SnackOrdering

AYESHA J

NOORUL RIGANA BARVEEN A

IRFANA BARVEEN A

JAFFRIN JAMEMA E A

1.Introduction

1.1 Overview

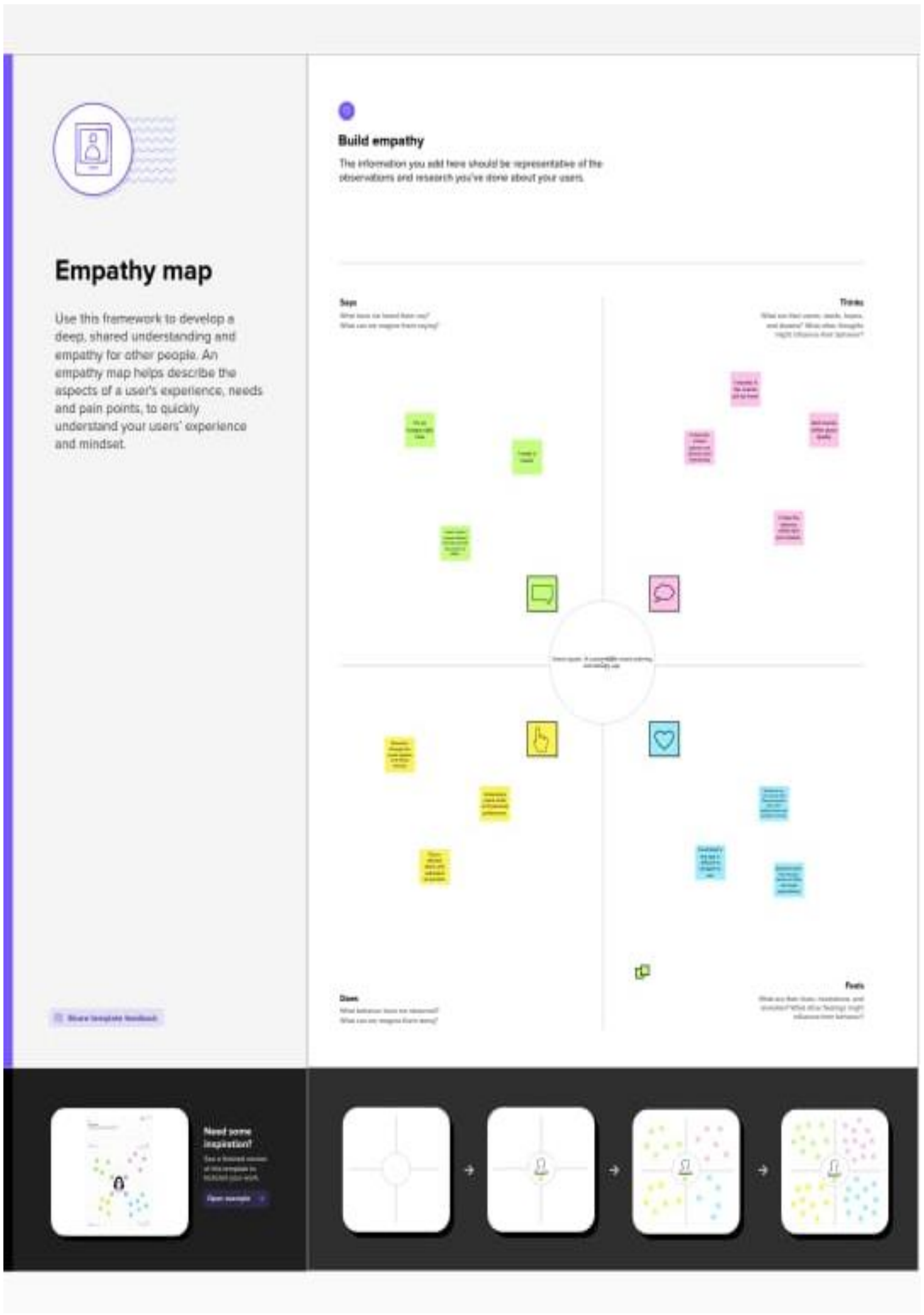
A project that demonstrates the use of android jetpack Compose to build a UI for a snack squad is a simple project build using the android compose UI toolkit .
It demonstrates how to create a simple e-commerce app for snacks using the compose libraries. The user can see a list of snacks, and by tapping on a snack, and by tapping on a snack, and, by tapping on the "Add to Card" button, the snack will be added, the snack will be added the card The user can also see the list of items in the cart and can proceed to checkout to make the purchase.

1.2 Purpose

The purpose of a snack ordering app is to provide a convenient and efficient way for users to order snacks from a particular vendor or restaurant. The app may allow users to browse a menu, customize their order, and make payments all within the app. The app may also provide features such as tracking the status of the order and providing estimated delivery times. The main goal of a snack ordering app is to provide a streamlined ordering process for users and make it easier for them to order their favorite snacks from their favorite vendors.

2.Problem definition & design thinking

2.1 Empathy Map



Empathy map

Use this framework to develop a deep, shared understanding and empathy for other people. An empathy map helps describe the aspects of a user's experience, needs and pain points, to quickly understand your users' experience and mindset.

Build empathy

The information you add here should be representative of the observations and research you've done about your users.

Says

What have we heard them say?
What can we imagine them saying?

Thinks

What are their wants, needs, hopes, and dreams? What other thoughts might influence their behavior?

Does

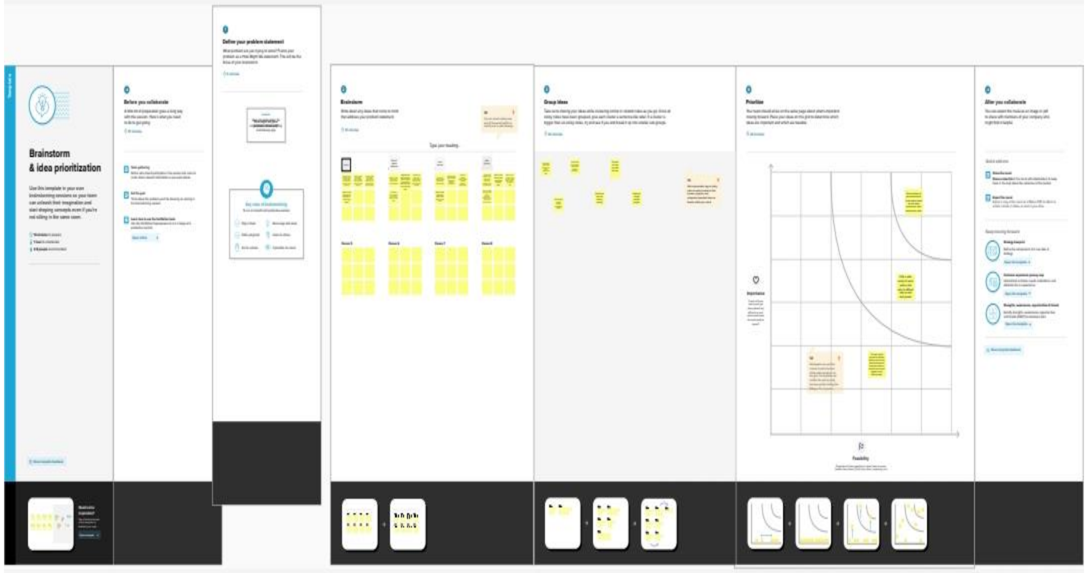
What behavior have we observed?
What can we imagine them doing?

Feels

What are their fears, frustrations, and anxieties? What other feelings might influence their behavior?

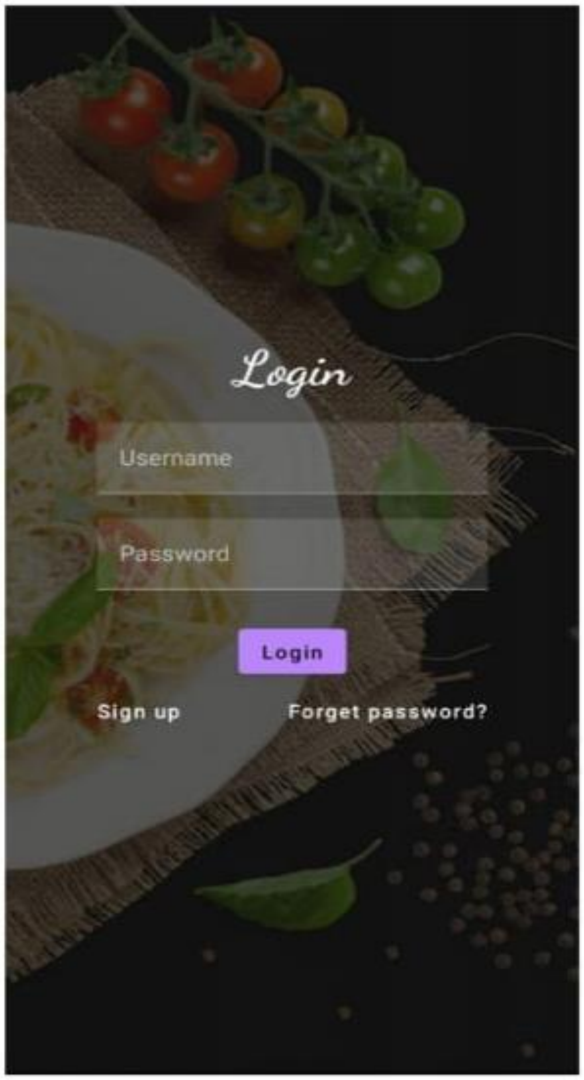
Share template feedback
Need some inspiration?
See a finished version of this template to kickstart your work.
Open example

