

LAPORAN PRAKTIK UAS APLIKASI MANAJEMEN TOKO SEMBAKO



DISUSUN OLEH :

Aditya Pratama 2105102001

Efrhillia Windy 2105102011

Isramiraza Ilma Redanda 2105102033

Kelas : MI-5A

Mata Kuliah : Praktik Pem. Mobile Lanjut

**PROGRAM STUDI MANAJEMEN INFORMATIKA
JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA
POLITEKNIK NEGERI MEDAN 2023**

KATA PENGANTAR

Puji syukur bagi Tuhan Yang Maha Esa yang telah memberikan nikmat serta hidayah-Nya sehingga penyusun dapat menyelesaikan Laporan Praktik yang berjudul “APLIKASI MANAJEMEN TOKO SEMBAKO”. Paper ini disusun bertujuan untuk memenuhi salah tugas mata kuliah Praktek Pemrograman Web.

Terima kasih saya ucapan kepada Bapak sebagai Dosen Pengampu Praktik Pemrograman Mobile Lanjut yang telah membantu kami secara materi. Terima kasih juga kami ucapan kepada teman—teman yang telah mendukung kami sehingga bisa menyelesaikan tugas ini tepat waktu. Kami menyadari, bahwa laporan yang kami buat ini masih jauh dari kata sempurna. Oleh karena itu, kami sangat menerima kritik dan saran yang membangun agar bisa menjadi lebih baik lagi dimasa mendatang.

Semoga laporan ini bisa menambah wawasan para pembaca dan bermanfaat untuk perkembangan dan peningkatan ilmu pengetahuan.

Medan, 23 Desember 2023

Penulis

DAFTAR ISI

KATA PENGANTAR	2
DAFTAR ISI	3
BAB I PENDAHULUAN	4
1.1 Latar Belakang Masalah	4
1.2 Rumusan Masalah.....	5
1.3 Tujuan	5
BAB II PEMBAHASAN.....	6
2.1 Proses Pembuatan Aplikasi dan CRUD	6
2.2. Cara Kerja Aplikasi.....	22
BAB III PENUTUP.....	26
3.1 Simpulan	26
3.2 Saran	26

BAB I PENDAHULUAN

1.1 Latar Belakang Masalah

Di era modern ini, penggunaan teknologi informasi telah menjadi elemen utama dalam perkembangan berbagai bisnis. Salah satu contoh bisnis yang telah mengadopsi teknologi untuk mengoptimalkan operasionalnya adalah bisnis toko sembako. Toko sembako memiliki peran yang sangat vital dalam memenuhi kebutuhan sehari-hari masyarakat, terutama dalam menyediakan produk-produk kebutuhan pokok seperti beras, minyak goreng, gula, dan produk sejenisnya.

Dalam menghadapi tantangan ini, pengembangan aplikasi manajemen toko sembako dapat menjadi solusi yang sangat berguna. Aplikasi semacam ini dapat membantu pemilik toko dalam manajemen persediaan, pencatatan penjualan dan keuangan, pelayanan pelanggan, analisis data, serta menjaga keamanan data. Dengan aplikasi ini, pemilik toko sembako dapat mengoptimalkan operasional, meningkatkan efisiensi, dan mengambil keputusan yang lebih baik berdasarkan informasi yang akurat.

Dalam makalah ini, kami akan mendiskusikan lebih lanjut tentang pentingnya aplikasi manajemen toko sembako, potensi manfaat yang dapat diperoleh, serta beberapa fitur kunci yang dapat diimplementasikan dalam aplikasi semacam ini. Tujuan dari makalah ini adalah memberikan pemahaman yang lebih mendalam tentang bagaimana teknologi informasi dapat memberikan dampak positif pada bisnis toko sembako dan memberikan panduan bagi pemilik toko yang ingin mempertimbangkan penggunaan aplikasi manajemen toko sembako dalam operasional mereka. Maka dari itu kami tertarik untuk membuat Tugas UAS Praktik Pemrograman Aplikasi Mobile

Lanjut yang berjudul **“Manajemen Toko Sembako”**

1.2 Rumusan Masalah

Berdasarkan latar belakang yang dikemukakan, rumusan masalah dari tugas akhir ini, yaitu:

1. Bagaimana penggunaan aplikasi manajemen dapat meningkatkan efisiensi operasional dalam toko sembako?
2. Bagaimana aplikasi manajemen dapat membantu dalam pengelolaan stok dan persediaan di toko sembako?
3. Apa saja fitur yang dibutuhkan dalam aplikasi manajemen toko sembako?

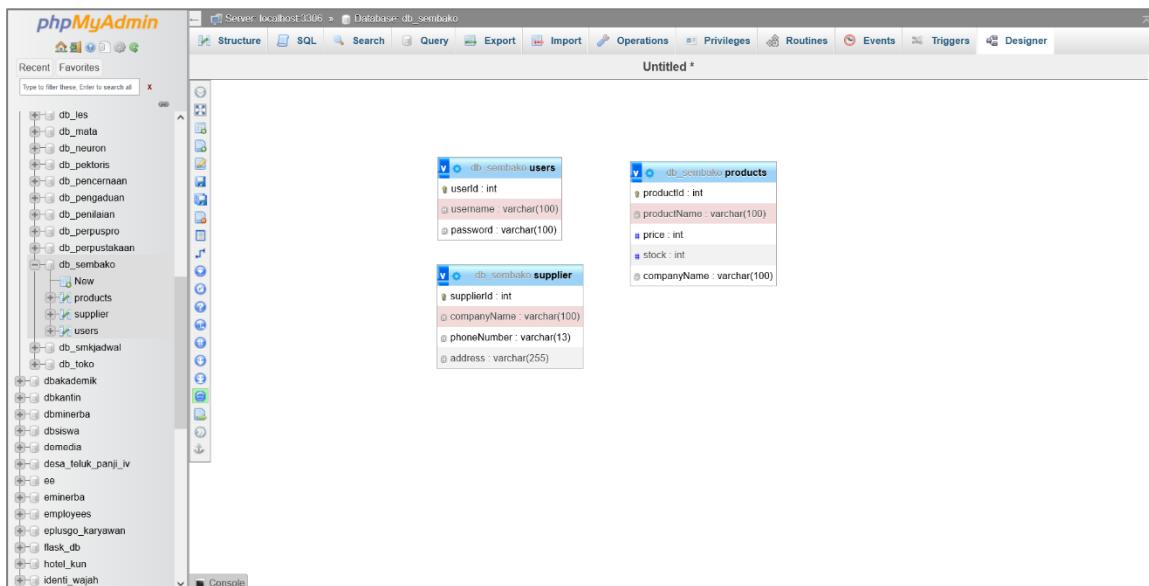
1.3 Tujuan

Berdasarkan Latar Belakang dan Rumusan Masalah dapat diketahui tujuannya. Tujuannya adalah sebagai berikut :

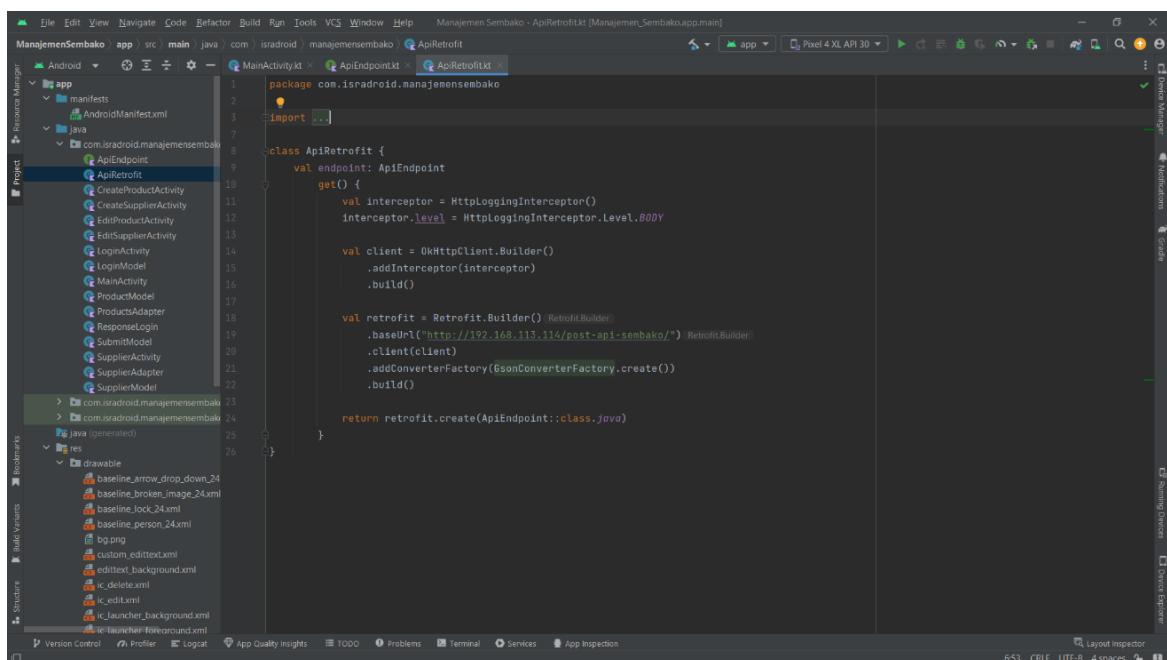
1. Memanfaatkan aplikasi manajemen toko untuk meningkatkan efisiensi dan efektivitas operasional toko sembako.
2. Mengelola stok barang dan persediaan secara akurat untuk mendukung kelancaran bisnis.
3. Mempermudah proses pencatatan penjualan, keuangan, dan pelayanan pelanggan dengan fitur-fitur yang relevan.

