldListSize(): Int = oldSiswaList.size
    override fun getNewListSize(): Int = newSiswaList.size
    override fun areItemsTheSame(oldItemPosition: Int, newItemPosition: Int): Boolean {
        return oldSiswaList[oldItemPosition].id == newSiswaList[newItemPosition].id
    }

    override fun areContentsTheSame(oldItemPosition: Int, newItemPosition: Int): Boolean {
        val oldSiswa = oldSiswaList[oldItemPosition]
        val newSiswa = newSiswaList[newItemPosition]
        return oldSiswa.namaLengkap == newSiswa.namaLengkap && oldSiswa.nis == newSiswa.nis
    }
}
```

12. `SiswaDiffCallback.kt`

Fungsi:

Membandingkan 2 list `Siswa` untuk update efisien di `RecyclerView`.

Penjelasan:

- Dipakai oleh `DiffUtil` dalam `SiswaAdapter`.
- Menentukan apakah data dan konten item sama, untuk menghindari reload semua list.

Deskripsi Aplikasi

Aplikasi ini merupakan aplikasi **list data siswa** berbasis Android, yang dibuat menggunakan **Room Database** untuk menyimpan data secara lokal. Setiap entri data terdiri dari:

- **Nama Lengkap**
- **NIS (Nomor Induk Siswa)**
- **Tanggal Input Otomatis** (menggunakan format `yyyy/MM/dd HH:mm:ss`)

Fitur Utama

1. **Tambah Data Siswa**
   Pengguna dapat menambahkan data siswa berupa nama lengkap dan NIS.

2. **Edit Data**
   Pengguna dapat mengubah data siswa yang sudah ada.

3. **Hapus Data**
   Data siswa dapat dihapus setelah konfirmasi dari pengguna.

4. **Pencarian Real-Time**
   Fitur pencarian memudahkan pengguna mencari siswa berdasarkan nama atau NIS.

5. **Penyimpanan Offline (Room)**
   Data disimpan secara lokal menggunakan Room, sehingga tetap tersedia tanpa koneksi internet.

Alur Aplikasi

1. **MainActivity**
    - Menampilkan daftar siswa dalam bentuk list.
    - Menyediakan tombol **tambah data** dan **fitur pencarian**.

2. **SiswaAddUpdateActivity**
    - Digunakan untuk menambahkan atau mengedit data siswa.
    - Menampilkan form dengan dua input: Nama Lengkap dan NIS.
    - Menyimpan atau memperbarui data ke Room Database.
3. **SiswaAdapter**
    - Menangani tampilan list siswa di RecyclerView.
    - Saat item diklik, akan membuka halaman edit (SiswaAddUpdateActivity).