Dokumentasi Project SiswaApps

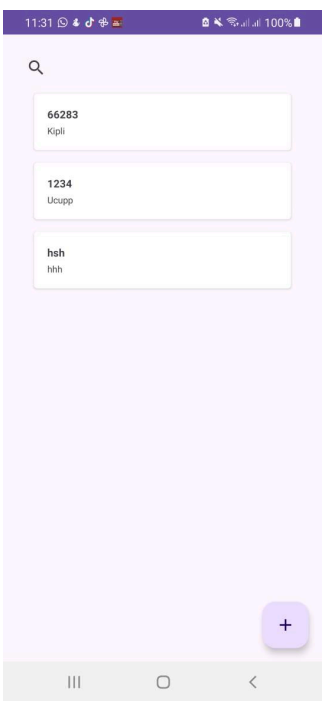
Deskripsi Project

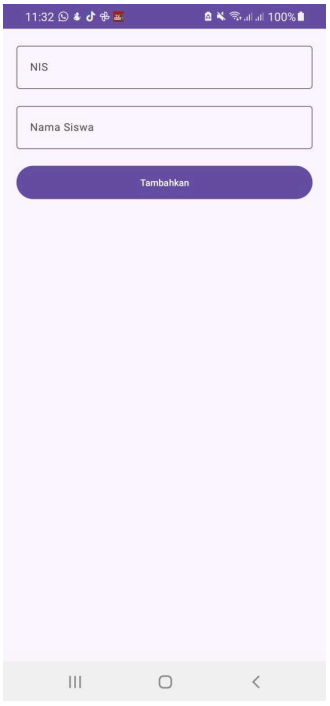
Dalam proyek ini, saya membuat sebuah aplikasi manajemen data siswa menggunakan bahasa pemrograman Kotlin. Aplikasi ini dirancang untuk mempermudah pengelolaan informasi siswa dengan menyediakan fitur-fitur utama seperti input data, pencarian, pembaruan (update), dan penghapusan data siswa. Data yang dikelola mencakup Nomor Induk Siswa (NIS) dan Nama Lengkap, yang kemudian ditampilkan dalam bentuk daftar menggunakan komponen ListView atau RecyclerView untuk antarmuka yang responsif dan efisien.

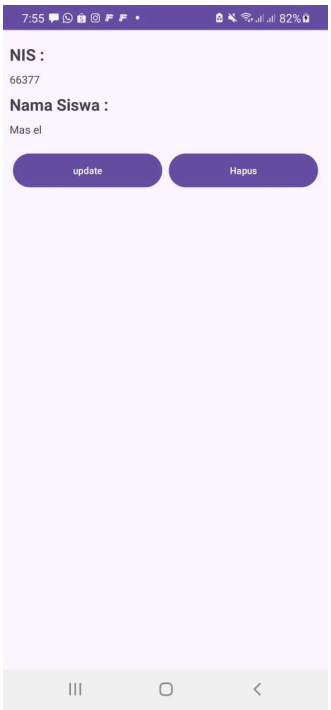
Fitur Aplikasi

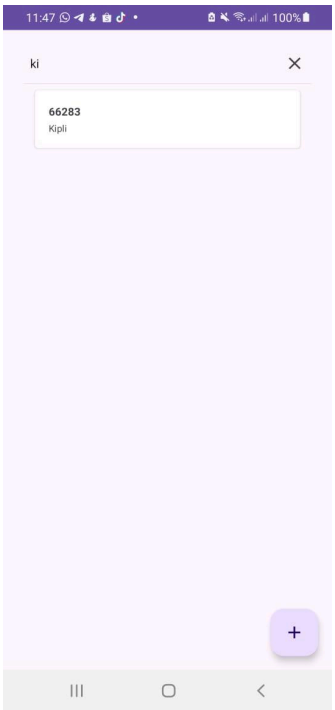
1. **Input Data Siswa**: Memungkinkan pengguna untuk menambahkan data siswa baru dengan rincian NIS dan Nama Lengkap.
2. **Tampilan Daftar**: Menampilkan seluruh data siswa dalam daftar yang diurutkan berdasarkan nama secara ascending.
3. **Pencarian**: Fitur untuk mencari siswa berdasarkan NIS atau Nama Lengkap menggunakan query.
4. **Update dan Delete**: Memungkinkan pengguna untuk memperbarui atau menghapus data siswa yang sudah ada.