BAB II PEMBAHASAN

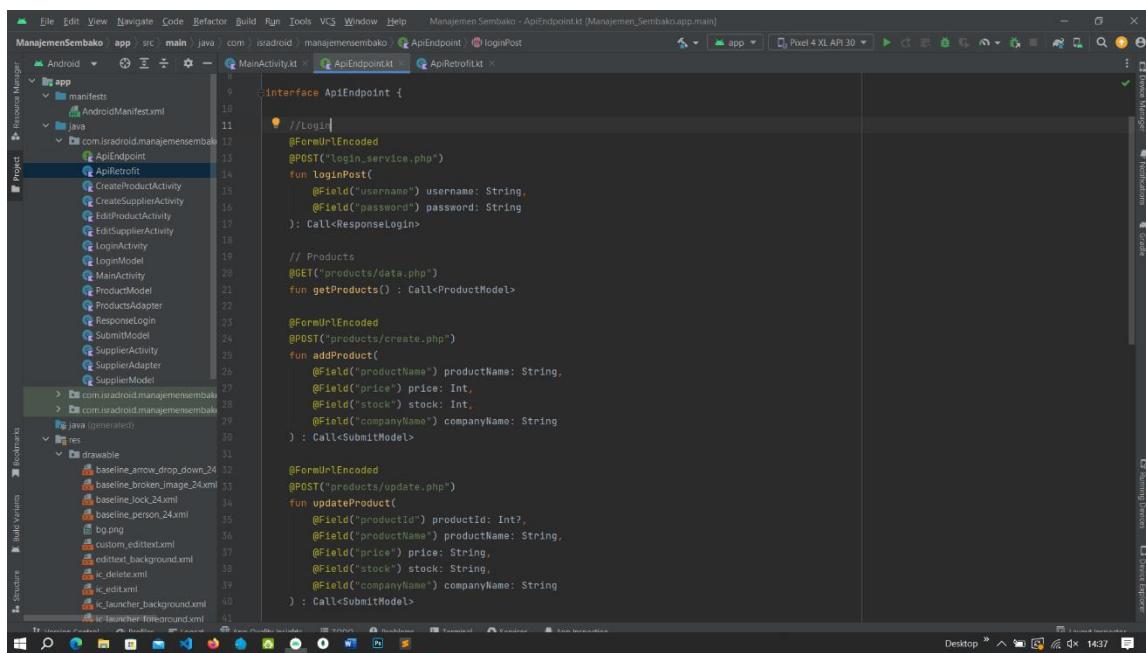
2.1 Proses Pembuatan Aplikasi dan CRUD



Gambar 2.1.1 Design Database db_sembako



Gambar 2.1.2 Konfigurasi API PHP menggunakan Retrofit



The screenshot shows the Android Studio interface with the project 'ManajemenSembako' open. The code editor displays the file 'ApiEndpoint.kt' which contains the following code:

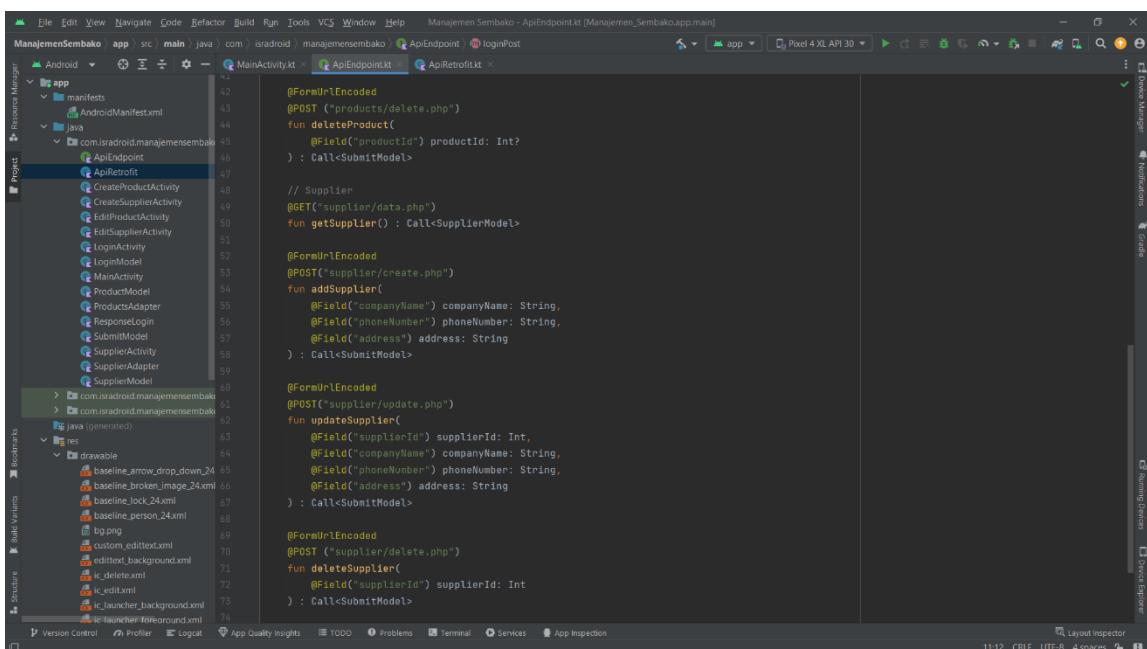
```
interface ApiEndpoint {
    // Login
    @FormUrlEncoded
    @POST("login_service.php")
    fun loginPost(
        @Field("username") username: String,
        @Field("password") password: String
    ): Call<ResponseLogin>

    // Products
    @GET("products/data.php")
    fun getProducts(): Call<ProductModel>

    @FormUrlEncoded
    @POST("products/create.php")
    fun addProduct(
        @Field("productName") productName: String,
        @Field("price") price: Int,
        @Field("stock") stock: Int,
        @Field("companyName") companyName: String
    ): Call<SubmitModel>

    @FormUrlEncoded
    @POST("products/update.php")
    fun updateProduct(
        @Field("productId") productId: Int,
        @Field("productName") productName: String,
        @Field("price") price: String,
        @Field("stock") stock: String,
        @Field("companyName") companyName: String
    ): Call<SubmitModel>
}
```

Gambar 2.1.3 Membuat interface dari setiap endpoint



The screenshot shows the Android Studio interface with the project 'ManajemenSembako' open. The code editor displays the file 'ApiEndpoint.kt' which contains the following code:

```
    @FormUrlEncoded
    @POST("products/delete.php")
    fun deleteProduct(
        @Field("productId") productId: Int
    ): Call<SubmitModel>

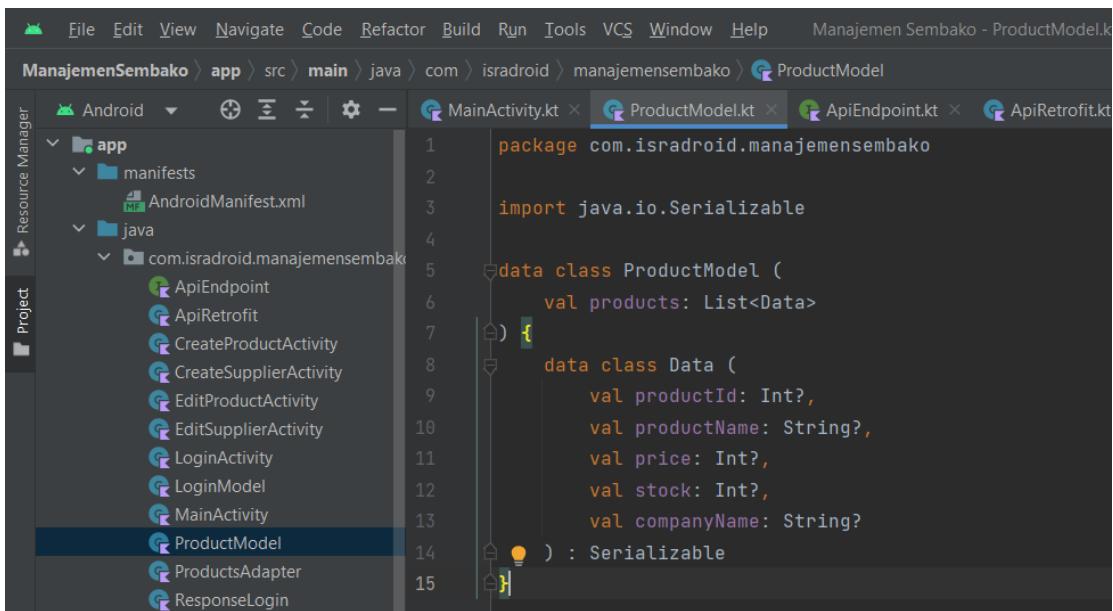
    // Supplier
    @GET("supplier/data.php")
    fun getSupplier(): Call<SupplierModel>

    @FormUrlEncoded
    @POST("supplier/create.php")
    fun addSupplier(
        @Field("companyName") companyName: String,
        @Field("phoneNumber") phoneNumber: String,
        @Field("address") address: String
    ): Call<SubmitModel>

    @FormUrlEncoded
    @POST("supplier/update.php")
    fun updateSupplier(
        @Field("supplierId") supplierId: Int,
        @Field("companyName") companyName: String,
        @Field("phoneNumber") phoneNumber: String,
        @Field("address") address: String
    ): Call<SubmitModel>

    @FormUrlEncoded
    @POST("supplier/delete.php")
    fun deleteSupplier(
        @Field("supplierId") supplierId: Int
    ): Call<SubmitModel>
}
```