2.2 Ideation & Brainstorming Map

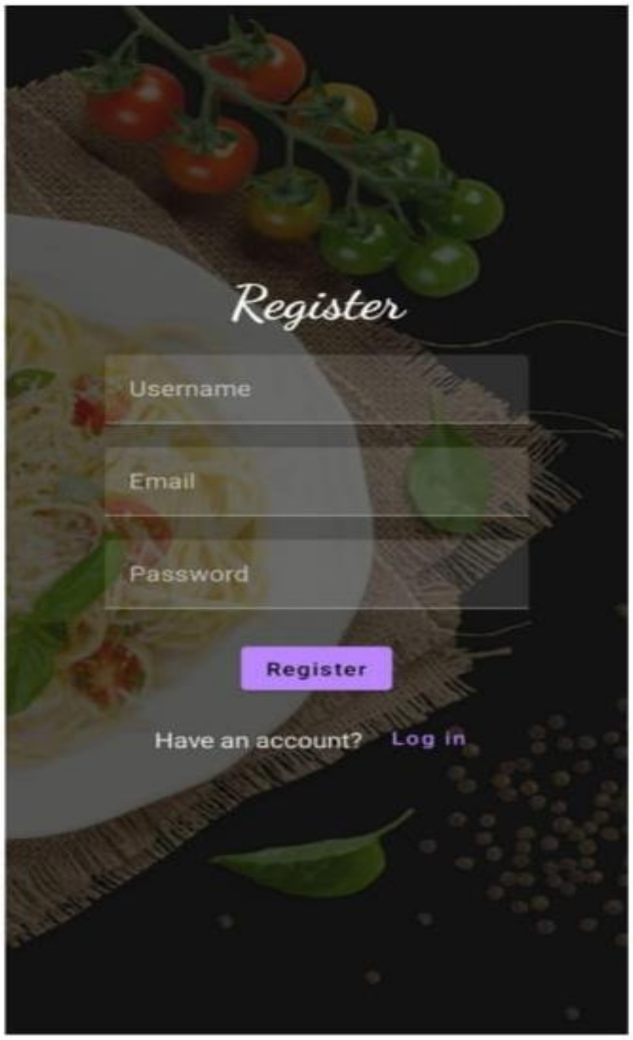

Brainstorm & idea prioritization
Before you collaborate
Define your problem statement
Brainstorm
Group ideas
Prioritize
After you collaborate
Importance
Feasibility

3.Result

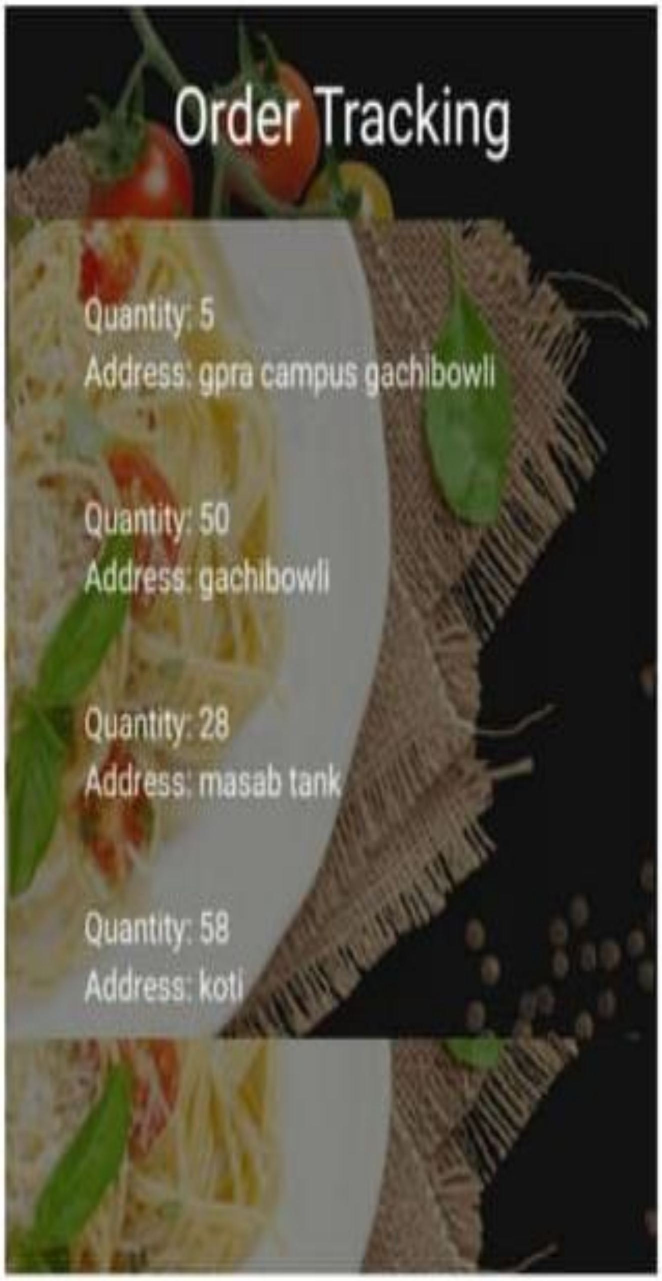
Login Page :



Register Page :



Admin page:



Order Tracking

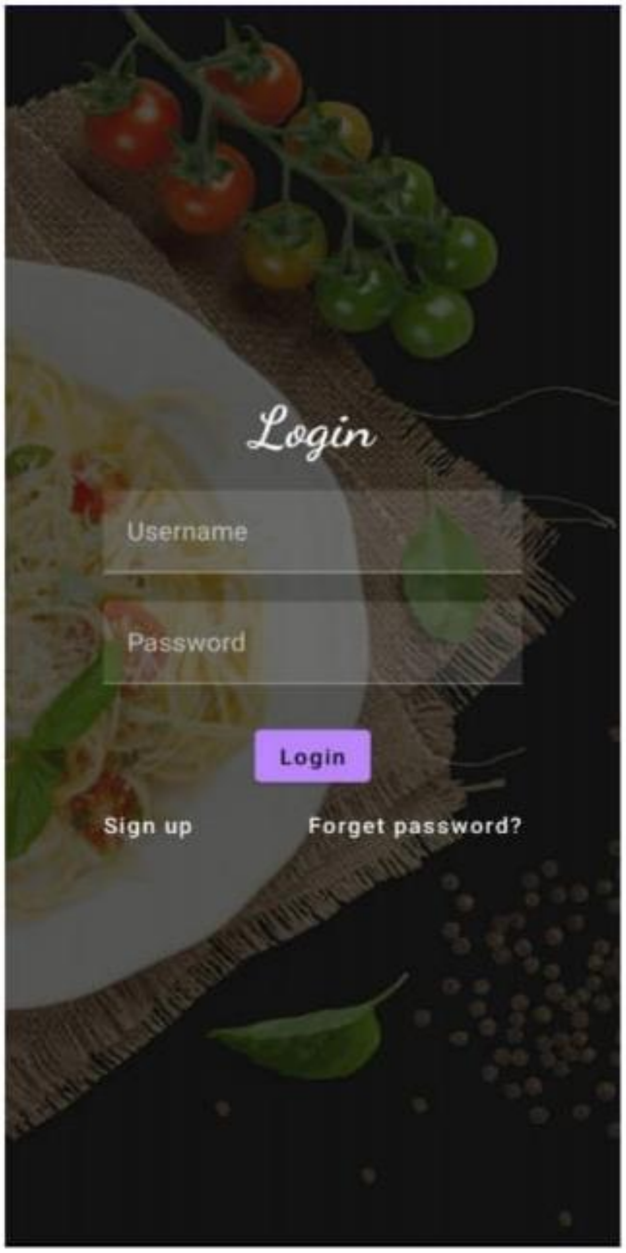
Quantity: 5
Address: gpra campus gachibowli

