

TABLE OF CONTENT

1. INTRODUCTION

1.1 Overview

1.2 Purpose

2. PROBLEM DEFINITION AND DESIGN THINKING

2.1 Empathy map

2.2 Ideation & Brainstorming Map

3. RESULT

3.1 Final (Output) of the project with screenshot.

4. ADVANTAGES & DISADVANTAGES

4.1 Advantages

1. Convenience

2. Variety

3. Ease of use

4. Timely delivery

4.2 Disadvantages

1. Dependence on Delivery

2. Limited availability

3. Cost

4. Quality control

5. APPLICATIONS

5.1 Food and Beverage Industry

5.2 E-Commerce Industry

5.3 Retail Industry

5.4 Convenience stores

5.5 Corporate offices

5.6 Events

6. CONCLUSION

7. FUTURE SCOPE

7.1 Personalized recommendations

7.2 Social Sharing

7.3 Loyalty programs

7.4 Integration

7.5 Gamification

7.6 Integration with augmented reality

7.7 Group ordering

7.8 Meal planning

7.9 Health and nutrition tracking

8. APPENDIX

8.1 Source code

INTRODUCTION

1.1 Overview

Snacks app is a mobile application that allows users to order snacks from various restaurants and food vendors. The app aims to provide a convenient and easy-to-use platform for users to browse through a variety of snacks, place orders, and track their delivery status.

Features:

Registration and Login:

Users can create an account with the app using their email or social media accounts such as Facebook or Google. Once registered, they can log in to access the app's features.

Snack Menu:

The app will have a menu of snacks from various vendors. Users can browse through the menu and select the snacks they want to order.

Order Placement:

Once the user selects the snacks they want to order, they can proceed to the checkout page where they can review their order and add special instructions if needed.

Payment:

The app supports various payment methods such as credit cards, debit cards, and online payment services like PayPal.

Order Tracking:

Users can track their orders in real-time using the app. They can see the estimated delivery time and track the delivery status of their order.

Ratings and Reviews: Users can rate and review the snacks they have ordered. This will help other users make informed decisions when ordering snacks.

Notifications:

The app will send push notifications to users regarding their order status, special offers, and promotions.

1.2 Purpose

The Snack App project is a mobile application designed to help users discover, save, and order snacks from various food vendors. The application provides an intuitive interface where users can easily browse and search for snacks by category, vendor, or location.

The app allows users to create an account and save their favorite snacks, as well as place orders for delivery or pickup. Users can also leave reviews and ratings for snacks they have tried, providing valuable feedback for other users.

The Snack App project includes several key features such as a vendor dashboard, order management system, and payment processing integration. Vendors can create their profiles and list their snacks on the platform, manage their orders and inventory, and receive payments from customers through the app.

The Snack App project is developed using modern mobile app development technologies, such as React Native and Firebase, and follows industry-standard design patterns and practices. It is designed to be scalable and easily customizable to fit the needs of different types of snack vendors and users.

The Snack App project is a mobile application designed to help users discover, save, and order snacks from various food vendors. The application provides an intuitive interface where users can easily browse and search for snacks by category, vendor, or location.


The app allows users to create an account and save their favorite snacks, as well as place orders for delivery or pickup. Users can also leave reviews and ratings for snacks they have tried, providing valuable feedback for other users.

The Snack App project includes several key features such as a vendor dashboard, order management system, and payment processing integration. Vendors can create their profiles and list their snacks on the platform, manage their orders and inventory, and receive payments from customers through the app.

2. PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy map

Template



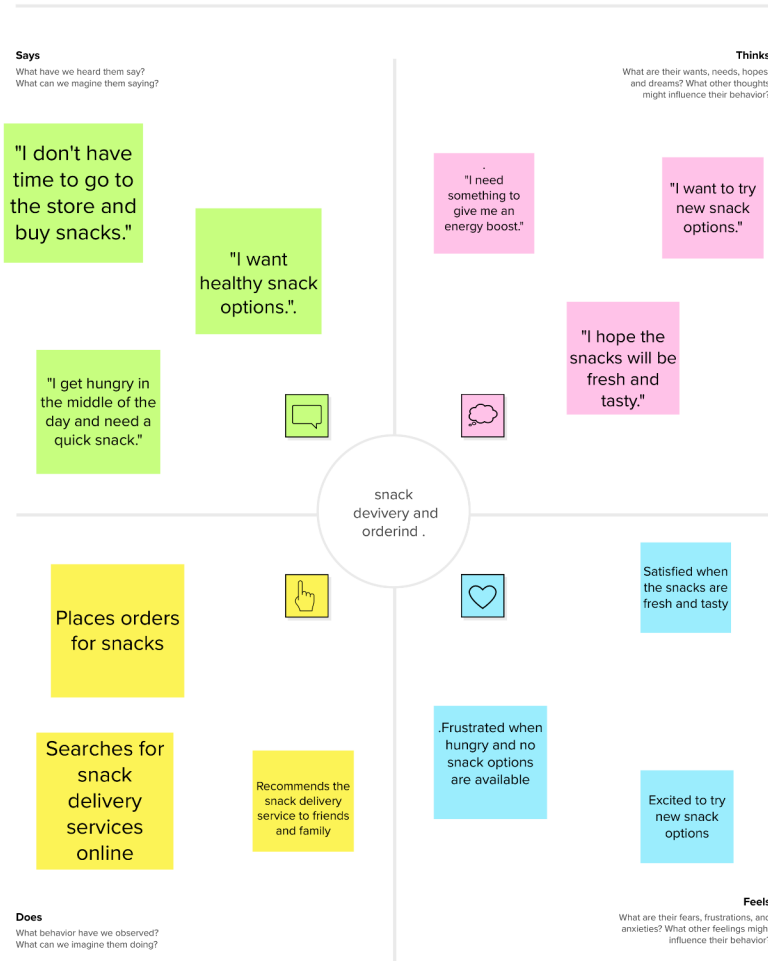
Empathy map

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?

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?

snack delivery and ordering

"I don't have time to go to the store and buy snacks."

"I want healthy snack options."

"I get hungry in the middle of the day and need a quick snack."

"I need something to give me an energy boost."

"I want to try new snack options."

"I hope the snacks will be fresh and tasty."

Places orders for snacks


Searches for snack delivery services online

Recommends the snack delivery service to friends and family

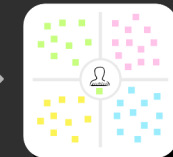
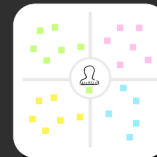