Gambar 2.1.4 Membuat interface dari setiap endpoint



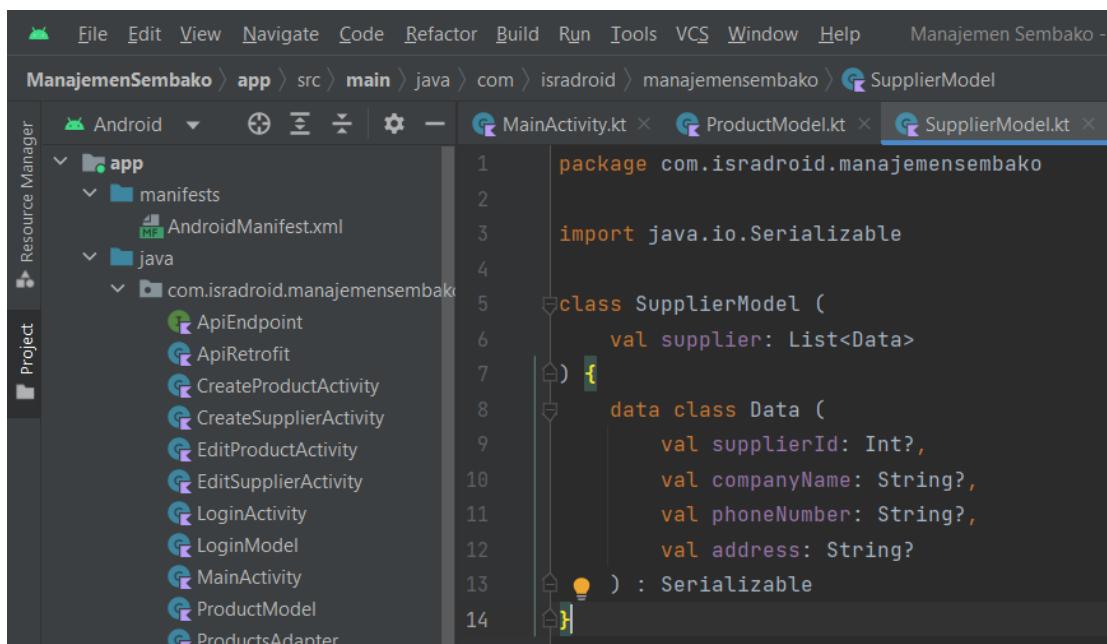
```
package com.isradroid.manajemensemba

import java.io.Serializable

data class ProductModel (
    val products: List<Data>
)

data class Data (
    val productId: Int?,
    val productName: String?,
    val price: Int?,
    val stock: Int?,
    val companyName: String?
) : Serializable
```

Gambar 2.1.4 Class ProductModel



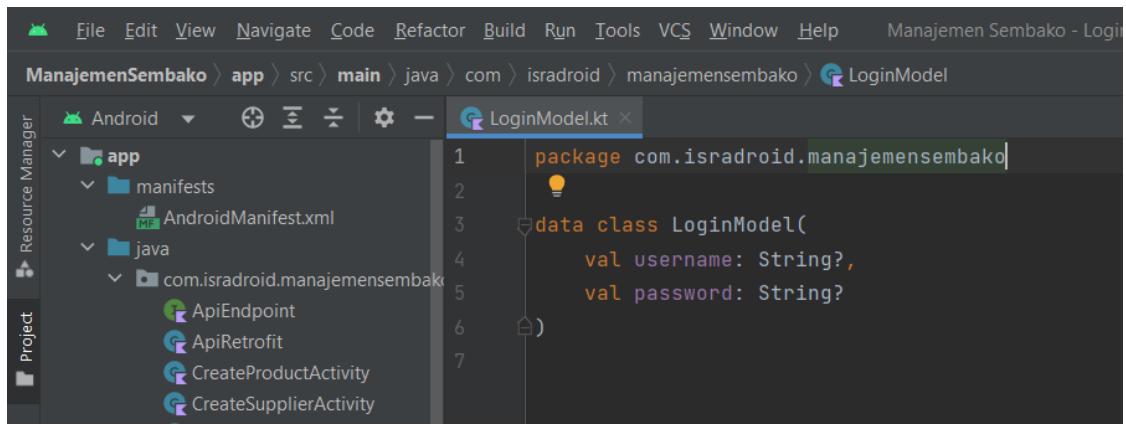
```
package com.isradroid.manajemensemba

import java.io.Serializable

class SupplierModel (
    val supplier: List<Data>
)

data class Data (
    val supplierId: Int?,
    val companyName: String?,
    val phoneNumber: String?,
    val address: String?
) : Serializable
```

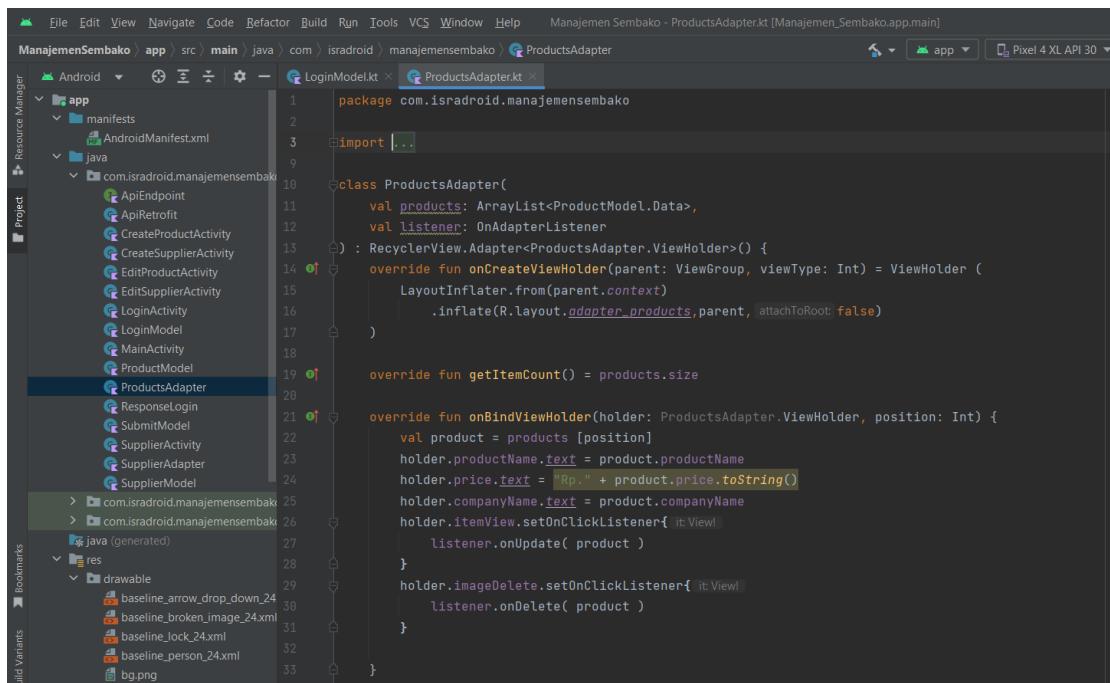
Gambar 2.1.5 Class SupplierModel



The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it, the file path is displayed as `ManajemenSembako > app > src > main > java > com > isradroid > manajemensemba`. The current file is `LoginModel.kt`, which contains the following code:

```
1 package com.isradroid.manajemensemba
2
3 data class LoginModel(
4     val username: String?,
5     val password: String?
6 )
```

Gambar 2.1.6 Class LoginModel



The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it, the file path is displayed as `ManajemenSembako > app > src > main > java > com > isradroid > manajemensemba`. The current file is `ProductsAdapter.kt`, which contains the following code:

```
1 package com.isradroid.manajemensemba
2
3 import ...
4
5 class ProductsAdapter(
6     val products: ArrayList<ProductModel.Data>,
7     val listener: OnAdapterListener
8 ) : RecyclerView.Adapter<ProductsAdapter.ViewHolder>() {
9
10     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int) = ViewHolder(
11         LayoutInflater.from(parent.context)
12             .inflate(R.layout.adapter_products, parent, attachToRoot: false)
13     )
14
15     override fun getItemCount() = products.size
16
17     override fun onBindViewHolder(holder: ProductsAdapter.ViewHolder, position: Int) {
18
19         val product = products [position]
20
21         holder.productName.text = product.productName
22         holder.price.text = "Rp." + product.price.toString()
23         holder.companyName.text = product.companyName
24         holder.itemView.setOnClickListener{ it: View!
25             listener.onUpdate( product )
26         }
27         holder.imageDelete.setOnClickListener{ it: View!
28             listener.onDelete( product )
29         }
30     }
31
32 }
```

Gambar 2.1.7 Class ProductAdapter

The screenshot shows the Android Studio interface with the code editor open to the `ProductsAdapter.kt` file. The code defines a RecyclerView adapter for products. It includes methods for setting up the adapter with data and interfaces for updating and deleting items. The code editor highlights several variables and functions in orange and yellow.

```
ManajemenSembako > app > src > main > java > com > isradroid > manajemensemba > ProductsAdapter.kt [Manajemen_Sembako.app.main]
```

```
27     listener.onUpdate( product )
28 }
29 holder.imageDelete.setOnClickListener{ it:View!
30     listener.onDelete( product )
31 }
32
33 class ViewHolder(view: View): RecyclerView.ViewHolder(view) {
34     val productName = view.findViewById<TextView>(R.id.product_name)
35     val price = view.findViewById<TextView>(R.id.product_price)
36     val companyName = view.findViewById<TextView>(R.id.company_name)
37     val imageDelete = view.findViewById<ImageView>(R.id.image_delete)
38 }
39
40 fun setProducts(data: List<ProductModel.Data>) {
41     products.clear()
42     products.addAll(data)
43     notifyDataSetChanged()
44 }
45
46 interface OnAdapterListener {
47     fun onUpdate(product: ProductModel.Data)
48     fun onDelete(product: ProductModel.Data)
49 }
50 }
```