Tampilan UI


11:31
100%
66283
Kipli
1234
Ucupp
hsh
hhh


11:32
100%
NIS
Nama Siswa
Tambahkan


7:55
82%
NIS :
66377
Nama Siswa :
Mas el
update
Hapus


11:47
100%
ki
66283
Kipli

Kode Aplikasi

1. Student.kt

```
@Entity(tableName = "student")
data class Student(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id")
    val id: Int = 0,

    @ColumnInfo(name = "nis")
    val nis: String,

    @ColumnInfo(name = "full_name")
    val fullName: String
)
```

Class Student didefinisikan sebagai entitas dengan anotasi @Entity(tableName = "student"), yang merepresentasikan tabel dalam database Room. Setiap objek Student memiliki atribut:

- id (Primary Key, auto-generate),
- nis (Nomor Induk Siswa),
- fullName (Nama Lengkap).

Class ini menjadi dasar penyimpanan data siswa dalam bentuk baris pada tabel "student".

2. StudentDao.kt

```
@Dao
interface StudentDao {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insert(student: Student)

    @Delete
    suspend fun delete(student: Student)

    @Update
    suspend fun update(student: Student)

    @Query("SELECT * FROM student ORDER BY full_name ASC")
    fun getAllStudents(): LiveData<List<Student>>
```

```
    @Query("SELECT * FROM student WHERE nis LIKE :query OR full_name
LIKE :query ORDER BY full_name ASC")
    fun searchStudents(query: String): LiveData<List<Student>>

    @Query("SELECT * FROM student WHERE id = :id")
    fun getStudentById(id: Int): LiveData<Student>
}
```

StudentDao berisi method - method untuk mengakses dan memanipulasi data pada database seperti:

- insert(): Menambahkan data siswa baru (mengabaikan konflik jika data sudah ada).
- delete(): Menghapus data siswa.
- update(): Memperbarui data siswa.
- getAllStudents(): Mengambil semua data siswa, diurutkan berdasarkan nama (menggunakan LiveData untuk pembaruan real-time).
- searchStudents(query): Mencari siswa berdasarkan NIS atau nama.
- getStudentById(id): Mengambil data siswa berdasarkan ID.

Semua method yang memodifikasi data bersifat suspend agar dapat dijalankan di background thread melalui coroutine, sehingga tidak mengganggu thread UI.

3. StudentDatabase.kt

```
@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 getDatabase(context: Context): StudentDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    StudentDatabase::class.java,
                    "student_database"
                ).build()
                INSTANCE = instance
                instance
            }
        }
```

```
    }
}
```

class abstract ini mendefinisikan database Room dengan entitas Student dan versi 1. Menggunakan pola Singleton melalui function getDatabase(), hanya satu instance database yang dibuat untuk menghemat sumber daya. Database ini menyediakan akses ke StudentDao.

4. StudentRepository.kt

```
class StudentRepository(private val studentDao: StudentDao) {

    fun getAllStudents(): LiveData<List<Student>> =
studentDao.getAllStudents()

    fun getStudentById(id: Int): LiveData<Student> {
        return studentDao.getStudentById(id)
    }

    fun searchStudents(query: String): LiveData<List<Student>> {
        val searchQuery = "%$query%"
        return studentDao.searchStudents(searchQuery)
    }

    suspend fun insert(student: Student) {
        studentDao.insert(student)
    }

    suspend fun update(student: Student) {
        studentDao.update(student)
    }

    suspend fun delete(student: Student) {
        studentDao.delete(student)
    }

}
```

Class StudentRepository bertindak sebagai penghubung antara StudentDao dan ViewModel. Function-function utamanya meliputi:

- Mengambil semua data siswa (getAllStudents()),
- Mencari siswa berdasarkan query (searchStudents()),
- Mengambil siswa berdasarkan ID (getStudentById()),
- Operasi CRUD (insert, update, delete) yang dijalankan secara asynchronus dengan coroutine.

5. SiswaApps.kt

```
class SiswaApps: Application() {
    // untuk inisialisasi database dan repository
    val database by lazy { StudentDatabase.getDatabase(this) }
    val repository by lazy { StudentRepository(database.studentDao()) }
}
```

Class ini merupakan turunan dari Application yang menginisialisasi database dan repository secara global menggunakan by lazy. Pendekatan ini memastikan inisialisasi hanya dilakukan saat pertama kali dibutuhkan, mengoptimalkan penggunaan resource.

