

# Project Report

## 1 Introduction:

### 1.1 Overview

Welcome to the Snack Ordering Application project! Our goal is to develop a user-friendly and efficient mobile application that allows users to easily order their favorite snacks from the comfort of their homes. This application will be developed using Kotlin and Java, two popular programming languages for Android app development.

In today's fast-paced world, people often crave snacks to satisfy their hunger or indulge in their favorite treats. However, traditional methods of snack ordering can be time-consuming and inconvenient. Our Snack Ordering Application aims to streamline this process by providing a seamless and delightful user experience.

### 1.2 Purpose


The purpose of the Snack Ordering Application project, developed using Kotlin and Java, is to create a convenient and enjoyable platform for users to order their favorite snacks with ease. This application aims to solve the challenges and limitations faced by traditional snack ordering methods by leveraging the capabilities of mobile technology.

The primary purpose of the Snack Ordering Application is to provide users with a seamless and user-friendly experience when it comes to ordering snacks. Instead of going through the hassle of visiting physical stores or making phone calls, users can simply use the application to explore a wide range of snacks and place their orders from anywhere at any time.

# 2 Problem Definition & Design Thinking:

## 2.1 Empathy Map

Template



### 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.

[Share template feedback](#)

### 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?

- Users see a visually appealing interface with attractive snack images.
- They notice clear and intuitive navigation options to browse different snack categories.

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

- They observe personalized suggestions and promotions tailored to their tastes.
- Users may encounter error messages or notifications when there are issues with the ordering process.
- I hope this application offers a wide variety of snack options to choose from.
- I want to find snacks that cater to my dietary preferences or restrictions.
- I hope the app provides personalized recommendations based on my past orders or preferences.
- I wonder if the app has a user-friendly interface that makes it easy to navigate and find what I want.

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


- Users may feel excited and hungry when using the application.
- They might feel frustrated if the ordering process is complicated or slow.
- Users may feel relieved if they can easily find their favorite snacks and place orders quickly.
- They might feel satisfied if the application offers personalized recommendations based on their preferences.

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

- Users may share their positive experiences with the application on social media platforms or recommend it to others.
- They might leave reviews or ratings on the app store based on their overall satisfaction.
- Users might provide feedback or report any issues they encounter while using the application.
- They may actively explore different snack options, add items to their cart, and proceed with the checkout process.

## 2.2 Ideation & Brainstorming Map

Template



### Brainstorm & idea prioritization

Use this template in your own brainstorming sessions to help you generate ideas and prioritize them. The template includes a list of ideas and a prioritization matrix to help you select the best ideas for your project.

[Share template feedback](#)

### Brainstorm

Use this template to brainstorm ideas for your project. The template includes a list of ideas and a prioritization matrix to help you select the best ideas for your project.

**Before you brainstorm**

- Define your problem statement
- Brainstorm ideas
- Prioritize ideas

**Brainstorming session**

- Define your problem statement
- Brainstorm ideas
- Prioritize ideas

**After you brainstorm**

- Define your problem statement
- Brainstorm ideas
- Prioritize ideas

### Ideation & Brainstorming Map

Use this template to brainstorm ideas for your project. The template includes a list of ideas and a prioritization matrix to help you select the best ideas for your project.

**Before you brainstorm**

- Define your problem statement
- Brainstorm ideas
- Prioritize ideas

**Brainstorming session**

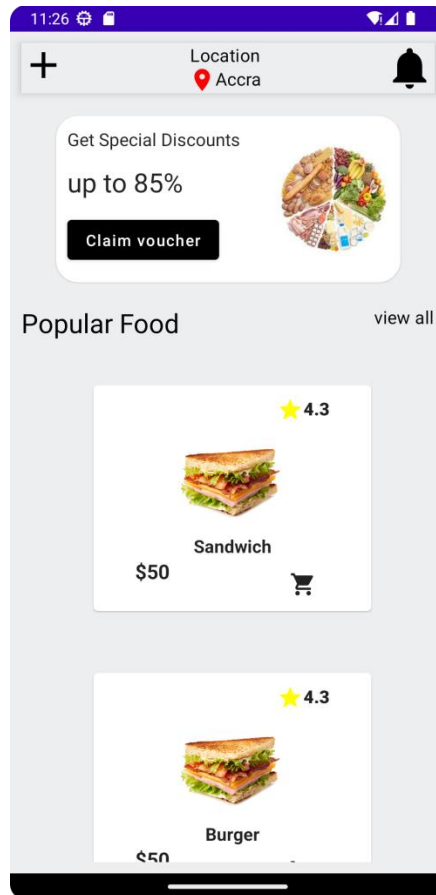
- Define your problem statement
- Brainstorm ideas
- Prioritize ideas