Quantity: 50
Address: gachibowli

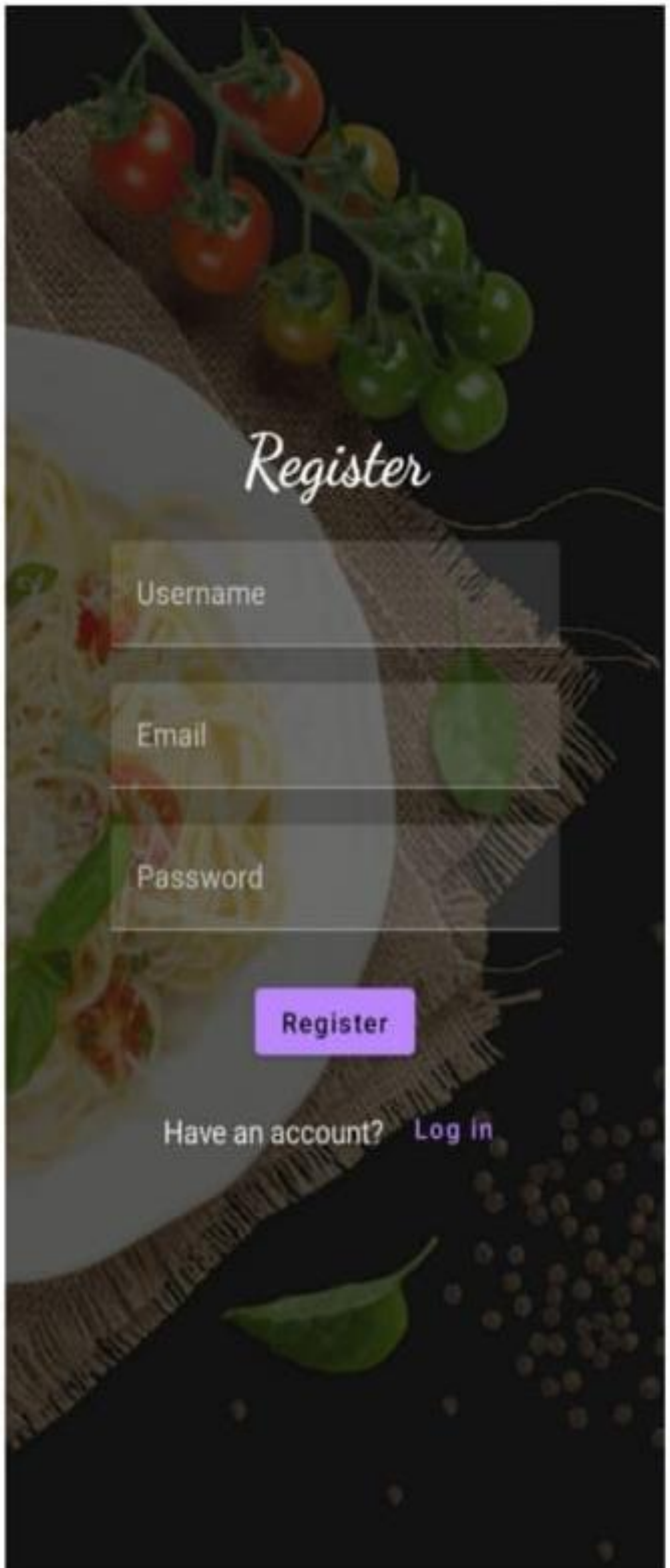
Quantity: 28
Address: masab tank

Quantity: 58
Address: koti

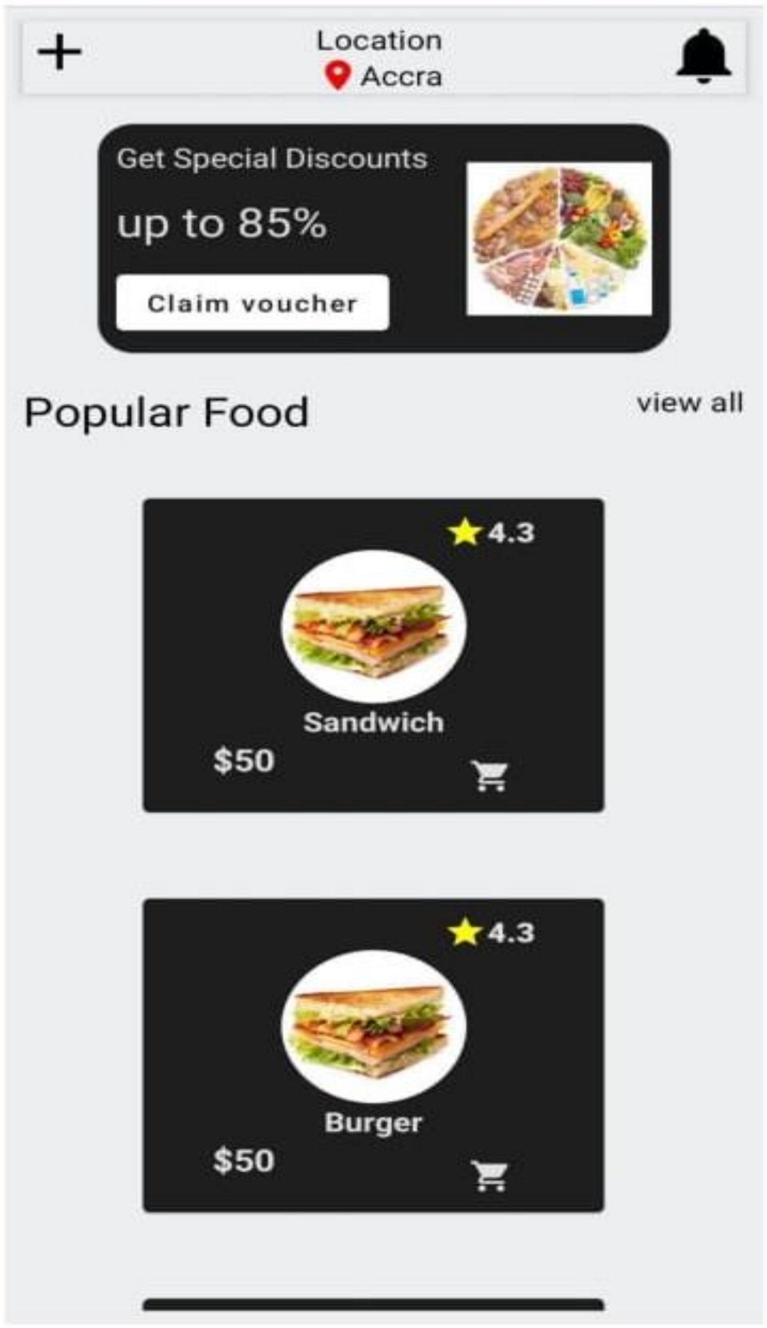
User Module:
Login Page :



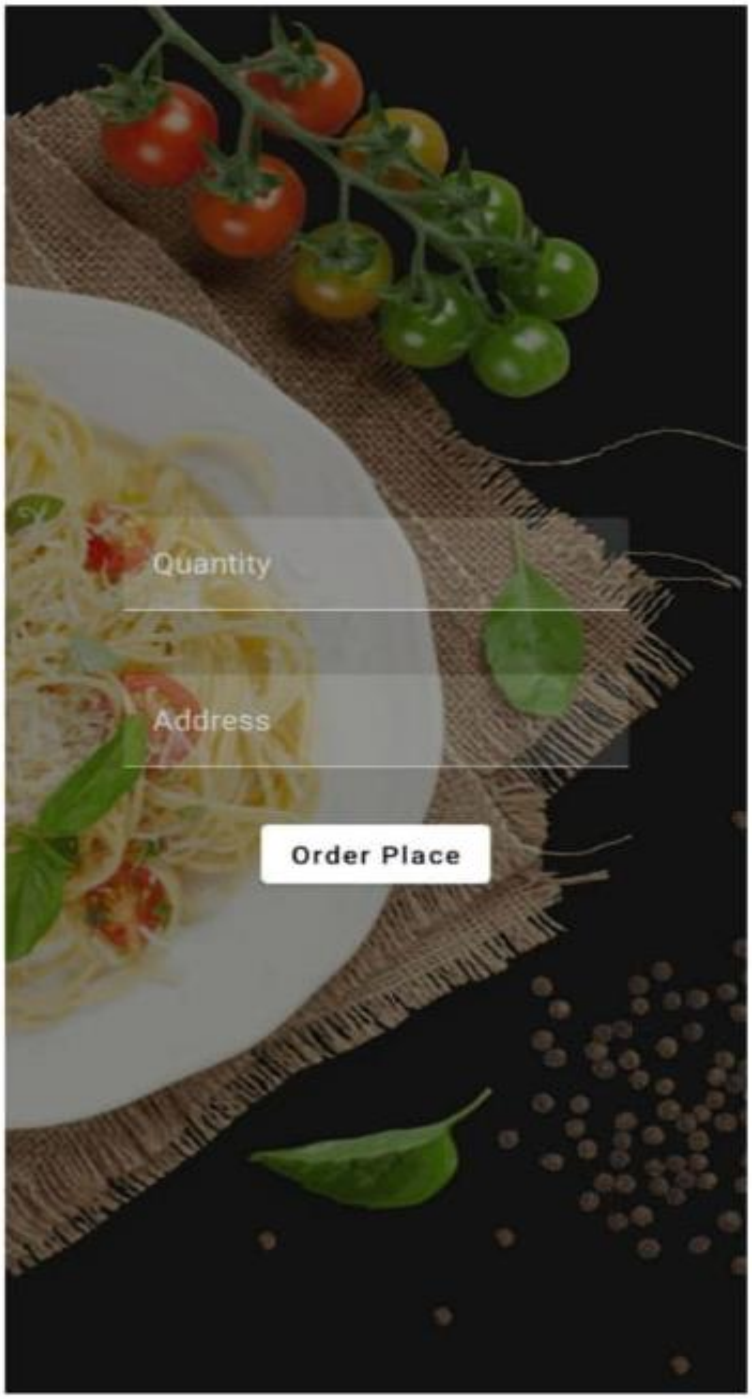
Register Page :



Main Page :


+
Location
Accra
Get Special Discounts
up to 85%
Claim voucher
Popular Food
view all
4.3
Sandwich
$50
4.3
Burger
$50

Order Page :



4.Advantages & Disadvantages

Advantages of a snack order app:

- Convenience: Users can order snacks from anywhere, anytime using the app, which saves time and effort

- Increased Efficiency: The app eliminates the need for phone calls, reducing the chances of miscommunication and errors

- Personalization: Users can customize their orders and preferences, which can increase customer satisfaction.