Gambar 2.1.8 ProductAdapter

The screenshot shows the Android Studio interface with the code editor open to the `SupplierAdapter.kt` file. The code defines a RecyclerView adapter for suppliers. It includes methods for creating view holders and binding data to them. The code editor highlights several variables and functions in orange and yellow.

```
ManajemenSembako > app > src > main > java > com > isradroid > manajemensemba > SupplierAdapter.kt [Manajemen_Sembako.app.main]
```

```
1 package com.isradroid.manajemensemba
2
3 import ...
4
5 class supplierAdapter (
6     val supplier: ArrayList<SupplierModel.Data>,
7     val listener: OnAdapterListener
8 ) : RecyclerView.Adapter<supplierAdapter.ViewHolder>() {
9
10     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int) = ViewHolder(
11         LayoutInflater.from(parent.context)
12             .inflate(R.layout.adapter_supplier, parent, attachToRoot = false)
13     )
14
15     override fun onBindViewHolder(holder: supplierAdapter.ViewHolder, position: Int) {
16         val dataSupplier = supplier[position]
17         holder.companyName.text = dataSupplier.companyName
18         holder.itemView.setOnClickListener{ it:View!
19             listener.onUpdate( dataSupplier )
20         }
21         holder.imageDelete.setOnClickListener{ it:View!
22             listener.onDelete( dataSupplier )
23         }
24     }
25
26     class ViewHolder(itemView: View) {
27         val companyName = itemView.findViewById<TextView>(R.id.company_name)
28         val imageDelete = itemView.findViewById<ImageView>(R.id.image_delete)
29     }
30 }
```

Gambar 2.1.9 SupplierAdapter

The screenshot shows the Android Studio interface with the following details:

- File Path:** ManajemenSembako > app > src > main > java > isradroid > managemensembaiko > SupplierAdapter
- Code Editor:** The code editor displays the `SupplierAdapter.kt` file.
- Project Structure:** The left sidebar shows the project structure under the `app` module, including the `java` directory which contains the `SupplierAdapter` file.
- Code Content:** The code defines a `SupplierAdapter` class extending `RecyclerView.Adapter`. It includes methods for setting data, getting item count, and defining a `ViewHolder`. It also defines an `OnAdapterListener` interface with `onUpdate` and `onDelete` methods.

```
24     listener.onUpdate( dataSupplier )
25 }
26 holder.imageDelete.setOnClickListener{ it: View!
27     listener.onDelete( dataSupplier )
28 }
29 }
30 }
31 }
32 override fun getItemCount() = supplier.size
33
34 class ViewHolder(view: View): RecyclerView.ViewHolder(view) {
35     val companyName = view.findViewById<TextView>(R.id.company_name)
36     val imageDelete = view.findViewById<ImageView>(R.id.image_delete)
37 }
38
39 fun setSupplier(data: List<SupplierModel.Data>) {
40     supplier.clear()
41     supplier.addAll(data)
42     notifyDataSetChanged()
43 }
44
45 interface OnAdapterListener {
46     fun onUpdate(supplier: SupplierModel.Data)
47     fun onDelete(supplier: SupplierModel.Data)
48 }
```

Gambar 2.1.10 SupplierAdapter

The screenshot shows the Android Studio interface with the code editor open to the `LoginActivity.kt` file. The code implements an `AppCompatActivity` and overrides the `onCreate` method to set up the view and listeners. It also defines private fields for `username` and `password`, and a `buttonLogin`.

```
1 package com.isradroid.manajemensabko
2
3 import ...
4
5 class LoginActivity : AppCompatActivity() {
6
7     private lateinit var buttonLogin: MaterialButton
8     private val api by lazy { ApiRetrofit().endpoint }
9     private lateinit var username: EditText
10    private lateinit var password: EditText
11
12    override fun onCreate(savedInstanceState: Bundle?) {
13        super.onCreate(savedInstanceState)
14        setContentView(R.layout.activity_login)
15        setupView()
16        setupListener()
17    }
18
19    private fun setupView() {
20        username = findViewById(R.id.usernameText)
21        password = findViewById(R.id.passwordText)
22        buttonLogin = findViewById(R.id.loginButton)
23    }
24
25    private fun setupListener() {
26        buttonLogin.setOnClickListener { view ->
27            val usernameText = username.text.toString()
28            val passwordText = password.text.toString()
29
30            if(usernameText.isNotEmpty() && passwordText.isNotEmpty()) {
31                api.loginPost(
32                    ...
33                )
34            }
35        }
36    }
37}
```

Gambar 2.1.11 LoginActivity

The screenshot shows the Android Studio interface with the project 'ManajemenSembako' open. The code editor displays the LoginActivity.java file. The code handles user input for login, sends a POST request to the API endpoint, and displays a success or error message using Toast notifications.

```
if(usernameText.isNotEmpty() && passwordText.isNotEmpty()) {
    api.loginPost(
        usernameText,
        passwordText
    ).enqueue(object : Callback<ResponseLogin> {
        override fun onResponse(
            call: Call<ResponseLogin>,
            response: Response<ResponseLogin>
        ) {
            if(response.body() != null) {
                startActivity(Intent(packageContext, MainActivity::class.java))
            } else {
                Toast.makeText(
                    applicationContext,
                    "Username atau Password salah!",
                    Toast.LENGTH_SHORT
                ).show()
            }
        }

        override fun onFailure(call: Call<ResponseLogin>, t: Throwable) {
        }
    })
} else {
    Toast.makeText(
        applicationContext,
        "Username atau Password tidak boleh kosong!",
        Toast.LENGTH_SHORT
    ).show()
}
```

Gambar 2.1.12 LoginActivity

The screenshot shows the Android Studio interface with the project 'ManajemenSembako' open. The code editor displays the MainActivity.java file. The code initializes the application components like Retrofit, RecyclerView, and FloatingActionButton, and sets up the product list view.

```
package com.isradroid.manajemensembaiko

import ...

class MainActivity : AppCompatActivity() {
    private val api by lazy { ApiRetrofit().endpoint }
    private lateinit var productsAdapter: ProductsAdapter
    private lateinit var fabCreate: FloatingActionButton
    private lateinit var toSupplier: MaterialButton
    private lateinit var listProducts: RecyclerView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setupView()
        setupList()
        setupListener()
    }

    override fun onStart() {
        super.onStart()
        getProducts()
    }

    private fun setupView() {
        listProducts = findViewById(R.id.list_product)
        fabCreate = findViewById(R.id.fab_create)
        toSupplier = findViewById(R.id.to_supplier)
    }
}
```

Gambar 2.1.13 MainActivity



The screenshot shows the Android Studio interface with the project 'ManajemenSembako' open. The code editor displays the MainActivity.java file, which contains the implementation for a Main Activity. The code includes imports for Java and the application package, defines a Main Activity class extending AppCompatActivity, and implements onCreate and onStart methods. It also includes setupView and getProducts methods, and initializes various UI components like FloatingActionButton, MaterialButton, and RecyclerView.

```
package com.isandroid.manajemensembako
import ...
class MainActivity : AppCompatActivity() {
    private val api by lazy { ApiRetrofit().endpoint }
    private lateinit var productsAdapter: ProductsAdapter
    private lateinit var fabCreate: FloatingActionButton
    private lateinit var toSupplier: MaterialButton
    private lateinit var listProducts: RecyclerView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setupView()
        setupList()
        setupListener()
    }
    override fun onStart() {
        super.onStart()
        getProducts()
    }
    private fun setupView() {
        listProducts = findViewById(R.id.list_product)
        fabCreate = findViewById(R.id.fab_create)
        toSupplier = findViewById(R.id.to_supplier)
    }
}
```

Gambar 2.1.14 MainActivity

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "ManajemenSembako". The package structure is `com.isradroid.manajemensembako`, containing `MainActivity`, `ProductsAdapter`, `SupplierAdapter`, and `LoginModel`.
- MainActivity.kt:** The code implements `OnItemClickListener` and defines methods for listing products, deleting products, and handling responses.
- Code Snippet:**

```
private fun setupList() {
    productsAdapter = ProductsAdapter(arrayListOf(), object: ProductsAdapter.OnAdapterListener {
        override fun onUpdate(product: ProductModel.Data) {
            startActivity(Intent(packageContext, EditProductActivity::class.java)
                .putExtra("name", "productData"), product)
        }

        override fun onDelete(product: ProductModel.Data) {
            api.deleteProduct(product.productId!!).enqueue(object: Callback<SubmitModel> {
                override fun onResponse(call: Call<SubmitModel>, response: Response<SubmitModel>) {
                    if (response.isSuccessful) {
                        val submit = response.body()
                        Toast.makeText(applicationContext, submit!!.message, Toast.LENGTH_SHORT)
                            .show()
                        getProducts()
                    }
                }

                override fun onFailure(call: Call<SubmitModel>, t: Throwable) { }
            })
        }
    })
}
```
- Build Variants:** The build variant is set to "Debug".
- Layout Inspector:** The Layout Inspector tab is visible on the right side of the interface.