**After you brainstorm**

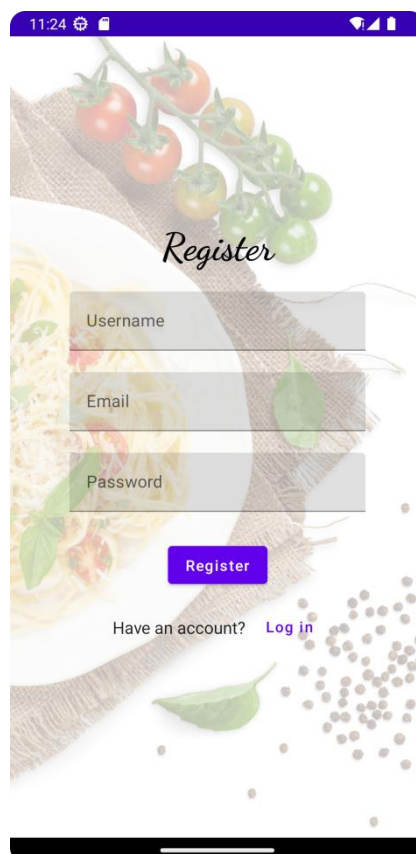
- Define your problem statement
- Brainstorm ideas
- Prioritize ideas

### 3 RESULT:

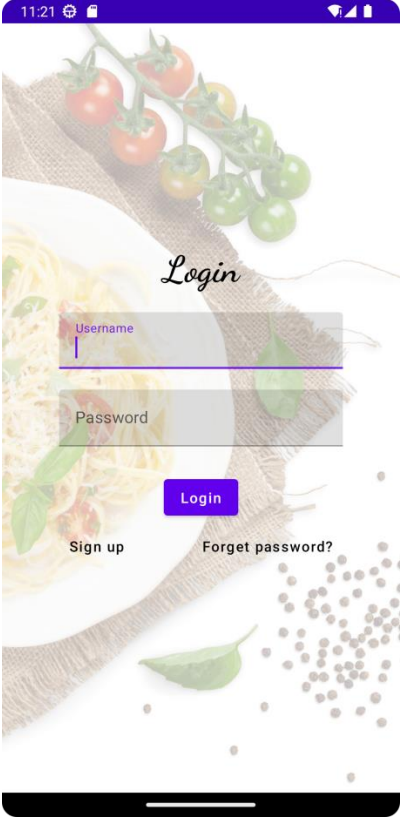
#### Home Page



#### Register Page



## Login Page

A mobile app login screen with a background image of a plate of spaghetti topped with cherry tomatoes and basil leaves. The screen features a purple status bar at the top with the time 11:21 and various icons. The word "Login" is written in a cursive font. Below it are two input fields: "Username" and "Password". A purple "Login" button is positioned below the password field. At the bottom, there are two links: "Sign up" and "Forget password?".

11:21

*Login*

Username

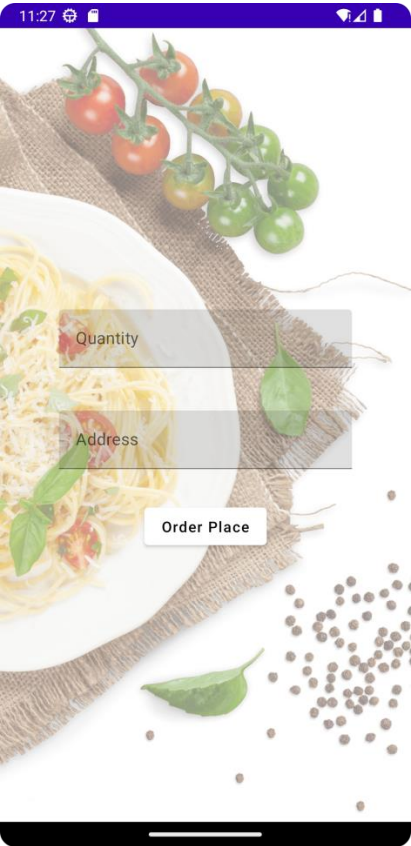
Password

Login

Sign up

Forget password?