6. StudentViewModel

```
class StudentViewModel(private val repository: StudentRepository) :
ViewModel() {
    fun getAllStudents(): LiveData<List<Student>> =
repository.getAllStudents()

    fun searchStudents(query: String): LiveData<List<Student>> =
repository.searchStudents(query)

    fun getStudentById(id: Int): LiveData<Student> {
        return repository.getStudentById(id)
    }

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

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

    fun update(student: Student) {
        viewModelScope.launch {
            repository.update(student)
        }
    }
```

```
}
```

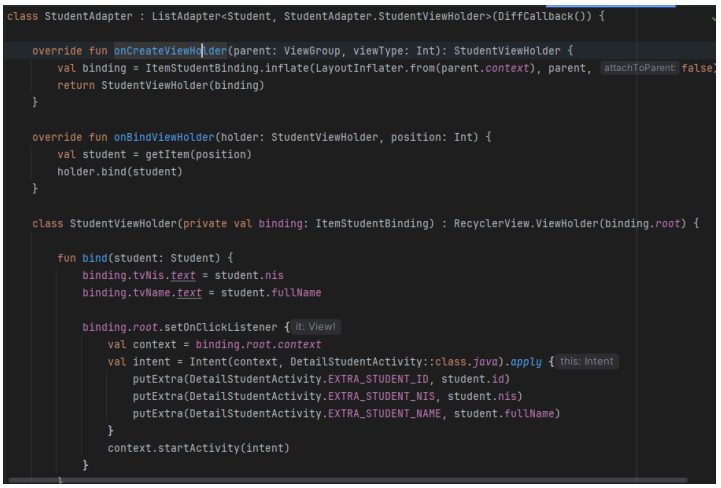
StudentViewModel menghubungkan UI dengan StudentRepository. Menggunakan LiveData, class ini menyediakan data secara reaktif ke antarmuka pengguna dan menangani operasi seperti insert, update, dan delete melalui viewModelScope untuk eksekusi asynchronus.

7. StudentViewModelFactory

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

Class ini digunakan untuk membuat instance StudentViewModel dengan menyediakan StudentRepository sebagai dependensi.

8. StudentAdapter



```
    }

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

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

StudentAdapter bertugas menampilkan daftar siswa dalam RecyclerView. Dengan memanfaatkan ListAdapter, adapter ini secara otomatis mendeteksi perubahan data dan memperbarui tampilan.

9. MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding
    private lateinit var studentAdapter: StudentAdapter
    private val studentViewModel: StudentViewModel by viewModels{
        StudentViewModelFactory((application as SiswaApps).repository)
    }

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

        binding.fabAdd.setOnClickListener { it: View!
            val intent = Intent( packageContext: this, AddUpdateStudentActivity::class.java)
            startActivity(intent)
        }

        getListStudent()
        searchStudent()
    }
```

```
private fun getListStudent() {
    studentAdapter = StudentAdapter()
    binding.rvStudent.apply { this: RecyclerView
        layoutManager = LinearLayoutManager( context: this@MainActivity)
        adapter = studentAdapter
    }

    studentViewModel.getAllStudents().observe( owner: this) { students ->
        studentAdapter.submitList(students)
    }
}
```

```
    private fun searchStudent() {
        binding.searchBar.setOnQueryTextListener(object : SearchView.OnQueryTextListener {
            override fun onQueryTextSubmit(query: String?): Boolean {
                return true
            }

            override fun onQueryTextChange(newText: String?): Boolean {
                val query = newText?.trim() ?: ""
                if (query.isNotEmpty()) {
                    studentViewModel.searchStudents(query).observe( owner: this@MainActivity) { students ->
                        studentAdapter.submitList(students)
                    }
                } else {
                    studentViewModel.getAllStudents().observe( owner: this@MainActivity) { students ->
                        studentAdapter.submitList(students)
                    }
                }
                return true
            }
        })
    }
}
```

MainActivity merupakan activity utama yang menampilkan daftar siswa dan menyediakan tombol untuk menambah, memperbarui, atau menghapus data. Activity ini terhubung dengan StudentViewModel untuk mengelola data secara real-time.

