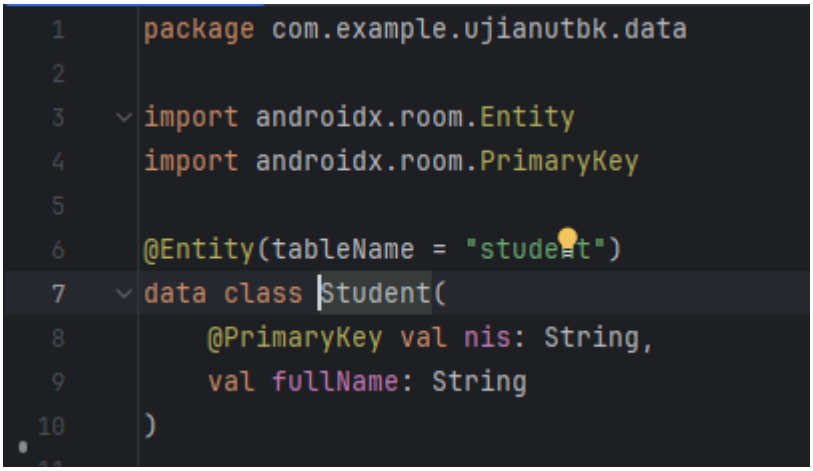
Dokumentasi Aplikasi CRUD Data Siswa (Room + MVVM + Search)

Struktur Proyek

Aplikasi ini dibangun dengan arsitektur **MVVM**. Tujuannya biar kodenya rapi, gampang dikembangin, dan gak semua logic ditumpuk di satu tempat.

- **Model**: ngurusin struktur data dan akses database
- **ViewModel**: jembatan antara UI dan data
- **Repository**: perantara antara ViewModel dan DAO
- **View (Activity)**: tampilan buat user
- **Adapter**: nampilin data ke RecyclerView

Model: Student.kt



```kotlin
package com.example.ujianutbk.data

import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "student")
data class Student(
    @PrimaryKey val nis: String,
    val fullName: String
)
```

Model `Student` mewakili 1 entitas data siswa yang bakal disimpen ke database. `nis` sebagai ID unik.

DAO: StudentDao.kt

```kotlin
@Dao
interface StudentDao {
    @Query("SELECT * FROM student ORDER BY fullName ASC")
    fun getAll(): LiveData<List<Student>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(student: Student)

    @Update
    suspend fun update(student: Student)

    @Delete
    suspend fun delete(student: Student)

    @Query("SELECT * FROM student WHERE fullName LIKE :query OR nis LIKE :query")
    fun search(query: String): LiveData<List<Student>>
}
```

DAO ini nyediain perintah buat ambil, cari, nambah, update, dan hapus data siswa di database.

Database: StudentDatabase.kt

```kotlin
@Database(entities = [Student::class], version = 1, exportSchema = false)
abstract class StudentDatabase : RoomDatabase() {
    abstract fun studentDao(): StudentDao

    companion object {
        @Volatile private var INSTANCE: StudentDatabase? = null

        fun getInstance(context: Context): StudentDatabase =
            INSTANCE ?: synchronized( lock: this) {
                INSTANCE ?: Room.databaseBuilder(
                    context.applicationContext,
                    StudentDatabase::class.java,
                     name: "student_db"
                ).build().also { INSTANCE = it }
            }
    }
}
```

File ini ngebuat instance database Room dan nyediain akses ke DAO-nya.

Repository: StudentRepository.kt

```
class StudentRepository(private val dao: StudentDao) {
    val allStudents: LiveData<List<Student>> = dao.getAll()

    suspend fun insert(student: Student) = dao.insert(student)
    suspend fun update(student: Student) = dao.update(student)
    suspend fun delete(student: Student) = dao.delete(student)

    fun search(query: String): LiveData<List<Student>> = dao.search( query: "%$query%")
}
```

Repository ini jadi jembatan antara ViewModel dan DAO, biar ViewModel gak langsung kontak database.