Gambar 2.1.15 MainActivity

The screenshot shows the Android Studio interface with the project 'ManajemenSembako' open. The code editor displays the MainActivity.kt file. The code implements a FloatingActionButton's click listener to start a CreateProductActivity, and a SupplierAdapter's click listener to start a SupplierActivity. It also defines a getProducts() method to fetch product data from an API and handle onFailure callbacks.

```
private fun setupListener() {
    fabCreate.setOnClickListener { it: View? ->
        startActivity(Intent(packageContext, CreateProductActivity::class.java))
    }
    toSupplier.setOnClickListener { it: View? ->
        startActivity(Intent(packageContext, SupplierActivity::class.java))
    }
}

private fun getProducts() {
    //Get Products
    api.getProducts().enqueue(object : Callback<ProductModel> {
        override fun onResponse(call: Call<ProductModel>, response: Response<ProductModel>) {
            if(response.isSuccessful) {
                val listProducts = response.body()!!.products
                productsAdapter.setProducts(listProducts)
            }
        }
        override fun onFailure(call: Call<ProductModel>, t: Throwable) {
            Log.e(tag, "MainActivity", t.toString())
        }
    })
}
```

Gambar 2.1.16 MainActivity

The screenshot shows the Android Studio interface with the project 'ManajemenSembako' open. The code editor displays the SupplierActivity.kt file. This activity extends AppCompatActivity and uses a RecyclerView to display a list of suppliers. It overrides onCreate and onStart methods, and implements setupView(), setupList(), and setupListener() methods to manage the supplier list and adapter.

```
package com.isradroid.manajemensembako
import ...

class SupplierActivity : AppCompatActivity() {

    private val api by lazy { ApiRetrofit().endpoint }
    private lateinit var fabCreate: FloatingActionButton
    private lateinit var supplierAdapter: SupplierAdapter
    private lateinit var listSupplier: RecyclerView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_supplier)
        setupView()
        setupList()
        setupListener()
    }

    override fun onStart() {
        super.onStart()
        getSupplier()
    }

    private fun setupView() {
        listSupplier = findViewById(R.id.list_supplier)
        fabCreate = findViewById(R.id.fab_create)
    }

    private fun setupList() {
        supplierAdapter = SupplierAdapter(arrayListOf(), object : SupplierAdapter.OnAdapterListener {
            override fun onUpdate(supplier: SupplierModel.Data) {

```

Gambar 2.1.17 SupplierActivity

The screenshot shows the Android Studio interface with the code editor open to `CreateProductActivity.kt`. The code implements a `SupplierAdapter` and handles product creation logic. The code editor has syntax highlighting and code completion suggestions.

```
private lateinit var editProductName: EditText
private lateinit var editPrice: EditText
private lateinit var editTextStock: EditText
private lateinit var buttonCreate: MaterialButton
private val api by lazy { ApiRetrofit().endpoint }
private lateinit var supplierAdapter: SupplierAdapter
private lateinit var supplierDropdown: Spinner

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_create)
    // supportActionBar!!.title = "Tambah Produk"
    setupView()
    supplierAdapter = SupplierAdapter(ArrayList(), object : SupplierAdapter.OnAdapterListener {
        override fun onUpdate(supplier: SupplierModel.Data) {
            // Handle update action
        }

        override fun onDelete(supplier: SupplierModel.Data) {
            // Handle delete action
        }
    })
}

getSupplier()
setupListener()
```

Gambar 2.1.18 CreatProductActivity

The screenshot shows the Android Studio interface with the code editor open to `SupplierActivity.kt`. The code implements a `SupplierAdapter` and handles supplier management logic. It includes methods for listing suppliers, updating supplier details, and deleting suppliers. The code editor shows various annotations and method signatures.

```
private fun setupList() {
    supplierAdapter = SupplierAdapter(arrayListOf(), object : SupplierAdapter.OnAdapterListener {
        override fun onUpdate(supplier: SupplierModel.Data) {
            startActivity(
                Intent(packageContext, EditSupplierActivity::class.java)
                    .putExtra("name", "supplierData", supplier)
            )
        }

        override fun onDelete(supplier: SupplierModel.Data) {
            api.deleteSupplier(supplier.supplierId!!).enqueue(object : Callback<SubmitModel> {
                override fun onResponse(
                    call: Call<SubmitModel>,
                    response: Response<SubmitModel>
                ) {
                    if (response.isSuccessful) {
                        val submit = response.body()
                        Toast.makeText(
                            applicationContext,
                            submit!!.message,
                            Toast.LENGTH_SHORT
                        ).show()
                        getSupplier()
                    }
                }

                override fun onFailure(call: Call<SubmitModel>, t: Throwable) {}
            })
        }
    })
}
```

Gambar 2.1.19 SupplierActivity

The screenshot shows the Android Studio interface with the code editor open to the `SupplierActivity.java` file. The code is part of the `ManajemenSembako` application. It includes imports for `com.isradroid.manajemensembako`, `com.google.gson.Gson`, `com.google.gson.reflect.TypeToken`, `java.util.List`, and `java.util.ArrayList`. The code defines a `SupplierAdapter` and handles supplier list setup and retrieval.

```
    listSupplier.adapter = supplierAdapter

    private fun setupListener() {
        fabCreate.setOnClickListener { it: View ->
            startActivity(Intent(context, CreateSupplierActivity::class.java))
        }
    }

    private fun getSupplier() {
        //Get Supplier
        api.getSupplier().enqueue(object : Callback<SupplierModel>, Response<SupplierModel> {
            override fun onResponse(call: Call<SupplierModel>, response: Response<SupplierModel>) {
                if(response.isSuccessful) {
                    val listSupplier = response.body()!!.supplier
                    supplierAdapter.setSupplier(listSupplier)
                }
            }

            override fun onFailure(call: Call<SupplierModel>, t: Throwable) {
                Log.e("MainActivity", t.toString())
            }
        })
    }
```

Gambar 2.1.20 SupplierActivity

The screenshot shows the Android Studio interface with the code editor open to the `CreateProductActivity.java` file. The code is part of the `Manajemen Sembako` application. It includes imports for `com.isradroid.manajemensembako`, `com.google.gson.Gson`, `com.google.gson.reflect.TypeToken`, `java.util.List`, and `java.util.ArrayList`. The code handles product creation logic, including enqueueing requests and displaying success or error messages via Toast.

```
    priceText.toInt(),
    stockText.toInt(),
    supplierDropdownText
)

.enqueue(object : Callback<SubmitModel> {
    override fun onResponse(
        call: Call<SubmitModel>,
        response: Response<SubmitModel>
    ) {
        if (response.isSuccessful) {
            val submit = response.body()
            Toast.makeText(
                applicationContext,
                submit.message,
                Toast.LENGTH_SHORT
            ).show()
            finish()
        }
    }

    override fun onFailure(call: Call<SubmitModel>, t: Throwable) {
    }
}

else{
    Toast.makeText(
        applicationContext,
        "Harap isi semua data!",
        Toast.LENGTH_SHORT
    ).show()
}
```

Gambar 2.1.21 CreatPruductActivity

The screenshot shows the Android Studio interface with the code editor open to the `CreateProductActivity.kt` file. The code is written in Kotlin and defines two main functions: `setupView()` and `setupListener()`. The `setupView()` function initializes various UI components like edit texts and a dropdown. The `setupListener()` function sets up a click listener for a button, retrieves values from the UI, and sends a POST request to the API using Retrofit. The code uses nullable properties and safe calls.

```
private fun setupView() {
    editProductName = findViewById(R.id.product_name)
    editPrice = findViewById(R.id.price)
    editStock = findViewById(R.id.stock)
    buttonCreate = findViewById(R.id.button_create)
    supplierDropdown = findViewById(R.id.list_supplier)
}

private fun setupListener() {
    buttonCreate.setOnClickListener { view ->
        val productNameText = editProductName.text.toString()
        val priceText = editPrice.text.toString()
        val stockText = editStock.text.toString()
        val supplierDropdownText = supplierDropdown.selectedItem as String

        if (productNameText.isNotEmpty() && priceText.isNotEmpty() && stockText.isNotEmpty()) {
            api.addProduct(
                productNameText,
                priceText.toInt(),
                stockText.toInt(),
                supplierDropdownText
            ).enqueue(object : Callback<SubmitModel> {
                override fun onResponse(call: Call<SubmitModel>, response: Response<SubmitModel>) {
                    if (response.isSuccessful) {
                        val submit = response.body()
                        Toast.makeText(
                            applicationContext,
                            "Success"
                        )
                    }
                }

                override fun onFailure(call: Call<SubmitModel>, t: Throwable) {
                    Log.e("MainActivity", t.toString())
                }
            })
        }
    }
}
```

Gambar 2.1.22 CreateProductActivity

This screenshot shows the same `CreateProductActivity.kt` file as the previous one, but with more of the code visible. It includes the implementation of the `getSupplier()` method, which uses Retrofit to fetch a list of suppliers and then populates a spinner with their names. The code handles success and failure cases for the API call.

```
private fun getSupplier() {
    // Get Supplier
    api.getSupplier().enqueue(object : Callback<SupplierModel> {
        override fun onResponse(call: Call<SupplierModel>, response: Response<SupplierModel>) {
            if (response.isSuccessful) {
                val listSupplier = response.body()?.supplier
                val companyNames = listSupplier?.map { it.companyName } // Extract company names
                supplierAdapter.setSupplier(listSupplier!!)
                val adapter = ArrayAdapter(
                    context: this@CreateProductActivity,
                    android.R.layout.simple_spinner_item,
                    objects: companyNames ?: emptyList()
                )
                adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
                supplierDropdown.adapter = adapter
            }
        }

        override fun onFailure(call: Call<SupplierModel>, t: Throwable) {
            Log.e("MainActivity", t.toString())
        }
    })
}
```

Gambar 2.1.23 CreateProductActivity

```
package com.isradroid.manajemensembako

import ...

class EditProductActivity : AppCompatActivity() {

    private lateinit var editProductName: EditText
    private lateinit var editPrice: EditText
    private lateinit var editStock: EditText
    private lateinit var editSupplierDropdown: Spinner
    private lateinit var buttonUpdate: MaterialButton
    private lateinit var supplierAdapter: SupplierAdapter
    private val api by lazy { ApiRetrofit().endpoint }

    private val getProduct by lazy { Intent.getStringExtra("productData") as ProductModel.Data }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_edit_product)
        setupView()
        getSupplier()
        supplierAdapter = SupplierAdapter(ArrayList(), object : SupplierAdapter.OnAdapterListener {
            override fun onUpdate(supplier: SupplierModel.Data) {
                // Handle update action
            }

            override fun onDelete(supplier: SupplierModel.Data) {
                // Handle delete action
            }
        })
        setupListener()
    }

    private fun setupView() {
        editProductName = findViewById(R.id.product_name)
        editPrice = findViewById(R.id.price)
        editStock = findViewById(R.id.stock)
        editSupplierDropdown = findViewById(R.id.list_supplier)
        buttonUpdate = findViewById(R.id.button_update)

        editProductName.setText(getProduct.productName)
        editPrice.setText(getProduct.price.toString())
        editStock.setText(getProduct.stock.toString())
    }

    private fun setupListener() {
        buttonUpdate.setOnClickListener {
            val productNameText = editProductName.text.toString()
            val priceText = editPrice.text.toString()
            val stockText = editStock.text.toString()
            val companyNameText = editSupplierDropdown.selectedItem as String

            if(productNameText.isNotEmpty() && priceText.isNotEmpty() && stockText.isNotEmpty() && companyNameText.isNotEmpty()){
                api.updateProduct(
                    getProductId,
                    productNameText,
                    priceText,
                    stockText,
                    companyNameText
                ).enqueue(object : Callback<SubmitModel> {
                    override fun onResponse(call: Call<SubmitModel>, response: Response<SubmitModel>)