10. AddUpdateStudentActivity.kt

```
class AddUpdateStudentActivity : AppCompatActivity() {

    private lateinit var binding: ActivityAddUpdateStudentBinding
    private val studentViewModel: StudentViewModel by viewModels {
        StudentViewModelFactory((application as SiswaApps).repository)
    }
    private var studentId: Int = 0

    companion object {
        const val EXTRA_STUDENT_ID = "extra_student_id"
        const val EXTRA_STUDENT_NIS = "extra_student_nis"
        const val EXTRA_STUDENT_NAME = "extra_student_name"
    }

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

        checkUpdateMode()
        saveOrUpdate()
    }
```

```
private fun checkUpdateMode() {
    studentId = intent.getIntExtra(EXTRA_STUDENT_ID, defaultValue: 0)
    val nis = intent.getStringExtra(EXTRA_STUDENT_NIS)
    val name = intent.getStringExtra(EXTRA_STUDENT_NAME)

    if (studentId != 0 && nis != null && name != null) {
        binding.edtNis.setText(nis)
        binding.edtName.setText(name)
        binding.btnAdd.text = "Update"
    } else {
        binding.btnAdd.text = "Add"
    }
}
```

```
private fun saveOrUpdate() {
    binding.btnAdd.setOnClickListener { it: View!
        val nis = binding.edtNis.text.toString().trim()
        val fullName = binding.edtName.text.toString().trim()

        if (nis.isEmpty() || fullName.isEmpty()) {
            Toast.makeText( context: this, text: "NIS dan Nama Lengkap harus diisi", Toast.LENGTH_SHORT).show()
            return@setOnClickListener
        }

        if (studentId != 0) {
            val updatedStudent = Student(id = studentId, nis = nis, fullName = fullName)
            studentViewModel.update(updatedStudent)
            Toast.makeText( context: this, text: "Data siswa berhasil diupdate", Toast.LENGTH_SHORT).show()
        } else {
            val newStudent = Student(nis = nis, fullName = fullName)
            studentViewModel.insert(newStudent)
            Toast.makeText( context: this, text: "Data siswa berhasil disimpan", Toast.LENGTH_SHORT).show()
        }
        finish()
    }
}
}
```

Activity ini digunakan untuk menambah siswa baru atau memperbarui data siswa yang sudah ada. Logika pengecekan mode (add atau update) dilakukan dengan memeriksa data yang dikirim melalui Intent. Jika ada ID siswa, form akan terisi otomatis dan tombol berubah menjadi "Update", dan jika tidak ada tombol bertulis Add

11. DetailStudent.kt

```
class DetailStudentActivity : AppCompatActivity() {

    private lateinit var binding: ActivityDetailStudentBinding
    private val studentViewModel: StudentViewModel by viewModels {
        StudentViewModelFactory((application as SiswaApps).repository)
    }
    private var studentId: Int = 0

    companion object {
        const val EXTRA_STUDENT_ID = "extra_student_id"
        const val EXTRA_STUDENT_NIS = "extra_student_nis"
        const val EXTRA_STUDENT_NAME = "extra_student_name"
    }
```

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

    studentId = intent.getIntExtra(EXTRA_STUDENT_ID,  defaultValue: 0)

    // Observasi data siswa berdasarkan ID
    observeStudentData()

    binding.btnDelete.setOnClickListener { it: View!
        deleteDialog()
    }

    binding.btnUpdate.setOnClickListener { it: View!
        val intent = Intent( packageContext: this, AddUpdateStudentActivity::class.java).apply { this: Intent
            putExtra(AddUpdateStudentActivity.EXTRA_STUDENT_ID, studentId)
            putExtra(AddUpdateStudentActivity.EXTRA_STUDENT_NIS, binding.tvNis.text.toString())
            putExtra(AddUpdateStudentActivity.EXTRA_STUDENT_NAME, binding.tvName.text.toString())
        }
        startActivity(intent)
    }
}
```

```
private fun observeStudentData() {
    studentViewModel.getStudentById(studentId).observe( owner: this) { student ->
        if (student != null) {
            binding.tvNis.text = student.nis
            binding.tvName.text = student.fullName
        }
    }
}

private fun deleteDialog() {
    AlertDialog.Builder( context: this)
        .setTitle("Hapus Data Siswa")
        .setMessage("Yakin mau hapus data siswa ini?")
        .setPositiveButton( text: "Ya") { _, _ ->
            deleteStudent()
        }
        .setNegativeButton( text: "Tidak",  listener: null)
        .show()
}
```

```
    private fun deleteStudent() {
        val student = Student(id = studentId, nis = binding.tvNis.text.toString(), fullName = binding.tvName.tex
        studentViewModel.delete(student)
        Toast.makeText( context: this,  text: "Data siswa berhasil dihapus", Toast.LENGTH_SHORT).show()
        finish()
    }
}
```

Activity ini menampilkan informasi detail siswa berdasarkan ID yang diterima melalui Intent. Data diambil dari StudentViewModel untuk ditampilkan ke user.