- Easy Payment: The app may offer multiple payment options, making it easy for users to pay for their orders.

- Improved Tracking: Users can track their orders in real-time, which can improve trust and satisfaction.

Disadvantages of a snack order app:

- Technical Issues: The app may experience technical glitches, which can affect the user experience and cause frustration.

- Limited Options: The app may only offer limited options from specific vendors, which may not cater to all users.

- Lack of Human Interaction: The app eliminates the human touch that customers may appreciate when ordering from their favorite vendors.

- Dependence on Internet: The app requires a stable internet connection, which may not always be available.

- Privacy and Security Concerns: Users need to provide personal information and payment details to use the app, which may pose privacy and security risks.

5.Application

The snack order app can be used in a variety of settings where people need a quick and convenient way to order snacks. Some of the potential applications of a snack order app include:

- Food Delivery Services: Snack order apps can be used by food delivery services to allow customers to order snacks from local restaurants or vendors.

- Corporate Cafeterias: Large companies can use snack order apps to streamline their cafeteria operationsand allow employees to order snacks from their desks.

- Schools and Universities: Snack order apps can be used by schools and universities to allow students to order snacks from on-campus vendors.

- Event Catering: Snack order apps can be used by event planners to allow attendees to order snacks during conferences, trade shows, and other events.

- Sports Venues: Sports venues can use snack order apps to allow fans to order snacks from their seats during games and events.Overall, the snack order app can be used in any setting where people want to order snacks quickly and easily.

About Android App

➢ Android Studio is the official integrated development environment (IDE) for building Android apps. It was first released by Google in 2013 and has since become the most popular development environment for Android app developers.

- Android Studio is based on the IntelliJ IDEA community edition, and it includes many tools and features designed specifically for developing Android apps. Some of the key features of Android Studio include:

User Interface (UI) Designer:

Android Studio includes a powerful UI designer that allows developers to easily create and modify app layouts using drag-and-drop tools. The UI designer supports a variety of layouts, including linear, relative, and constraint layouts.

Code Editor:

Android Studio includes a powerful code editor that provides syntax highlighting, code completion,
and other features to help developers write clean, efficient code. The code editor also supports debugging
and refactoring tools.

Emulator:

Android Studio includes a built-in emulator that allows developers to test their apps on different An
droid devices without needing to own the actual devices. The emulator supports a wide range of Android
versions and device configurations.

Gradle Build System:

Android Studio uses the Gradle build system, which makes it easy to manage dependencies and build complex apps with multiple modules.

Version Control:

Android Studio supports version control systems like Git, allowing developers to easily manage their code changes and collaborate with other team members.

Performance Profiling:

- Android Studio includes performance profiling tools that allow developers to identify performance bottlenecks in their apps and optimize their code for betterperformance.

- Overall, Android Studio is a powerful tool for developing high-quality Android apps. It provides a range of features and tools that make it easy for developers to build, test, and deploy their apps.

6. Conclusion

In conclusion, a snack order app is a useful tool that can simplify the process of ordering snacks from local vendors or restaurants. With a snack order app, users can browse menus, customize their orders, and make payments all within the app. The app can be used in various settings, including food delivery services, corporate cafeterias, schools and universities, event catering, and sports venues. While there are advantages to using a snack order app, such as increased convenience and efficiency, there are also some potential drawbacks, such as technical issues, limited options, and privacy and security concerns. Overall, a snack order app can be a valuable addition to any setting where people want to order snacks quickly and easily

7.Future Scope

Snack order apps have gained immense popularity in recent years due to the convenience they offer to customers who want to order food online. With the increasing demand for such apps, the future scope of snack order apps looks promising.

Here are some potential areas of growth and development for snack order apps in the future:

Enhanced personalization:

Snack order apps can use machine learning and artificial intelligence algorithms to track customer preferences and offer personalized recommendations for food items based on their previous orders, search history, and other relevant data. This will make the ordering process more efficient and enjoyable for customers.

Integration with voice assistants:

The integration of snack order apps with voice assistants such as Amazon’s Alexa, Google Assistant, or Apple’s Siri can provide a hands-free experience for customers. Customers can place their orders through voice commands, making the process much faster and more accessible.

Integration with augmented reality:

Snack order apps can integrate augmented reality (AR) technology, allowing customers to visualize their food orders in real-time before placing their orders. This will give customers a better idea of what to expect and help reduce the chances of ordering incorrect items.

Integration with drones:

Delivery of food items using drones can revolutionize the snack ordering industry,
providing faster delivery times and reducing costs. Snack order apps can incorporate drone technology for delivery in the future.

Expansion of geographical coverage:

- The expansion of geographical coverage of snack order apps to more cities and countries will increase the customer base, creating more revenue opportunities for the companies.

- In conclusion, snack order apps have a bright future ahead, with the potential to integrate emerging technologies and expand their reach. By doing so, they can provide a more personalized and convenient experience for customers, leading to increased customer loyalty and revenue growth.

8.Appendix

Source Code

Database 1:

Step 1: Create User Data class

```
Package
com.example.snackorder
ing
```

```
import
androidx.room.ColumnI
nfo
import
androidx.room.Entity
import
androidx.room.Primary
Key
```

```
@Entity(tableName =
"user_table")
data class User(

@PrimaryKey(autoGener
ate = true) val id:
Int?,
    @ColumnInfo(name
= "first_name") val
firstName: String?,
    @ColumnInfo(name
= "last_name") val
lastName: String?,
    @ColumnInfo(name
= "email") val email:
String?,
    @ColumnInfo(name
= "password") val
password: String?,

    )
```

Step 2: Create UserDao interface class

```
package
com.example.snackorde
ring

import androidx.room.*

@Dao
interface UserDao {

    @Query("SELECT *
FROM user_table WHERE
email = :email")
    suspend fun
getUserByEmail(email:
String): User?

    @Insert(onConflict
=
OnConflictStrategy.REP
LACE)
    suspend fun
insertUser(user: User)

    @Update
    suspend fun
updateUser(user: User)

    @Delete
    suspend fun
deleteUser(user: User)
}
```

Step 3: Create an UserDatabase class

```
package
com.example.snackor
dering
```

```
import
android.content.Cont
ext
import
androidx.room.Databa
se
import
androidx.room.Room
import
androidx.room.RoomDa
tabase

@Database(entities =
[User::class],
version = 1)
```

```
abstract class
UserDatabase :
RoomDatabase() {

    abstract fun
userDao(): UserDao

    companion object
{

        @Volatile
        private var
instance:
UserDatabase? = null

        fun
getDatabase(context:
Context):
UserDatabase {
            return
instance ?:
synchronized(this) {
                val
newInstance =
Room.databaseBuilder
(
                                context.applicationC
                                ontext,

                                UserDatabase::class.
                                java,

                                "user_database"

                                ).build()

                                instance =
                                newInstance

                                newInstance
                                            }
                                        }
                                    }
                                }
```