```

Gambar 2.1.24 EditProductActivity

```
                    ...
                )
            }
        }
    }
}
```

Gambar 2.1.25 EditProductActivity

The screenshot shows the Android Studio interface with the code editor open to the `EditProductActivity.kt` file. The code is written in Kotlin and handles the logic for updating a product. It includes methods for getting a supplier, setting up listeners, and displaying toast messages for success or failure. The code uses the RxJava library for asynchronous operations.

```
    .enqueue(object : Callback<SubmitModel> {
        override fun onResponse(
            call: Call<SubmitModel>,
            response: Response<SubmitModel>
        ) {
            if (response.isSuccessful) {
                val submit = response.body()
                Toast.makeText(
                    applicationContext,
                    submit!!.message,
                    Toast.LENGTH_SHORT
                ).show()
                finish()
            }
        }

        override fun onFailure(call: Call<SubmitModel>, t: Throwable) { }
    })
} else{
    Toast.makeText(
        applicationContext,
        "Harap isi semua data!",
        Toast.LENGTH_SHORT
    ).show()
}
}

private fun getSupplier() {
    // Get Supplier
    api.getSupplier().enqueue(object : Callback<SupplierModel> {
        override fun onResponse(call: Call<SupplierModel>, response: Response<SupplierModel>) {
            if (response.isSuccessful) {
                val listSupplier = response.body()?.supplier
                val companyNames = listSupplier?.map { it.companyName } // Extract company names
                supplierAdapter.setSupplier(listSupplier!!)
                val adapter = ArrayAdapter(
                    context: this@EditProductActivity,
                    android.R.layout.simple_spinner_item,
                    objects: companyNames ?: emptyList()
                )
                adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
                editSupplierDropdown.adapter = adapter
            }
        }

        override fun onFailure(call: Call<SupplierModel>, t: Throwable) {
            Log.e(tag: "MainActivity", t.toString())
        }
    })
}
```

Gambar 2.1.25 EditProductActivity

This screenshot shows the same `EditProductActivity.kt` file as the previous one, but with more of the code visible. It includes the implementation of the `getSupplier` method, which uses an `ArrayAdapter` to populate a spinner with company names. The code also includes error handling for network failures.

```
    .enqueue(object : Callback<SubmitModel> {
        override fun onResponse(
            call: Call<SubmitModel>,
            response: Response<SubmitModel>
        ) {
            if (response.isSuccessful) {
                val submit = response.body()
                Toast.makeText(
                    applicationContext,
                    submit!!.message,
                    Toast.LENGTH_SHORT
                ).show()
                finish()
            }
        }

        override fun onFailure(call: Call<SubmitModel>, t: Throwable) { }
    })
} else{
    Toast.makeText(
        applicationContext,
        "Harap isi semua data!",
        Toast.LENGTH_SHORT
    ).show()
}
}

private fun getSupplier() {
    // Get Supplier
    api.getSupplier().enqueue(object : Callback<SupplierModel> {
        override fun onResponse(call: Call<SupplierModel>, response: Response<SupplierModel>) {
            if (response.isSuccessful) {
                val listSupplier = response.body()?.supplier
                val companyNames = listSupplier?.map { it.companyName } // Extract company names
                supplierAdapter.setSupplier(listSupplier!!)
                val adapter = ArrayAdapter(
                    context: this@EditProductActivity,
                    android.R.layout.simple_spinner_item,
                    objects: companyNames ?: emptyList()
                )
                adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
                editSupplierDropdown.adapter = adapter
            }
        }

        override fun onFailure(call: Call<SupplierModel>, t: Throwable) {
            Log.e(tag: "MainActivity", t.toString())
        }
    })
}
```

Gambar 2.1.26 EditProductActivity

The screenshot shows the Android Studio interface with the code editor open to the `EditSupplierActivity.kt` file. The code is written in Kotlin and handles the creation and update of supplier data. It uses a Retrofit API endpoint to interact with the server. The code includes methods for setting up the view, handling user input, and updating the supplier record.

```
private lateinit var editCompanyName: EditText
private lateinit var editPhoneNumber: EditText
private lateinit var editAddress: EditText
private lateinit var buttonUpdate: MaterialButton
private val api by lazy { ApiRetrofit().endpoint }
private val getSupplier by lazy { Intent().getSerializableExtra("supplierData") as SupplierModel.Data }

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_edit_supplier)
    setupView()
    setupListener()
}

private fun setupView() {
    editCompanyName = findViewById(R.id.company_name)
    editPhoneNumber = findViewById(R.id.phone_number)
    editAddress = findViewById(R.id.address)
    buttonUpdate = findViewById(R.id.button_update)

    editCompanyName.setText( getSupplier.companyName )
    editPhoneNumber.setText( getSupplier.phoneNumber )
    editAddress.setText( getSupplier.address )
}

private fun setupListener() {
    buttonUpdate.setOnClickListener { it: View ->
        val companyName = editCompanyName.text.toString()
        val phoneNumber = editPhoneNumber.text.toString()
        val address = editAddress.text.toString()

        if(companyName.isNotEmpty() && phoneNumber.isNotEmpty() && address.isNotEmpty()) {
            api.updateSupplier(
                getSupplier.supplierId!!,
                companyName,
                phoneNumber,
                address
            ).enqueue(object : Callback<SubmitModel> {
                override fun onResponse(
                    call: Call<SubmitModel>,
                    response: Response<SubmitModel>
                ) {
                    if (response.isSuccessful) {
                        val submit = response.body()
                        Toast.makeText(
                            applicationContext,
                            submit.message,
                            Toast.LENGTH_SHORT
                        ).show()
                        finish()
                    }
                }

                override fun onFailure(call: Call<SubmitModel>, t: Throwable) {}
            })
        } else{
            Toast.makeText(
                applicationContext,
                "Harap isi semua data",
                Toast.LENGTH_SHORT
            ).show()
        }
    }
}
```

Gambar 2.1.27 EditSupplierActivity

The screenshot shows the Android Studio interface with the code editor open to the `EditSupplierActivity.kt` file. The code is identical to the one in Gambar 2.1.27, handling the creation and update of supplier data using a Retrofit API endpoint. The code includes methods for setting up the view, handling user input, and updating the supplier record.

```
private lateinit var editCompanyName: EditText
private lateinit var editPhoneNumber: EditText
private lateinit var editAddress: EditText
private lateinit var buttonUpdate: MaterialButton
private val api by lazy { ApiRetrofit().endpoint }
private val getSupplier by lazy { Intent().getSerializableExtra("supplierData") as SupplierModel.Data }

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_edit_supplier)
    setupView()
    setupListener()
}

private fun setupView() {
    editCompanyName = findViewById(R.id.company_name)
    editPhoneNumber = findViewById(R.id.phone_number)
    editAddress = findViewById(R.id.address)
    buttonUpdate = findViewById(R.id.button_update)

    editCompanyName.setText( getSupplier.companyName )
    editPhoneNumber.setText( getSupplier.phoneNumber )
    editAddress.setText( getSupplier.address )
}

private fun setupListener() {
    buttonUpdate.setOnClickListener { it: View ->
        val companyName = editCompanyName.text.toString()
        val phoneNumber = editPhoneNumber.text.toString()
        val address = editAddress.text.toString()

        if(companyName.isNotEmpty() && phoneNumber.isNotEmpty() && address.isNotEmpty()) {
            api.updateSupplier(
                getSupplier.supplierId!!,
                companyName,
                phoneNumber,
                address
            ).enqueue(object : Callback<SubmitModel> {
                override fun onResponse(
                    call: Call<SubmitModel>,
                    response: Response<SubmitModel>
                ) {
                    if (response.isSuccessful) {
                        val submit = response.body()
                        Toast.makeText(
                            applicationContext,
                            submit.message,
                            Toast.LENGTH_SHORT
                        ).show()
                        finish()
                    }
                }

                override fun onFailure(call: Call<SubmitModel>, t: Throwable) {}
            })
        } else{
            Toast.makeText(
                applicationContext,
                "Harap isi semua data",
                Toast.LENGTH_SHORT
            ).show()
        }
    }
}
```

Gambar 2.1.28 EditSupplierActivity

The screenshot shows the Android Studio interface with the code editor open. The file is `ResponseLogin.kt` located in the `com.isradroid.manajemensemba` package. The code defines a data class `ResponseLogin` with two properties: `response` (Boolean) and `payload` (LoginModel). The code editor has syntax highlighting and a code completion feature.

```
package com.isradroid.manajemensemba
data class ResponseLogin(
    var response: Boolean,
    var payload: LoginModel)
```