## Order Page

A mobile app order screen with the same spaghetti background as the login page. It features a purple status bar at the top with the time 11:27. The screen has two input fields: "Quantity" and "Address". Below these fields is a white "Order Place" button.

11:27

Quantity

Address

Order Place

## **4 ADVANTAGES AND DISADVANTAGES:**

### **4.1 Advantages**

- **Convenience:** The application provides users with the convenience of ordering snacks from anywhere at any time, eliminating the need to physically visit stores or make phone calls.
- **Wide Variety of Snacks:** Users have access to a diverse range of snacks, catering to different tastes and preferences. The application can offer a comprehensive selection, including savory, sweet, healthy, and specialty snacks.
- **Personalized Recommendations:** The application can utilize user data and preferences to offer personalized snack recommendations, enhancing the user experience and introducing users to new snacks they may enjoy.
- **Dietary Filters:** The inclusion of dietary filters allows users to easily find snacks that align with their specific dietary restrictions or preferences, such as vegan, gluten-free, or nut-free options.

### **4.2 Disadvantages**

- **Technical Issues:** The application may encounter technical issues such as crashes, bugs, or slow performance, which can impact the user experience and frustrate users.
- **Reliance on Internet Connectivity:** Users require a stable internet connection to use the application effectively. Poor connectivity or network issues can disrupt the ordering process.
- **Potential Security Risks:** As the application deals with personal and payment information, there is a risk of security breaches or unauthorized access. It is crucial to implement robust security measures to protect user data.
- **Dependence on Delivery Services:** The application relies on third-party delivery services to fulfill orders. Delays or issues with delivery services can affect the overall user experience and satisfaction.

## 5 APPLICATION:

- **Home Ordering:** The application is well-suited for users who want to order snacks from the comfort of their homes. Whether it's for a movie night, a small gathering, or simply satisfying personal cravings, users can browse and order snacks without leaving their houses.
- **Office or Workplace:** Employees can use the application to conveniently order snacks for themselves or their teams during work hours. It can be a great option for office celebrations, meetings, or boosting morale by providing a variety of snacks for the workplace.
- **Events and Parties:** The application can be utilized for ordering snacks for events and parties. Users can explore the snack options, place bulk orders, and ensure a variety of snacks are available to cater to different preferences and dietary needs.
- **Late-Night Snacking:** For users who have late-night cravings, the application offers a solution by allowing them to order snacks even when physical stores are closed. It provides a convenient way to satisfy hunger or indulge in midnight snacking.
- **Special Dietary Requirements:** The application's dietary filters make it applicable for individuals with specific dietary restrictions or preferences. Users can easily search for snacks that align with their requirements, such as vegan, gluten-free, or nut-free options.

## 6 CONCLUSION:

The Snack Ordering Application is an innovative and convenient solution for users to order snacks from their favorite vendors.

The application will be developed using Java and Kotlin programming languages.

## 7 FUTURE SCOPE:

The future scope of the Snack Ordering Application, developed using Kotlin and Java, is vast and offers several opportunities for expansion and improvement. Here are some potential areas for future development:

- **Integration with Restaurants and Food Services:** The application can be expanded to include partnerships with local restaurants and food services. This would allow users to order not only snacks but also meals, providing a wider range of options and catering to different dining preferences.
- **Social Features:** Introducing social features within the application can enhance the user experience. Users could have the option to create profiles, follow friends, and share their snack orders or recommendations on social media platforms, fostering a sense of community and encouraging user engagement.
- **Augmented Reality (AR) Integration:** Implementing AR technology within the application could allow users to visualize snacks in their real environment. By leveraging AR, users can see how a snack might look or fit within their surroundings before placing an order, enhancing the decision-making process.
- **Gamification and Loyalty Programs:** Adding gamification elements, such as earning points or rewards for frequent orders or referrals, can incentivize user engagement and create a sense of loyalty. Loyalty programs can be introduced to offer exclusive discounts, personalized offers, or early access to new snacks.
- **Voice Assistant Integration:** Integrating voice assistant technology, such as Amazon Alexa or Google Assistant, can provide users with a hands-free and seamless ordering experience. Users can place orders, inquire about snack recommendations, or track their orders using voice commands.