ViewModel: StudentViewModel.kt

```
class StudentViewModel(private val repo: StudentRepository) : ViewModel() {
    val allStudents: LiveData<List<Student>> = repo.allStudents

    fun insert(student: Student) = viewModelScope.launch {
        repo.insert(student)
    }

    fun update(student: Student) = viewModelScope.launch {
        repo.update(student)
    }

    fun delete(student: Student) = viewModelScope.launch {
        repo.delete(student)
    }

    fun search(query: String): LiveData<List<Student>> = repo.search(query)
}
```

ViewModel ini ngatur logika antara UI dan repository. Dia yang handle semua aksi insert, update, delete, dan cari.

ViewModelFactory: StudentViewModelFactory.kt

```
class StudentViewModelFactory(private val repo: StudentRepository) : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(StudentViewModel::class.java)) {
            @Suppress( ...names: "UNCHECKED_CAST")
            return StudentViewModel(repo) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```

Factory ini dipake buat ngasih repository ke ViewModel karena ViewModel-nya butuh parameter.

Adapter: StudentAdapter.kt

```
class StudentAdapter(
    private val onEdit: (Student) -> Unit,
    private val onDelete: (Student) -> Unit
) : ListAdapter<Student, StudentAdapter.ViewHolder>(DiffCallback()) {

    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val tvNis: TextView = view.findViewById(R.id.tvNis)
        val tvName: TextView = view.findViewById(R.id.tvName)
        val btnEdit: ImageButton = view.findViewById(R.id.btnEdit)
        val btnDelete: ImageButton = view.findViewById(R.id.btnDelete)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_student, parent,  attachToRoot: false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val student = getItem(position)
        holder.tvNis.text = student.nis
        holder.tvName.text = student.fullName
        holder.btnEdit.setOnClickListener { onEdit(student) }
        holder.btnDelete.setOnClickListener { onDelete(student) }
    }

    class DiffCallback : DiffUtil.ItemCallback<Student>() {
        override fun areItemsTheSame(old: Student, new: Student) = old.nis == new.nis
        override fun areContentsTheSame(old: Student, new: Student) = old == new
    }
}
```

Adapter ini dipake buat nampilin data siswa ke RecyclerView, dan nangkep event tombol edit dan delete.

MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding
    private lateinit var adapter: StudentAdapter
    private var isEditMode = false
    private var studentToEdit: Student? = null

    private val viewModel: StudentViewModel by viewModels {
        StudentViewModelFactory(StudentRepository(StudentDatabase.getInstance( context: this).studentDao()))
    }

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

        adapter = StudentAdapter(
            onEdit = { student ->
                AlertDialog.Builder( context: this).apply {
                    setTitle("Edit Siswa")
                    setMessage("Yakin mau edit data ${student.fullName}?")
                    setPositiveButton( text: "Iya") { _, _ ->
                        binding.etNis.setText(student.nis)
                        binding.etName.setText(student.fullName)
                        binding.etNis.isEnabled = false // Biar NIS gak bisa diedit
                        isEditMode = true
                        studentToEdit = student
                    }
                    setNegativeButton( text: "Batal",  listener: null)
                }.show()
            },
            onDelete = { student ->
                AlertDialog.Builder( context: this).apply {
                    setTitle("Hapus Siswa")
                    setMessage("Yakin mau hapus data ${student.fullName}?")
                    setPositiveButton( text: "Hapus") { _, _ ->
                        viewModel.delete(student)
                    }
                    setNegativeButton( text: "Batal",  listener: null)
                }.show()
            }
        )

        binding.rvStudents.layoutManager = LinearLayoutManager( context: this)
        binding.rvStudents.adapter = adapter

        viewModel.allStudents.observe( owner: this) {
            adapter.submitList(it)
        }
```

```
    binding.btnAdd.setOnClickListener {
        val nis = binding.etNis.text.toString()
        val name = binding.etName.text.toString()

        if (nis.isNotEmpty() && name.isNotEmpty()) {
            val student = Student(nis, name)
            if (isEditMode) {
                viewModel.update(student)
                isEditMode = false
                studentToEdit = null
                binding.etNis.isEnabled = true
            } else {
                viewModel.insert(student)
            }
            binding.etNis.text.clear()
            binding.etName.text.clear()
        }
    }

    binding.etSearch.addTextChangedListener {
        val query = it.toString()
        viewModel.search(query).observe( owner: this) { result ->
            adapter.submitList(result)
        }
    }
}
```


Activity ini ngurusin semua interaksi user: nambah, cari, edit, dan hapus data. Juga ngelink-in semua ViewModel ke UI.