Step 4: Create an UserDatabaseHelper class

```
package
com.example
```

```
.snackorder
ing

            import
            android.annotation.Suppress
            Lint
            import
            android.content.ContentValu
            es
            import
            android.content.Context
            import
            android.database.Cursor
            import
            android.database.sqlite.SQL
            iteDatabase
            import
            android.database.sqlite.SQL
            iteOpenHelper

            class
            UserDatabaseHelper(context:
            Context) :

            SQLiteOpenHelper(context,
            DATABASE_NAME, null,
            DATABASE_VERSION) {
```

```
    companion object {
        private const val
DATABASE_VERSION = 1
        private const val
DATABASE_NAME =
"UserDatabase.db"

        private const val
TABLE_NAME = "user_table"
        private const val
COLUMN_ID = "id"
        private const val
COLUMN_FIRST_NAME =
"first_name"
        private const val
COLUMN_LAST_NAME =
"last_name"
        private const val
COLUMN_EMAIL = "email"
        private const val
COLUMN_PASSWORD =
"password"
    }
    override fun
onCreate(db:
SQLiteDatabase?) {
        val createTable =
"CREATE TABLE $TABLE_NAME
(" +
                "$COLUMN_ID
INTEGER PRIMARY KEY
AUTOINCREMENT, " +

"$COLUMN_FIRST_NAME TEXT, "
+

"$COLUMN_LAST_NAME TEXT, "
+

"$COLUMN_EMAIL TEXT, " +

"$COLUMN_PASSWORD TEXT" +
                ")"
```

```
db?.execSQL(createTable)

    }

    override fun
onUpgrade(db:
SQLiteDatabase?,
oldVersion: Int,
newVersion: Int) {
        db?.execSQL("DROP
TABLE IF EXISTS
$TABLE_NAME")
        onCreate(db)
    }

    fun insertUser(user:
User) {
        val db =
writableDatabase
```

```
        val values =
ContentValues()

values.put(COLUMN_FIRST_NAM
E, user.firstName)

values.put(COLUMN_LAST_NAME
, user.lastName)

values.put(COLUMN_EMAIL,
user.email)

values.put(COLUMN_PASSWORD,
user.password)

db.insert(TABLE_NAME, null,
values)
        db.close()
    }

    @SuppressLint("Range")
    fun
getUserByUsername(username:
String): User? {
        val db =
readableDatabase
```

```
        val cursor: Cursor
= db.rawQuery("SELECT *
FROM $TABLE_NAME WHERE
$COLUMN_FIRST_NAME = ?",
arrayOf(username))
        var user: User? =
null
        if
(cursor.moveToFirst()) {
            user = User(
                id =
cursor.getInt(cursor.getCol
umnIndex(COLUMN_ID)),
                firstName =
cursor.getString(cursor.get
ColumnIndex(COLUMN_FIRST_NA
ME)),
                lastName =
cursor.getString(cursor.get
ColumnIndex(COLUMN_LAST_NAM
E)),
                email =
cursor.getString(cursor.get
ColumnIndex(COLUMN_EMAIL)),
                password =
cursor.getString(cursor.get
ColumnIndex(COLUMN_PASSWORD
)),
            )
        }
        cursor.close()
        db.close()
        return user
    }
    @SuppressLint("Range")
    fun getUserById(id:
Int): User? {
        val db =
readableDatabase
        val cursor: Cursor
= db.rawQuery("SELECT *
FROM $TABLE_NAME WHERE
$COLUMN_ID = ?",
arrayOf(id.toString()))
        var user: User? =
null
        if
(cursor.moveToFirst()) {
            user = User(
                id =
cursor.getInt(cursor.getCol
umnIndex(COLUMN_ID)),
                firstName =
cursor.getString(cursor.get
ColumnIndex(COLUMN_FIRST_NA
ME)),
                lastName =
cursor.getString(cursor.get
ColumnIndex(COLUMN_LAST_NAM
E)),
                email =
cursor.getString(cursor.get
ColumnIndex(COLUMN_EMAIL)),
                password =
cursor.getString(cursor.get
ColumnIndex(COLUMN_PASSWORD
)),
            )
        }
        cursor.close()
        db.close()
        return user
    }

    @SuppressLint("Range")
    fun getAllUsers():
List<User> {
        val users =
mutableListOf<User>()
```

```
        val db =
readableDatabase
        val cursor: Cursor
= db.rawQuery("SELECT *
FROM $TABLE_NAME", null)
        if
(cursor.moveToFirst()) {
            do {
                val user =
User(
                    id =
cursor.getInt(cursor.getCol
umnIndex(COLUMN_ID)),

firstName =
cursor.getString(cursor.get
ColumnIndex(COLUMN_FIRST_NA
ME)),

lastName =
cursor.getString(cursor.get
ColumnIndex(COLUMN_LAST_NAM
E)),
                    email =
cursor.getString(cursor.get
ColumnIndex(COLUMN_EMAIL)),
password =
cursor.getString(cursor.get
ColumnIndex(COLUMN_PASSWORD
)),
                )

users.add(user)
            } while
(cursor.moveToNext())
        }
        cursor.close()
        db.close()
        return users
    }

}
```

Database 2:

Step 1: Create an Order data class

```
package
com.example.snackord
ering

                    import
                    androidx.room.Colum
                    nInfo
                    import
                    androidx.room.Entit
                    y
                    import
                    androidx.room.Prima
                    ryKey

                    @Entity(tableName =
                    "order_table")
                    data class Order(

                    @PrimaryKey(autoGen
                    erate = true) val
                    id: Int?,

                    @ColumnInfo(name =
                    "quantity") val
                    quantity: String?,

                    @ColumnInfo(name =
                    "address") val
                    address: String?,
                    )
```

Step 2: Create OrderDao interface

```
package
com.example.snackor
dering
```

```
import
androidx.room.*

@Dao
interface OrderDao {

    @Query("SELECT *
FROM order_table
WHERE  address=
:address")
    suspend fun
getOrderByAddress(ad
dress: String):
Order?
@Insert(onConflict =
OnConflictStrategy.R
EPLACE)
    suspend fun
insertOrder(order:
Order)

    @Update
    suspend fun
updateOrder(order:
Order)

    @Delete
    suspend fun
deleteOrder(order:
Order)
}
```

Step 3: Create OrderDatabase class

```
package
com.example.snackor
dering
```

```
import
android.content.Cont
ext
import
androidx.room.Databa
se
import
androidx.room.Room
import
androidx.room.RoomDa
tabase

@Database(entities =
[Order::class],
version = 1)
abstract class
OrderDatabase :
RoomDatabase() {

    abstract fun
orderDao(): OrderDao

    companion object
{

        @Volatile
        private var
instance:
OrderDatabase? =
null

        fun
getDatabase(context:
Context):
OrderDatabase {
            return
instance ?:
synchronized(this) {
                val
newInstance =
Room.databaseBuilder
(

context.applicationC
ontext,

OrderDatabase::class
.java,

"order_database"

).build()
```

```
                          instance =
                          newInstance

                          newInstance
                                      }
                                  }
                              }
                          }
```

Step 4: Create OderDatabaseHelper class

```
package
com.example.
snackorderin
g

             import
             android.annotation.Suppress
             Lint
             import
             android.content.ContentValu
             es
```

```
import
android.content.Context
import
android.database.Cursor
import
android.database.sqlite.SQL
iteDatabase
import
android.database.sqlite.SQL
iteOpenHelper

class
OrderDatabaseHelper(context
: Context) :

SQLiteOpenHelper(context,
DATABASE_NAME,
null,DATABASE_VERSION){

    companion object {
        private const val
DATABASE_VERSION = 1
        private const val
DATABASE_NAME =
"OrderDatabase.db"
```

```
        private const val
TABLE_NAME = "order_table"
        private const val
COLUMN_ID = "id"
        private const val
COLUMN_QUANTITY =
"quantity"
        private const val
COLUMN_ADDRESS = "address"
    }

    override fun
onCreate(db:
SQLiteDatabase?) {
        val createTable =
"CREATE TABLE $TABLE_NAME
(" +

"${COLUMN_ID} INTEGER
PRIMARY KEY AUTOINCREMENT,
" +

"${COLUMN_QUANTITY} Text, "
+

"${COLUMN_ADDRESS} TEXT " +
                ")"
```

```
db?.execSQL(createTable)
    }

    override fun
onUpgrade(db:
SQLiteDatabase?,
oldVersion: Int,
newVersion: Int) {
        db?.execSQL("DROP
TABLE IF EXISTS
$TABLE_NAME")
        onCreate(db)
    }

    fun insertOrder(order:
Order) {
        val db =
writableDatabase
        val values =
ContentValues()

values.put(COLUMN_QUANTITY,
order.quantity)
```

```
values.put(COLUMN_ADDRESS,
order.address)

db.insert(TABLE_NAME, null,
values)
        db.close()
    }



    @SuppressLint("Range")
    fun
getOrderByQuantity(quantity
: String): Order? {
        val db =
readableDatabase
        val cursor: Cursor
= db.rawQuery("SELECT *
FROM $TABLE_NAME WHERE
$COLUMN_QUANTITY = ?",
arrayOf(quantity))
        var order: Order? =
null
        if
(cursor.moveToFirst()) {
            order = Order(
                id =
cursor.getInt(cursor.getCol
umnIndex(COLUMN_ID)),
                quantity =
cursor.getString(cursor.get
ColumnIndex(COLUMN_QUANTITY
)),
                address =
cursor.getString(cursor.get
ColumnIndex(COLUMN_ADDRESS)
),
            )
        }
        cursor.close()
        db.close()
        return order
    }
    @SuppressLint("Range")
    fun getOrderById(id:
Int): Order? {
        val db =
readableDatabase
        val cursor: Cursor
= db.rawQuery("SELECT *
FROM $TABLE_NAME WHERE
$COLUMN_ID = ?",
arrayOf(id.toString()))
```

```
        var order: Order? =
null
        if
(cursor.moveToFirst()) {
            order = Order (
                id =
cursor.getInt(cursor.getCol
umnIndex(COLUMN_ID)),
                quantity =
cursor.getString(cursor.get
ColumnIndex(COLUMN_QUANTITY
)),
                address =
cursor.getString(cursor.get
ColumnIndex(COLUMN_ADDRESS)
),
            )
        }
        cursor.close()
        db.close()
        return order
    }

    @SuppressLint("Range")
    fun getAllOrders():
List<Order> {
        val orders =
mutableListOf<Order>()
        val db =
readableDatabase
        val cursor: Cursor
= db.rawQuery("SELECT *
FROM $TABLE_NAME", null)
        if
(cursor.moveToFirst()) {
            do {
                val order =
Order(
                    id =
cursor.getInt(cursor.getCol
umnIndex(COLUMN_ID)),

quantity =
cursor.getString(cursor.get
ColumnIndex(COLUMN_QUANTITY
)),
                    address
=
cursor.getString(cursor.get
ColumnIndex(COLUMN_ADDRESS)
),
                )
orders.add(order)
            } while
(cursor.moveToNext())
        }
        cursor.close()
        db.close()
        return orders
    }

}
```