# 8 APPENDIX:

## 8.1 Source Code

### AdminActivity.kt

```
package com.example.snackordering

import android.icu.text.SimpleDateFormat
import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
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.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.example.snackordering.ui.theme.SnackOrderingTheme
import java.util.*

class AdminActivity : ComponentActivity() {
```



```

private lateinit var orderDatabaseHelper: OrderDatabaseHelper
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    orderDatabaseHelper = OrderDatabaseHelper(this)
    setContent {
        SnackOrderingTheme {
            // A surface container using the 'background' color from
the theme
            Surface(
                modifier = Modifier.fillMaxSize(),
                color = MaterialTheme.colors.background
            ) {
                val data=orderDatabaseHelper.getAllOrders();
                Log.d("swathi" ,data.toString())
                val order = orderDatabaseHelper.getAllOrders()
                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}")
                }
            }
        }
    }
}

```

## LoginActivity.kt

```
package com.example.snackordering

import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat
import com.example.snackordering.ui.theme.SnackOrderingTheme

class LoginActivity : ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = UserDatabaseHelper(this)
    }
}
```

```

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

```

```

        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,
            onChange = { username = it },
            label = { Text("Username") },
            modifier = Modifier.padding(10.dp)
                .width(280.dp)
        )

        TextField(
            value = password,
            onChange = { 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.getUserByUsername(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.Black,text = "Sign up") }
    TextButton(onClick = {
    })

    {
        Spacer(modifier = Modifier.width(60.dp))
        Text(color = Color.Black,text = "Forget password?")
    }
}
}
}

```

```
private fun startMainPage(context: Context) {  
    val intent = Intent(context, MainPage::class.java)  
    ContextCompat.startActivity(context, intent, null)  
}
```

## **RegisterActivity.kt**

```
package com.example.snackordering
```

```
import android.content.Context  
import android.content.Intent  
import android.os.Bundle  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.compose.foundation.Image  
import androidx.compose.foundation.layout.*  
import androidx.compose.material.*  
import androidx.compose.runtime.*  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.layout.ContentScale  
import androidx.compose.ui.res.painterResource  
import androidx.compose.ui.text.font.FontFamily  
import androidx.compose.ui.text.font.FontWeight  
import androidx.compose.ui.unit.dp  
import androidx.compose.ui.unit.sp  
import androidx.core.content.ContextCompat  
import com.example.snackordering.ui.theme.SnackOrderingTheme
```



```

class MainActivity : ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = UserDatabaseHelper(this)
        setContent {
            SnackOrderingTheme {
                // A surface container using the 'background' color from
the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colors.background
                ) {

                    RegistrationScreen(this, databaseHelper)

                }
            }
        }
    }
}

```

@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.CenterHorizontally,  
    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,  
    onChange = { email = it },  
    label = { Text("Email") },  
    modifier = Modifier  
        .padding(10.dp)  
        .width(280.dp)  
)
```

```
TextField(  
    value = password,  
    onChange = { 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(user)
            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)  
}
```

## TargetActivity.kt

```
package com.example.snackordering
```

```
import android.content.Context
```

```
import android.content.Intent
```

```
import android.os.Bundle
```

```
import android.util.Log
```

```
import android.widget.Toast
```

```
import androidx.activity.ComponentActivity
```

```
import androidx.activity.compose.setContent
```

```
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.Alignment
```

```
import androidx.compose.ui.Modifier
```

```
import androidx.compose.ui.graphics.Color
```

```
import androidx.compose.ui.layout.ContentScale
```

```
import androidx.compose.ui.platform.LocalContext
```

```
import androidx.compose.ui.platform.textInputServiceFactory
```

```
import androidx.compose.ui.res.painterResource
```

```
import androidx.compose.ui.text.input.KeyboardType
```

```
import androidx.compose.ui.tooling.preview.Preview
```

```
import androidx.compose.ui.unit.dp
```

```
import androidx.core.content.ContextCompat
```

```
import com.example.snackordering.ui.theme.SnackOrderingTheme
```

```

class TargetActivity : ComponentActivity() {
    private lateinit var orderDatabaseHelper: OrderDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        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.getAllOrders()
                    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.CenterHorizontally,
    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(keyboardType =
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.insertOrder(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)
}

```

## Order.kt

```
package com.example.snackordering

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

@Entity(tableName = "order_table")
data class Order(
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "quantity") val quantity: String?,
    @ColumnInfo(name = "address") val address: String?,
)
```

## OrderDao.kt