Gambar 2.1.29 Class ResponseLogin

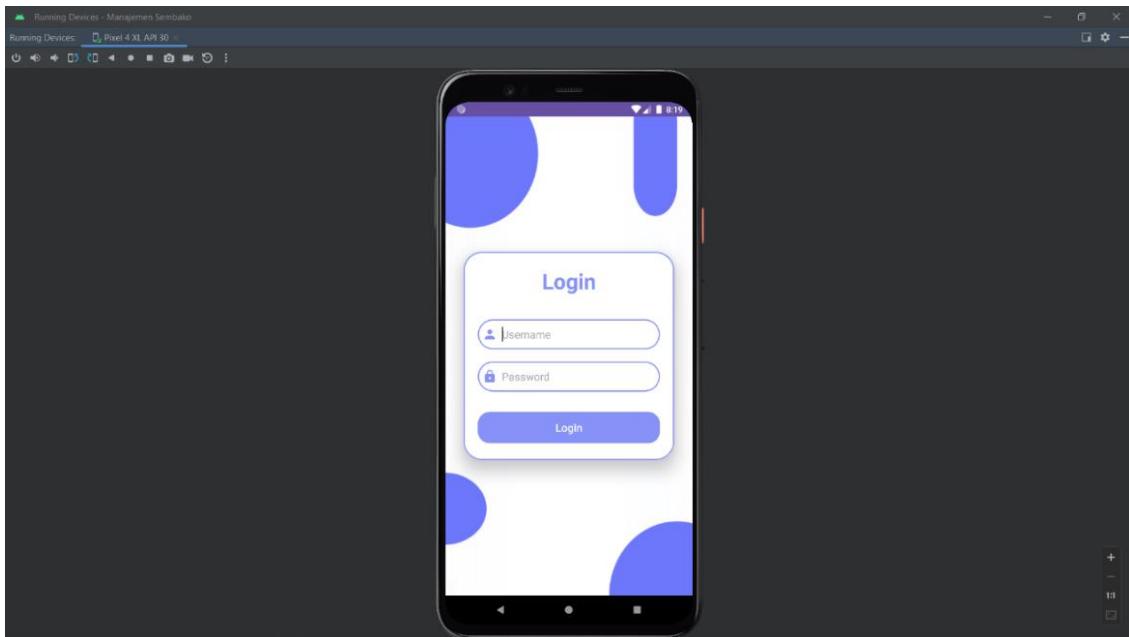
The screenshot shows the Android Studio interface with the code editor open. The file is `SubmitModel.kt` located in the `com.isradroid.manajemensemba` package. The code defines a data class `SubmitModel` with one property: `message` (String). The code editor has syntax highlighting and a code completion feature.

```
package com.isradroid.manajemensemba
data class SubmitModel(
    val message: String)
```

Gambar 2.1.30 Class SubmitModel

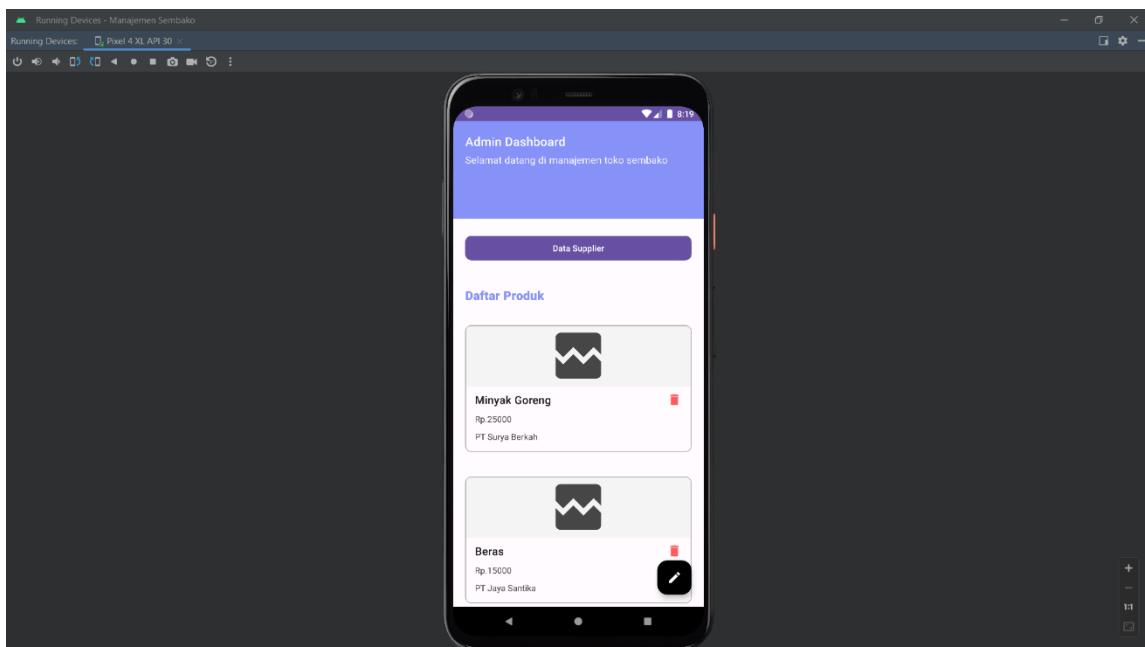
2.2. Cara Kerja Aplikasi

Pada activity ini user diminta untuk memasukan username dan password yang terdaftar di database



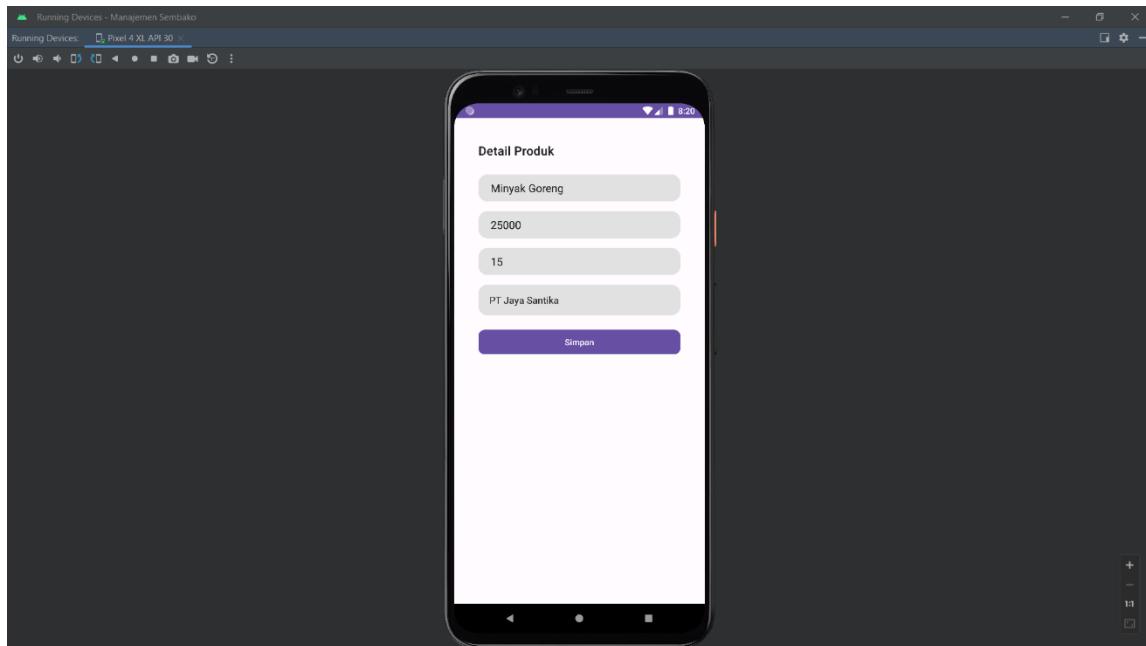
Gambar 2.2.1 Tampilan Login

Kemudian pada activity ini kita dapat menampilkan daftar produk dan terdapat button untuk beralih ke activity supplier serta floating button untuk beralih ke activity tambah produk



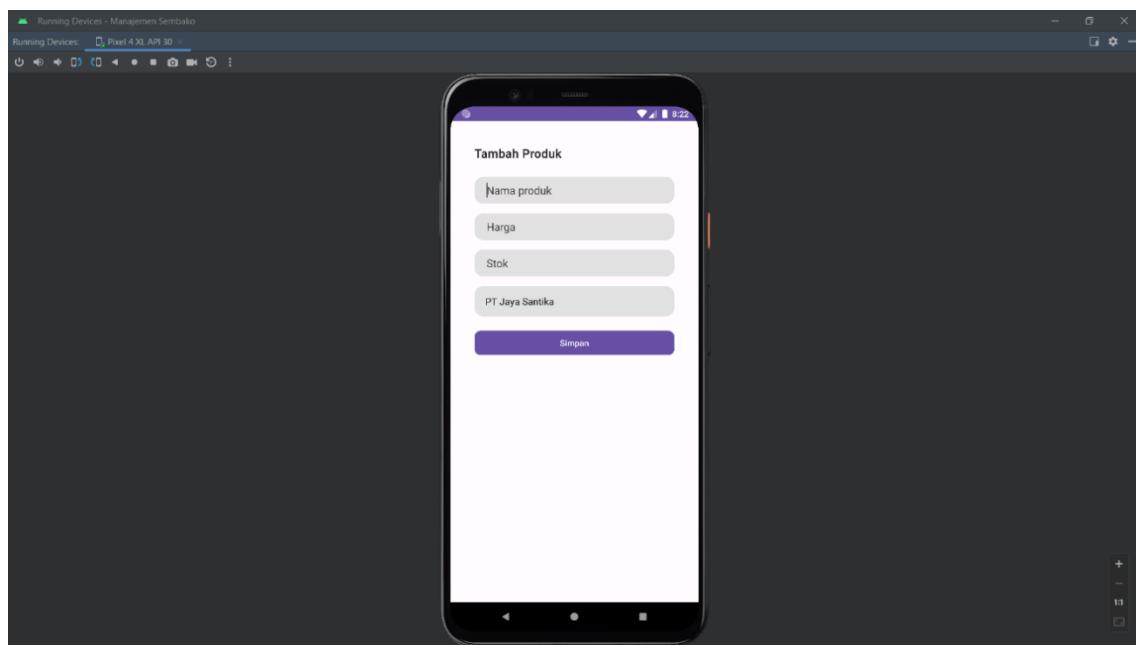
Gambar 2.2.2 Tampilan Dashboard

Di activity ini dapat menampilkan detail produk sekaligus dapat meng-update data dari produk tersebut