Building Application UI And Connecting to Database

Step 1: Creating LoginActivity.kt With database

```
package
com.example.
snackorderin
g

              import
              android.content.Context
```

```
import
android.content.Intent
import android.os.Bundle
import
androidx.activity.Component
Activity
import
androidx.activity.compose.s
etContent
import
androidx.compose.foundation
.Image
import
androidx.compose.foundation
.layout.*
import
androidx.compose.material.*
import
androidx.compose.runtime.*
import
androidx.compose.ui.Alignme
nt
import
androidx.compose.ui.Modifie
r
```

```
import
androidx.compose.ui.graphic
s.Color
import
androidx.compose.ui.layout.
ContentScale
import
androidx.compose.ui.res.pai
nterResource
import
androidx.compose.ui.text.fo
nt.FontFamily
import
androidx.compose.ui.text.fo
nt.FontWeight
import
androidx.compose.ui.unit.dp
import
androidx.compose.ui.unit.sp
import
androidx.core.content.Conte
xtCompat
import
com.example.snackordering.u
i.theme.SnackOrderingTheme
```

```
class LoginActivity :
ComponentActivity() {
    private lateinit var
databaseHelper:
UserDatabaseHelper
    override fun
onCreate(savedInstanceState
: Bundle?) {

super.onCreate(savedInstanc
eState)
        databaseHelper =
UserDatabaseHelper(this)
        setContent {

SnackOrderingTheme {
                // A
surface container using the
'background' color from the
theme
                Surface(

modifier =
Modifier.fillMaxSize(),
                    color =
MaterialTheme.colors.backgr
ound
```

```
                ) {

LoginScreen(this,
databaseHelper)
                }
            }
        }
    }
}
@Composable
fun LoginScreen(context:
Context, databaseHelper:
UserDatabaseHelper) {


Image(painterResource(id =
R.drawable.order),
contentDescription = "",
        alpha =0.3F,
        contentScale =
ContentScale.FillHeight,

    )

    var username by
remember {
mutableStateOf("") }
```

```
    var password by
remember {
mutableStateOf("") }
    var error by remember {
mutableStateOf("") }

    Column(
        modifier =
Modifier.fillMaxSize(),
        horizontalAlignment
=
Alignment.CenterHorizontall
y,
        verticalArrangement
= Arrangement.Center
    ) {

        Text(
            fontSize =
36.sp,
            fontWeight =
FontWeight.ExtraBold,
            fontFamily =
FontFamily.Cursive,
            color =
Color.White,
            text = "Login"
        )
        Spacer(modifier =
Modifier.height(10.dp))

        TextField(
            value =
username,
            onValueChange =
{ username = it },
            label = {
Text("Username") },
            modifier =
Modifier.padding(10.dp)

.width(280.dp)
        )

        TextField(
            value =
password,
            onValueChange =
{ password = it },
            label = {
Text("Password") },
            modifier =
Modifier.padding(10.dp)
```

```
.width(280.dp)
        )

        if
(error.isNotEmpty()) {
            Text(
                text =
error,
                color =
MaterialTheme.colors.error,
                modifier =
Modifier.padding(vertical =
16.dp)
            )
        }

        Button(
            onClick = {
                if
(username.isNotEmpty() &&
password.isNotEmpty()) {
                    val
user =
databaseHelper.getUserByUse
rname(username)
```

```
                    if
(user != null &&
user.password == password)
{

error = "Successfully log
in"

context.startActivity(

Intent(

context,

MainPage::class.java

)
                        )

//onLoginSuccess()
                    }
                        if
(user != null &&
user.password == "admin") {

error = "Successfully log
in"
context.startActivity(

Intent(

context,

AdminActivity::class.java

)

)
                        }

else {

error =  "Invalid username
or password"
                        }

                } else {
                    error =
"Please fill all fields"
                }
            },
            modifier =
Modifier.padding(top =
16.dp)
        ) {
            Text(text =
"Login")
        }
        Row {

TextButton(onClick =
{context.startActivity(
                Intent(

context,

MainActivity::class.java
                )
            )}
            )
            { Text(color =
Color.White,text = "Sign
up") }

TextButton(onClick = {
            })

            {
```

```
Spacer(modifier =
Modifier.width(60.dp))
                Text(color
= Color.White,text =
"Forget password?")
            }
        }
    }
}
private fun
startMainPage(context:
Context) {
    val intent =
Intent(context,
MainPage::class.java)

ContextCompat.startActivity
(context, intent, null)
}
```

Step 2: Creating MainActivity.kt With database

```
package
com.example.
snackorderin
g
```

```
import
android.content.Context
import
android.content.Intent
import android.os.Bundle
import
androidx.activity.Component
Activity
import
androidx.activity.compose.s
etContent
import
androidx.compose.foundation
.Image
import
androidx.compose.foundation
.layout.*
import
androidx.compose.material.*
import
androidx.compose.runtime.*
import
androidx.compose.ui.Alignme
nt
```

```
import
androidx.compose.ui.Modifie
r
import
androidx.compose.ui.graphic
s.Color
import
androidx.compose.ui.layout.
ContentScale
import
androidx.compose.ui.res.pai
nterResource
import
androidx.compose.ui.text.fo
nt.FontFamily
import
androidx.compose.ui.text.fo
nt.FontWeight
import
androidx.compose.ui.unit.dp
import
androidx.compose.ui.unit.sp
import
androidx.core.content.Conte
xtCompat
```

```
import
com.example.snackordering.u
i.theme.SnackOrderingTheme

class MainActivity :
ComponentActivity() {
    private lateinit var
databaseHelper:
UserDatabaseHelper
    override fun
onCreate(savedInstanceState
: Bundle?) {

super.onCreate(savedInstanc
eState)
        databaseHelper =
UserDatabaseHelper(this)
        setContent {

SnackOrderingTheme {
                // A
surface container using the
'background' color from the
theme
                Surface(
modifier =
Modifier.fillMaxSize(),
                    color =
MaterialTheme.colors.backgr
ound
                ) {


RegistrationScreen(this,dat
abaseHelper)
                }
            }
        }
    }
}


@Composable
fun
RegistrationScreen(context:
Context, databaseHelper:
UserDatabaseHelper) {

    Image(
        painterResource(id
= R.drawable.order),
contentDescription = "",
        alpha =0.3F,
        contentScale =
ContentScale.FillHeight,

        )

    var username by
remember {
mutableStateOf("") }
    var password by
remember {
mutableStateOf("") }
    var email by remember {
mutableStateOf("") }
    var error by remember {
mutableStateOf("") }

    Column(
        modifier =
Modifier.fillMaxSize(),
        horizontalAlignment
=
Alignment.CenterHorizontall
y,
        verticalArrangement
= Arrangement.Center
    ) {

        Text(
            fontSize =
36.sp,
            fontWeight =
FontWeight.ExtraBold,
            fontFamily =
FontFamily.Cursive,
            color =
Color.White,
            text =
"Register"
        )

        Spacer(modifier =
Modifier.height(10.dp))
        TextField(
            value =
username,
            onValueChange =
{ username = it },
            label = {
Text("Username") },
            modifier =
Modifier

.padding(10.dp)

.width(280.dp)

        )

        TextField(
            value = email,
            onValueChange =
{ email = it },
            label = {
Text("Email") },
            modifier =
Modifier

.padding(10.dp)

.width(280.dp)
        )

        TextField(
            value =
password,
            onValueChange =
{ password = it },
            label = {
Text("Password") },
            modifier =
Modifier

.padding(10.dp)

.width(280.dp)
        )


        if
(error.isNotEmpty()) {
            Text(
                text =
error,
                color =
MaterialTheme.colors.error,
                modifier =
Modifier.padding(vertical =
16.dp)
            )
        }

        Button(
            onClick = {
                if
(username.isNotEmpty() &&
password.isNotEmpty() &&
email.isNotEmpty()) {
                    val
user = User(
                        id
= null,

firstName = username,

lastName = null,

email = email,

password = password
                    )

databaseHelper.insertUser(u
ser)
                    error =
"User registered
successfully"
                    //
Start LoginActivity using
the current context
context.startActivity(

Intent(

context,

LoginActivity::class.java
                         )
                     )

                 } else {
                     error =
"Please fill all fields"
                 }
             },
             modifier =
Modifier.padding(top =
16.dp)
        ) {
            Text(text =
"Register")
        }
        Spacer(modifier =
Modifier.width(10.dp))
        Spacer(modifier =
Modifier.height(10.dp))
```

```
        Row() {
            Text(
                modifier =
Modifier.padding(top =
14.dp), text = "Have an
account?"
            )

TextButton(onClick = {

context.startActivity(
                    Intent(

context,

LoginActivity::class.java
                    )
                )
            })

            {

Spacer(modifier =
Modifier.width(10.dp))
                Text(text =
"Log in")
```

```
                }
            }
        }
    }
    private fun
    startLoginActivity(context:
    Context) {
        val intent =
    Intent(context,
    LoginActivity::class.java)

    ContextCompat.startActivity
    (context, intent, null)
    }
```