```
package com.example.snackordering

import androidx.room.*

@Dao
interface OrderDao {
    @Query("SELECT * FROM order_table WHERE address= :address")
    suspend fun getOrderByAddress(address: String): Order?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertOrder(order: Order)

    @Update
    suspend fun updateOrder(order: Order)

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

## OrderDatabase.kt

```
package com.example.snackordering

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@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.applicationContext,
                    OrderDatabase::class.java,
                    "order_database"
                ).build()
                instance = newInstance
                newInstance
            }
        }
    }
}
```

## OrderDatabaseHelper.kt

```
package com.example.snackordering

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

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()
}
```

```
@SuppressWarnings("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.getColumnIndex(COLUMN_ID)),
            quantity =
cursor.getString(cursor.getColumnIndex(COLUMN_QUANTITY)),

```

```

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

```

## User.kt

```
package com.example.snackordering
```

```

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

```

```

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

```

```
@PrimaryKey(autoGenerate = 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?,  
  
)
```

## UserDao.kt

```
package com.example.snackordering  
  
import androidx.room.*  
  
@Dao  
interface UserDao {  
  
    @Query("SELECT * FROM user_table WHERE email = :email")  
    suspend fun getUserByEmail(email: String): User?  
  
    @Insert(onConflict = OnConflictStrategy.REPLACE)  
    suspend fun insertUser(user: User)  
  
    @Update  
    suspend fun updateUser(user: User)  
  
    @Delete  
    suspend fun deleteUser(user: User)  
  
}
```



## UserDatabase.kt

```
package com.example.snackordering

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@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.applicationContext,
                    UserDatabase::class.java,
                    "user_database"
                ).build()
                instance = newInstance
                newInstance
            }
        }
    }
}
```

## UserDataBaseHelper.kt

```
package com.example.snackordering

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

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_NAME, 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()  
}
```

```
@SuppressWarnings("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.getColumnIndex(COLUMN_ID)),
            firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
            lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
            email =
cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
            password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
        )}

        cursor.close()
        db.close()
        return user
    }

    @SuppressWarnings("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.getColumnIndex(COLUMN_ID)),
                firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email =
cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password =
cursor.getString(cursor.getColumnIndex(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.getColumnIndex(COLUMN_ID)),
                firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email =
cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )

            users.add(user)

        } while (cursor.moveToNext())
    }

    cursor.close()

    db.close()

    return users
}
}
}

```

## MainPage.kt

```
package com.example.snackordering

import android.annotation.SuppressLint
import android.content.Context
import android.os.Bundle
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.annotation.DrawableRes
import androidx.annotation.StringRes
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.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.Text
import androidx.compose.ui.unit.dp
import androidx.compose.ui.graphics.RectangleShape
import androidx.compose.ui.layout.ContentScale
```

```

import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat.startActivity
import com.example.snackordering.ui.theme.SnackOrderingTheme

import android.content.Intent as Intent1

class MainPage : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            SnackOrderingTheme {
                // A surface container using the 'background' color from
the theme

                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colors.background
                ) {
                    FinalView(this)
                    val context = LocalContext.current
                    //PopularFoodColumn(context)
                }
            }
        }
    }
}

```

```
@Composable
```

```
fun TopPart() {
```

```
    Row(
```

```
        modifier = Modifier
```

```
            .fillMaxWidth()
```

```
            .background(Color(0xffeceef0)), Arrangement.SpaceBetween
```

```
    ) {
```

```
        Icon(
```

```
            imageVector = Icons.Default.Add, contentDescription = "Menu
```

```
Icon",
```

```
            Modifier
```

```
                .clip(CircleShape)
```

```
                .size(40.dp),
```

```
                tint = Color.Black,
```

```
        )
```

```
        Column(horizontalAlignment = Alignment.CenterHorizontally) {
```

```
            Text(text = "Location", style =
```

```
MaterialTheme.typography.subtitle1, 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.surface)

            }

        }

        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.CenterHorizontally
        ) {
            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(intent)

    }) {

        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(0xffeceef0))
            .padding(10.dp),
        verticalArrangement = Arrangement.Top,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        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.subtitle1, 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)
            }
        },
        verticalArrangement = Arrangement.spacedBy(16.dp))
}

```

```

@SuppressLint("UnusedMaterialScaffoldPaddingParameter")

```

@Composable

```

fun FinalView(mainPage: MainPage) {
    SnackOrderingTheme {
        Scaffold() {
            val context = LocalContext.current
            App(context)
        }
    }
}

```