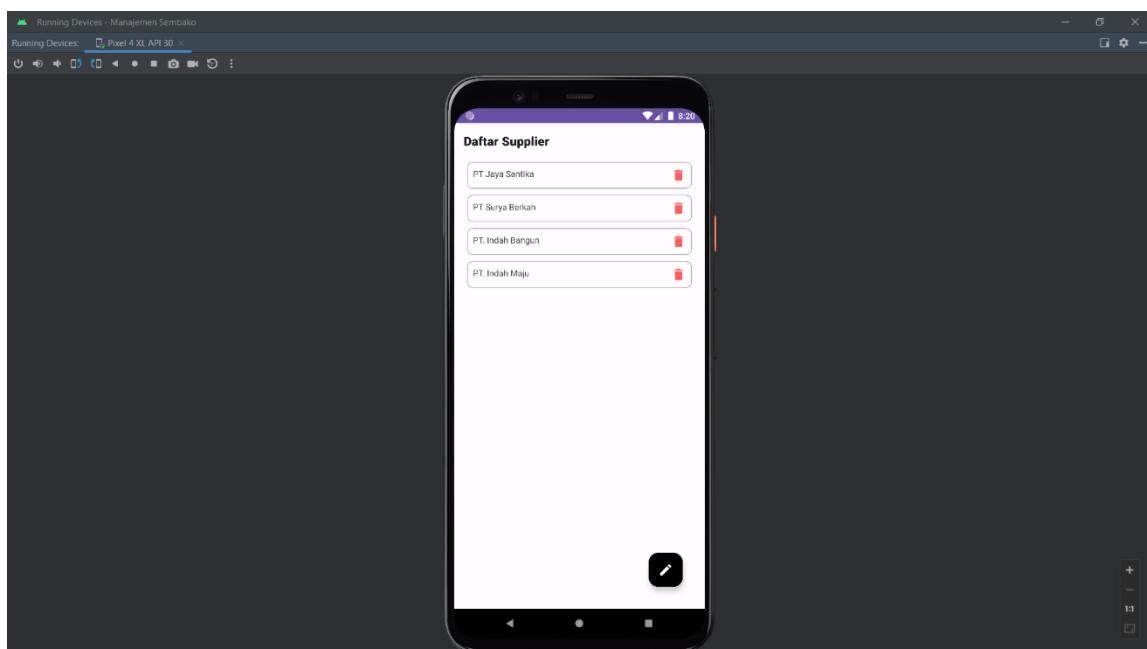
Gambar 2.2.3 Tampilan DetailProduk

Kemudian pada activity ini dapat menambahkan produk baru



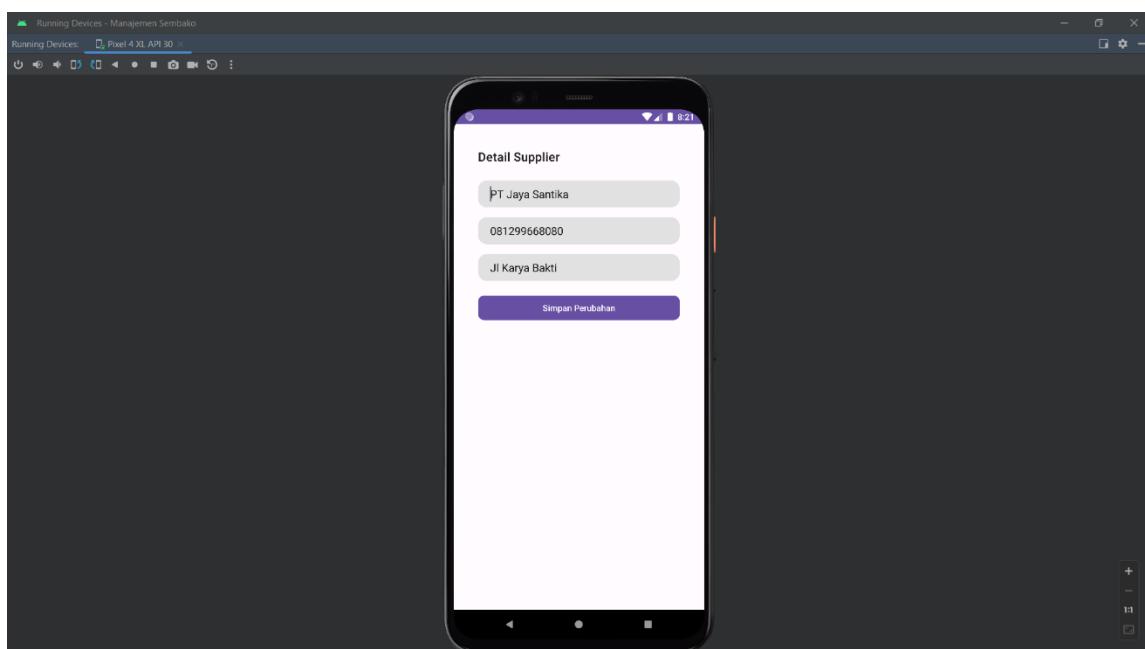
Gambar 2.1.4 Tampilan TambahProduk

Pada activity ini kita dapat menampilkan daftar supplier dan terdapat floating button untuk beralih ke activity tambah supplier



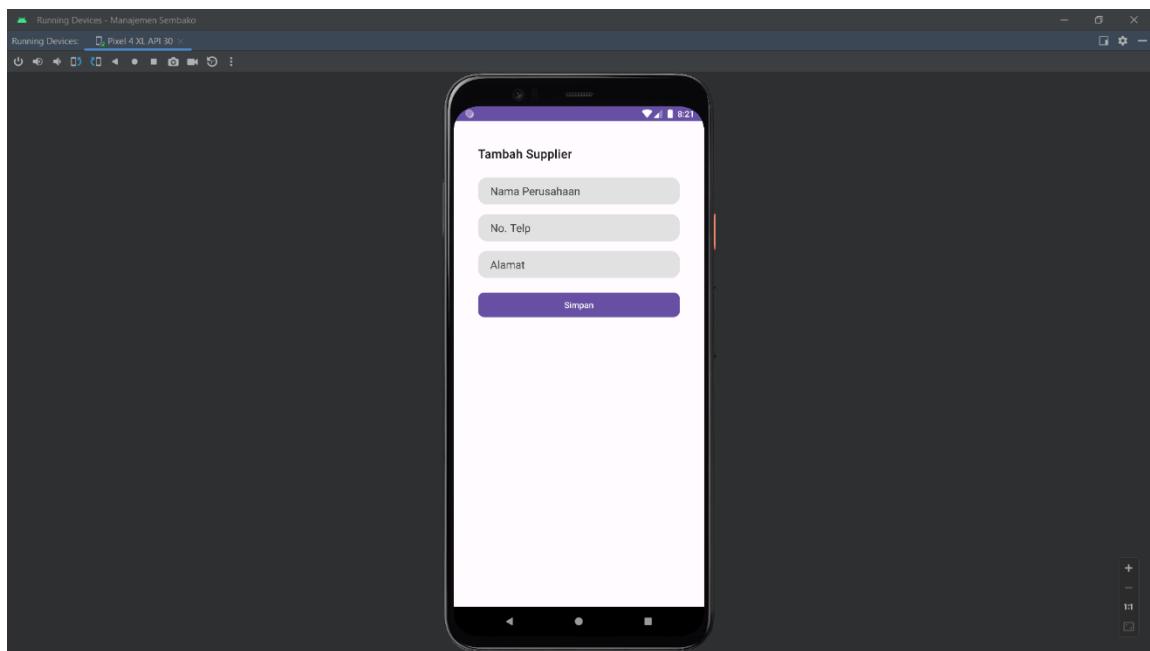
Gambar 2.2.5 Tampilan DataSupplier

Di activity ini dapat menampilkan detail supplier sekaligus dapat meng-update data dari supplier tersebut



Gambar 2.2.7 Tampilan DetailSupplier

Kemudian pada activity ini dapat menambahkan supplier baru .



Gambar 2.2.6 Tampilan TambahSupplier

BAB III PENUTUP

3.1 Simpulan

Berdasarkan pembahasan bab-bab sebelumnya maka pada bab ini penulis dapat mengambil simpulan sebagai berikut:

1. Dengan menggunakan aplikasi ini, pemilik toko dapat meningkatkan efisiensi operasional dengan mengotomatiskan beberapa proses, seperti pencatatan penjualan, manajemen persediaan, dan analisis data.
2. Aplikasi manajemen toko sembako merupakan solusi yang sangat berguna dalam mengoptimalkan operasional toko sembako dan meningkatkan daya saing bisnis.
3. Aplikasi ini juga membantu dalam pengelolaan stok dan persediaan dengan memberikan informasi yang akurat dan real-time.

3.2 Saran

Berdasarkan pembahasan bab-bab sebelumnya maka pada bab ini penulis dapat mengambil simpulan sebagai berikut:

1. Dalam mengembangkan aplikasi manajemen toko sembako, penting untuk mempertimbangkan kebutuhan operasional yang spesifik dari toko sembako. Identifikasi proses bisnis yang perlu diotomatisasi dan fungsionalitas yang diperlukan untuk meningkatkan efisiensi operasional.
2. Pastikan aplikasi manajemen toko sembako memiliki fitur yang memadai untuk mengelola stok dan persediaan. Hal ini mencakup kemampuan untuk mencatat masuk dan keluarnya produk, pemantauan level persediaan, pengingat untuk mengisi ulang persediaan yang menipis, dan kemampuan untuk menghasilkan laporan stok secara berkala.
3. Jaga keamanan data dengan menggunakan sistem keamanan yang handal dalam aplikasi manajemen toko sembako. Ini termasuk proteksi data pelanggan, enkripsi data sensitif, dan pengaturan izin akses yang tepat untuk pengguna aplikasi.
4. Selalu lakukan pembaruan dan perbaikan rutin pada aplikasi untuk memastikan pemeliharaan yang baik dan peningkatan fungsionalitas.