Step 3: Creating MainPage.kt file

```
package
com.example.
snackorderin
g

    import
    android.annotation.Suppress
    Lint
```

```
import
android.content.Context
import android.os.Bundle
import android.widget.Toast
import
androidx.activity.Component
Activity
import
androidx.activity.compose.s
etContent
import
androidx.annotation.Drawabl
eRes
import
androidx.annotation.StringR
es
import
androidx.compose.foundation
.Image
import
androidx.compose.foundation
.background
import
androidx.compose.foundation
.layout.*
```

```
import
androidx.compose.foundation
.shape.CircleShape
import
androidx.compose.foundation
.shape.RoundedCornerShape
import
androidx.compose.material.*
import
androidx.compose.material.i
cons.Icons
import
androidx.compose.material.i
cons.filled.*
import
androidx.compose.runtime.Co
mposable
import
androidx.compose.ui.Alignme
nt
import
androidx.compose.ui.Modifie
r
import
androidx.compose.ui.draw.cl
ip
```

```
import
androidx.compose.ui.graphic
s.Color
import
androidx.compose.foundation
.lazy.LazyColumn
import
androidx.compose.foundation
.lazy.items
import
androidx.compose.material.T
ext
import
androidx.compose.ui.unit.dp
import
androidx.compose.ui.graphic
s.RectangleShape
import
androidx.compose.ui.layout.
ContentScale
import
androidx.compose.ui.platfor
m.LocalContext
import
androidx.compose.ui.res.pai
nterResource
```

```
import
androidx.compose.ui.res.str
ingResource
import
androidx.compose.ui.text.fo
nt.FontWeight
import
androidx.compose.ui.unit.sp
import
androidx.core.content.Conte
xtCompat.startActivity
import
com.example.snackordering.u
i.theme.SnackOrderingTheme

import
android.content.Intent as
Intent1


class MainPage :
ComponentActivity() {
    override fun
onCreate(savedInstanceState
: Bundle?) {
```

```
super.onCreate(savedInstanc
eState)
        setContent {

SnackOrderingTheme {
                // A
surface container using the
'background' color from the
theme
                Surface(

modifier =
Modifier.fillMaxSize(),
                    color =
MaterialTheme.colors.backgr
ound
                ) {

FinalView(this)
                    val
context =
LocalContext.current

//PopularFoodColumn(context
)
                }
            }
        }
    }
}


@Composable
fun TopPart() {

    Row(
        modifier = Modifier
            .fillMaxWidth()

.background(Color(0xffeceef
0)),
Arrangement.SpaceBetween
    ) {
        Icon(
            imageVector =
Icons.Default.Add,
contentDescription = "Menu
Icon",
            Modifier


.clip(CircleShape)
```

```
.size(40.dp),
            tint =
Color.Black,
        )

Column(horizontalAlignment
=
Alignment.CenterHorizontall
y) {
            Text(text =
"Location", style =
MaterialTheme.typography.su
btitle1, color =
Color.Black)
            Row {
                Icon(

imageVector =
Icons.Default.LocationOn,

contentDescription =
"Location",
                    tint =
Color.Red,
                )
                Text(text =
"Accra" , color =
Color.Black)
            }

        }
        Icon(
            imageVector =
Icons.Default.Notifications
, contentDescription =
"Notification Icon",

            Modifier

.size(45.dp),
            tint =
Color.Black,
        )
    }
}

@Composable
fun CardPart() {
    Card(modifier =
Modifier.size(width =
310.dp, height = 150.dp),
RoundedCornerShape(20.dp))
{
        Row(modifier =
Modifier.padding(10.dp),
Arrangement.SpaceBetween) {

Column(verticalArrangement
=
Arrangement.spacedBy(12.dp)
) {
                Text(text =
"Get Special Discounts")
                Text(text =
"up to 85%", style =
MaterialTheme.typography.h5
)

Button(onClick = {}, colors
=
ButtonDefaults.buttonColors
(Color.White)) {

Text(text = "Claim
voucher", color =
MaterialTheme.colors.surfac
e)
                }
```

```
            }
            Image(
                painter =
painterResource(id =
R.drawable.food_tip_im),

contentDescription = "Food
Image", Modifier.size(width
= 100.dp, height = 200.dp)
            )
        }
    }
}


@Composable
fun PopularFood(
    @DrawableRes drawable:
Int,
    @StringRes text1: Int,
    context: Context
) {
    Card(
        modifier = Modifier

.padding(top=20.dp, bottom
= 20.dp, start = 65.dp)
```

```
            .width(250.dp)

    ) {
        Column(

verticalArrangement =
Arrangement.Top,

horizontalAlignment =
Alignment.CenterHorizontall
y
        ) {
            Spacer(modifier
= Modifier.padding(vertical
= 5.dp))
            Row(
                modifier =
Modifier

.fillMaxWidth(0.7f),
Arrangement.End
            ) {
                Icon(

imageVector =
Icons.Default.Star,
contentDescription = "Star
Icon",
                    tint =
Color.Yellow
                )
                Text(text =
"4.3", fontWeight =
FontWeight.Black)
            }
            Image(
                painter =
painterResource(id =
drawable),

contentDescription = "Food
Image",

contentScale =
ContentScale.Crop,
                modifier =
Modifier

.size(100.dp)

.clip(CircleShape)
            )
            Text(text =
stringResource(id = text1),
fontWeight =
FontWeight.Bold)
            Row(modifier =
Modifier.fillMaxWidth(0.7f)
, Arrangement.SpaceBetween)
{
                /*TODO
Implement Prices for each
card*/
                Text(
                    text =
"$50",
                    style =
MaterialTheme.typography.h6
,

fontWeight =
FontWeight.Bold,

fontSize = 18.sp
                )


IconButton(onClick = {
```

```
                    //var
no=FoodList.lastIndex;

//Toast.
                    val
intent = Intent1(context,
TargetActivity::class.java)

context.startActivity(inten
t)

                }) {
                    Icon(

imageVector =
Icons.Default.ShoppingCart,

contentDescription =
"shopping cart",
                    )
                }
            }
        }
    }
}
```

```
private val FoodList =
listOf(
    R.drawable.sandwish to
R.string.sandwich,
    R.drawable.sandwish to
R.string.burgers,
    R.drawable.pack to
R.string.pack,
    R.drawable.pasta to
R.string.pasta,
    R.drawable.tequila to
R.string.tequila,
    R.drawable.wine to
R.string.wine,
    R.drawable.salad to
R.string.salad,
    R.drawable.pop to
R.string.popcorn
).map {
DrawableStringPair(it.first
, it.second) }

private data class
DrawableStringPair(
    @DrawableRes val
drawable: Int,
    @StringRes val text1:
Int
)


@Composable
fun App(context: Context) {

    Column(
        modifier = Modifier
            .fillMaxSize()

.background(Color(0xffeceef
0))

.padding(10.dp),
        verticalArrangement
= Arrangement.Top,
        horizontalAlignment
=
Alignment.CenterHorizontall
y
    ) {
        Surface(modifier =
Modifier, elevation = 5.dp)
{
            TopPart()
        }
        Spacer(modifier =
Modifier.padding(10.dp))
        CardPart()

        Spacer(modifier =
Modifier.padding(10.dp))
        Row(modifier =
Modifier.fillMaxWidth(),
Arrangement.SpaceBetween) {
            Text(text =
"Popular Food", style =
MaterialTheme.typography.h5
, color = Color.Black)
            Text(text =
"view all", style =
MaterialTheme.typography.su
btitle1,  color =
Color.Black)
        }
        Spacer(modifier =
Modifier.padding(10.dp))
```

```
PopularFoodColumn(context)
// <- call the function
with parentheses
    }
}




@Composable
fun
PopularFoodColumn(context:
Context) {

    LazyColumn(
        modifier =
Modifier.fillMaxSize(),

        content = {
            items(FoodList)
{ item ->

PopularFood(context =
context,drawable =
item.drawable, text1 =
item.text1)
```

```
                abstract
class Context
            }
        },
        verticalArrangement
=
Arrangement.spacedBy(16.dp)
)
}


@SuppressLint("UnusedMateri
alScaffoldPaddingParameter"
)
@Composable
fun FinalView(mainPage:
MainPage) {
    SnackOrderingTheme {
        Scaffold() {
            val context =
LocalContext.current
            App(context)
        }
    }
}
```

Step 4: Creating TargetActivity.kt

```
package
com.example.
snackorderin
g
```

```
import
android.content.Context
import
android.content.Intent
import android.os.Bundle
import android.util.Log
import android.widget.Toast
import
androidx.activity.Component
Activity
import
androidx.activity.compose.s
etContent
import
androidx.compose.foundation
.Image
import
androidx.compose.foundation
.background
```

```
import
androidx.compose.foundation
.layout.*
import
androidx.compose.foundation
.text.KeyboardActions
import
androidx.compose.foundation
.text.KeyboardOptions
import
androidx.compose.material.*
import
androidx.compose.runtime.*
import
androidx.compose.ui.Alignme
nt
import
androidx.compose.ui.Modifie
r
import
androidx.compose.ui.graphic
s.Color
import
androidx.compose.ui.layout.
ContentScale
```

```
import
androidx.compose.ui.platfor
m.LocalContext
import
androidx.compose.ui.platfor
m.textInputServiceFactory
import
androidx.compose.ui.res.pai
nterResource
import
androidx.compose.ui.text.in
put.KeyboardType
import
androidx.compose.ui.tooling
.preview.Preview
import
androidx.compose.ui.unit.dp
import
androidx.core.content.Conte
xtCompat
import
com.example.snackordering.u
i.theme.SnackOrderingTheme

class TargetActivity :
ComponentActivity() {
    private lateinit var
orderDatabaseHelper:
OrderDatabaseHelper
    override fun
onCreate(savedInstanceState
: Bundle?) {

super.onCreate(savedInstanc
eState)
        orderDatabaseHelper
= OrderDatabaseHelper(this)
        setContent {

SnackOrderingTheme {
                // A
surface container using the
'background' color from the
theme
                Surface(

modifier = Modifier

.fillMaxSize()

.background(Color.White)

                ) {
```

```
Order(this,
orderDatabaseHelper)
                    val
orders =
orderDatabaseHelper.getAllO
rders()

Log.d("swathi",
orders.toString())

                }
            }
        }
    }
}

@Composable
fun Order(context: Context,
orderDatabaseHelper:
OrderDatabaseHelper){

Image(painterResource(id =
R.drawable.order),
contentDescription = "",
        alpha =0.5F,
    contentScale =
ContentScale.FillHeight)
    Column(
        horizontalAlignment
=
Alignment.CenterHorizontall
y,
        verticalArrangement
= Arrangement.Center) {

        val mContext =
LocalContext.current
        var quantity by
remember {
mutableStateOf("") }
        var address by
remember {
mutableStateOf("") }
        var error by
remember {
mutableStateOf("") }


        TextField(value =
quantity, onValueChange =
{quantity=it},
            label = {
Text("Quantity") },
           keyboardOptions
=
KeyboardOptions(keyboardTyp
e = KeyboardType.Number),
            modifier =
Modifier

.padding(10.dp)

.width(280.dp))

        Spacer(modifier =
Modifier.padding(10.dp))

        TextField(value =
address, onValueChange =
{address=it},
            label = {
Text("Address") },
            modifier =
Modifier

.padding(10.dp)

.width(280.dp))
```

```
        Spacer(modifier =
Modifier.padding(10.dp))


        if
(error.isNotEmpty()) {
            Text(
                text =
error,
                color =
MaterialTheme.colors.error,
                modifier =
Modifier.padding(vertical =
16.dp)
            )
        }


        Button(onClick = {
            if(
quantity.isNotEmpty() and
address.isNotEmpty()){
                val order =
Order(
                    id =
null,
quantity = quantity,
                    address
= address
                )

orderDatabaseHelper.insertO
rder(order)

Toast.makeText(mContext,
"Order Placed
Successfully",
Toast.LENGTH_SHORT).show()}
        },
            colors =
ButtonDefaults.buttonColors
(backgroundColor =
Color.White))
        {
            Text(text =
"Order Place", color =
Color.Black)
        }


    }
}
```

```
private fun
startMainPage(context:
Context) {
    val intent =
Intent(context,
LoginActivity::class.java)

ContextCompat.startActivity
(context, intent, null)
}
```

Step 4: Creating AdminActivity.kt

```
package
com.example.
snackorderin
g
```

```
import
android.icu.text.SimpleDate
Format
import android.os.Bundle
import android.util.Log
import
androidx.activity.Component
Activity
```

```
import
androidx.activity.compose.s
etContent
import
androidx.compose.foundation
.Image
import
androidx.compose.foundation
.layout.*
import
androidx.compose.foundation
.lazy.LazyColumn
import
androidx.compose.foundation
.lazy.LazyRow
import
androidx.compose.foundation
.lazy.items
import
androidx.compose.material.M
aterialTheme
import
androidx.compose.material.S
urface
import
androidx.compose.material.T
ext
```

```
import
androidx.compose.runtime.Co
mposable
import
androidx.compose.ui.Modifie
r
import
androidx.compose.ui.graphic
s.Color
import
androidx.compose.ui.layout.
ContentScale
import
androidx.compose.ui.res.pai
nterResource
import
androidx.compose.ui.unit.dp
import
androidx.compose.ui.unit.sp
import
com.example.snackordering.u
i.theme.SnackOrderingTheme
import java.util.*

class AdminActivity :
ComponentActivity() {
    private lateinit var
orderDatabaseHelper:
OrderDatabaseHelper
    override fun
onCreate(savedInstanceState
: Bundle?) {

super.onCreate(savedInstanc
eState)
        orderDatabaseHelper
= OrderDatabaseHelper(this)
        setContent {

SnackOrderingTheme {
                // A
surface container using the
'background' color from the
theme
                Surface(

modifier =
Modifier.fillMaxSize(),
                    color =
MaterialTheme.colors.backgr
ound
                ) {
                    val
data=orderDatabaseHelper.ge
tAllOrders();

Log.d("swathi"
,data.toString())
                    val
order =
orderDatabaseHelper.getAllO
rders()

ListListScopeSample(order)
                }
            }
        }
    }
}

@Composable
fun
ListListScopeSample(order:
List<Order>) {
    Image(
        painterResource(id
= R.drawable.order),
contentDescription = "",
        alpha =0.5F,
        contentScale =
ContentScale.FillHeight)
    Text(text = "Order
Tracking", modifier =
Modifier.padding(top =
24.dp, start = 106.dp,
bottom = 24.dp ), color =
Color.White, fontSize =
30.sp)
    Spacer(modifier =
Modifier.height(30.dp))
    LazyRow(
        modifier = Modifier
            .fillMaxSize()
            .padding(top =
80.dp),


horizontalArrangement =
Arrangement.SpaceBetween
    ){
        item {

            LazyColumn {

items(order) { order ->
```

```
                    Column(modifier =
                    Modifier.padding(top =
                    16.dp, start = 48.dp,
                    bottom = 20.dp)) {

                    Text("Quantity:
                    ${order.quantity}")

                    Text("Address:
                    ${order.address}")
                                        }
                                    }
                                }
                            }

                        }
                    }
```

Task: 6
Complete AndroidManifest.xml code:

```
<?xml
version
="1.0"
```

```
encodin
g="utf-
8"?>
          <manifest
          xmlns:android="http://schemas.an
          droid.com/apk/res/android"

          xmlns:tools="http://schemas.andr
          oid.com/tools">

              <application

          android:allowBackup="true"

          android:dataExtractionRules="@xm
          l/data_extraction_rules"

          android:fullBackupContent="@xml/
          backup_rules"

          android:icon="@drawable/fast_foo
          d"

          android:label="@string/app_name"

          android:supportsRtl="true"
```

```
android:theme="@style/Theme.Snac
kOrdering"
        tools:targetApi="31">
        <activity

android:name=".AdminActivity"

android:exported="false"

android:label="@string/title_act
ivity_admin"

android:theme="@style/Theme.Snac
kOrdering" />
        <activity

android:name=".LoginActivity"

android:exported="true"

android:label="SnackSquad"

android:theme="@style/Theme.Snac
kOrdering">
            <intent-filter>
```

```
                <action
        android:name="android.intent.act
        ion.MAIN" />

                <category
        android:name="android.intent.cat
        egory.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity

        android:name=".TargetActivity"

        android:exported="false"

        android:label="@string/title_act
        ivity_target"

        android:theme="@style/Theme.Snac
        kOrdering" />
        <activity

        android:name=".MainPage"

        android:exported="false"
```

```
android:label="@string/title_act
ivity_main_page"

android:theme="@style/Theme.Snac
kOrdering" />
        <activity

android:name=".MainActivity"

android:exported="false"

android:label="MainActivity"

android:theme="@style/Theme.Snac
kOrdering" />
    </application>

</manifest>
```