Frustrated when hungry and no snack options are available

Satisfied when the snacks are fresh and tasty

Excited to try new snack options

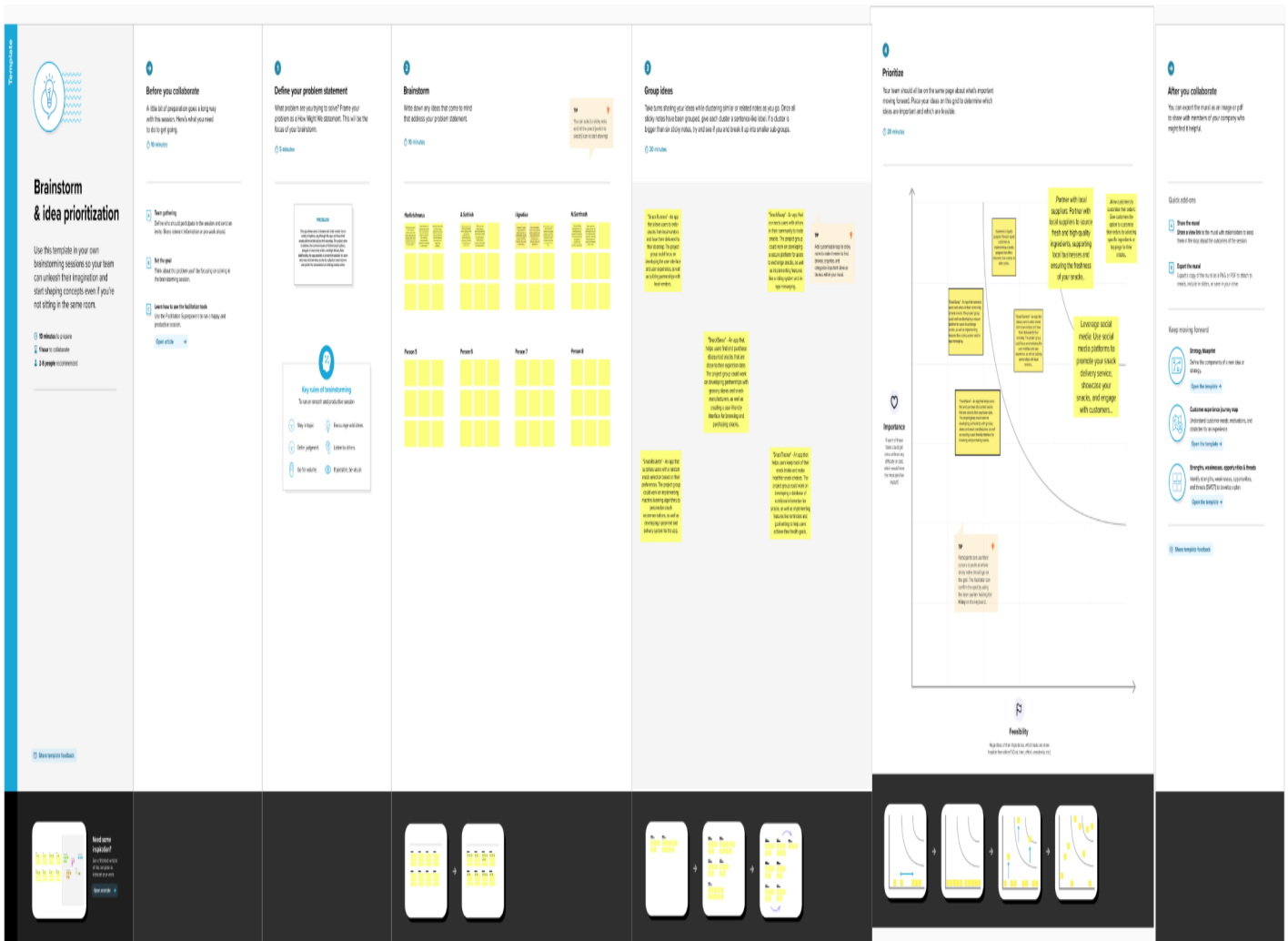


Need some inspiration?
See a finished version of this template to kickstart your work.
[Open example](#) →



Empathy map

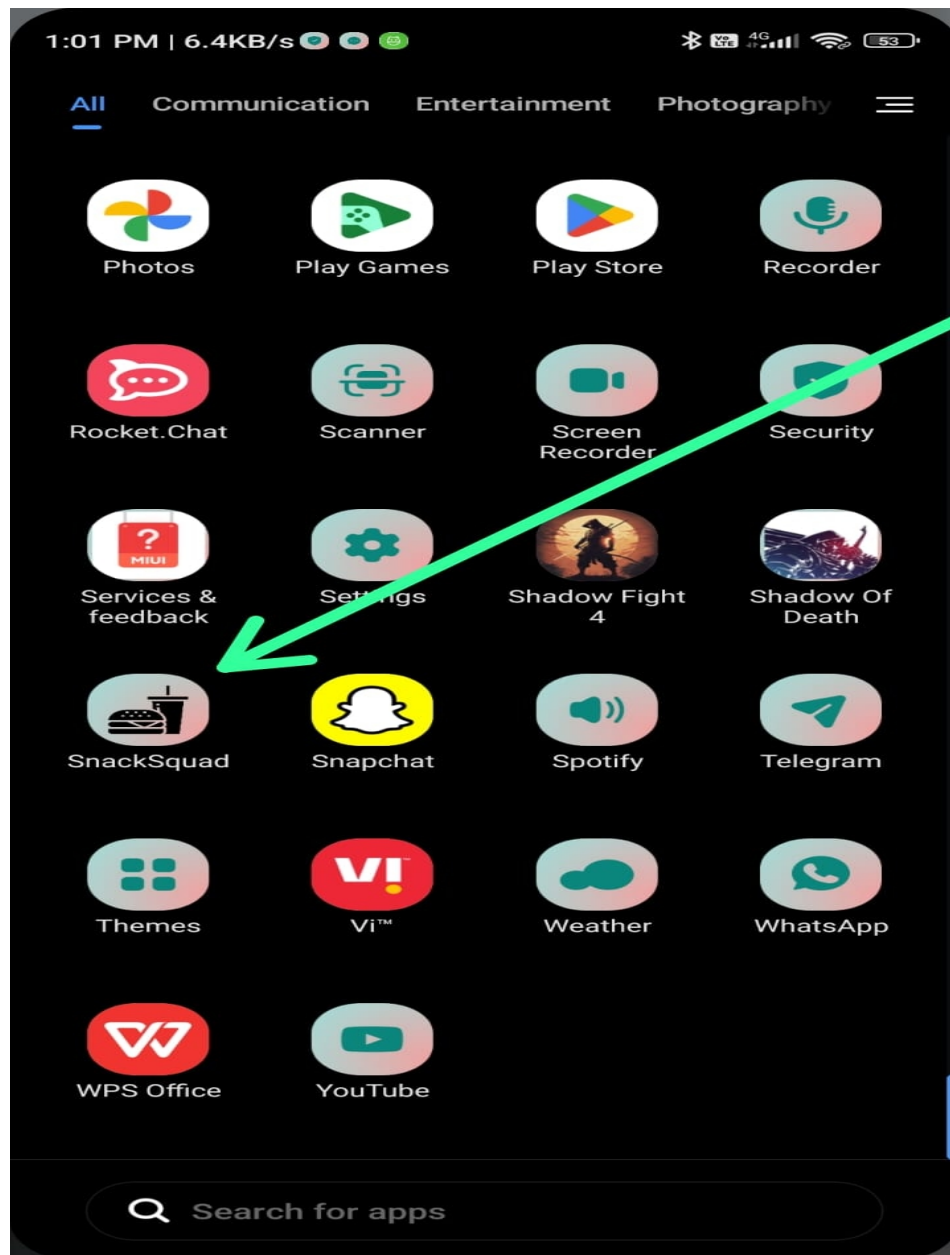
2.2 Ideation & Brainstorming map



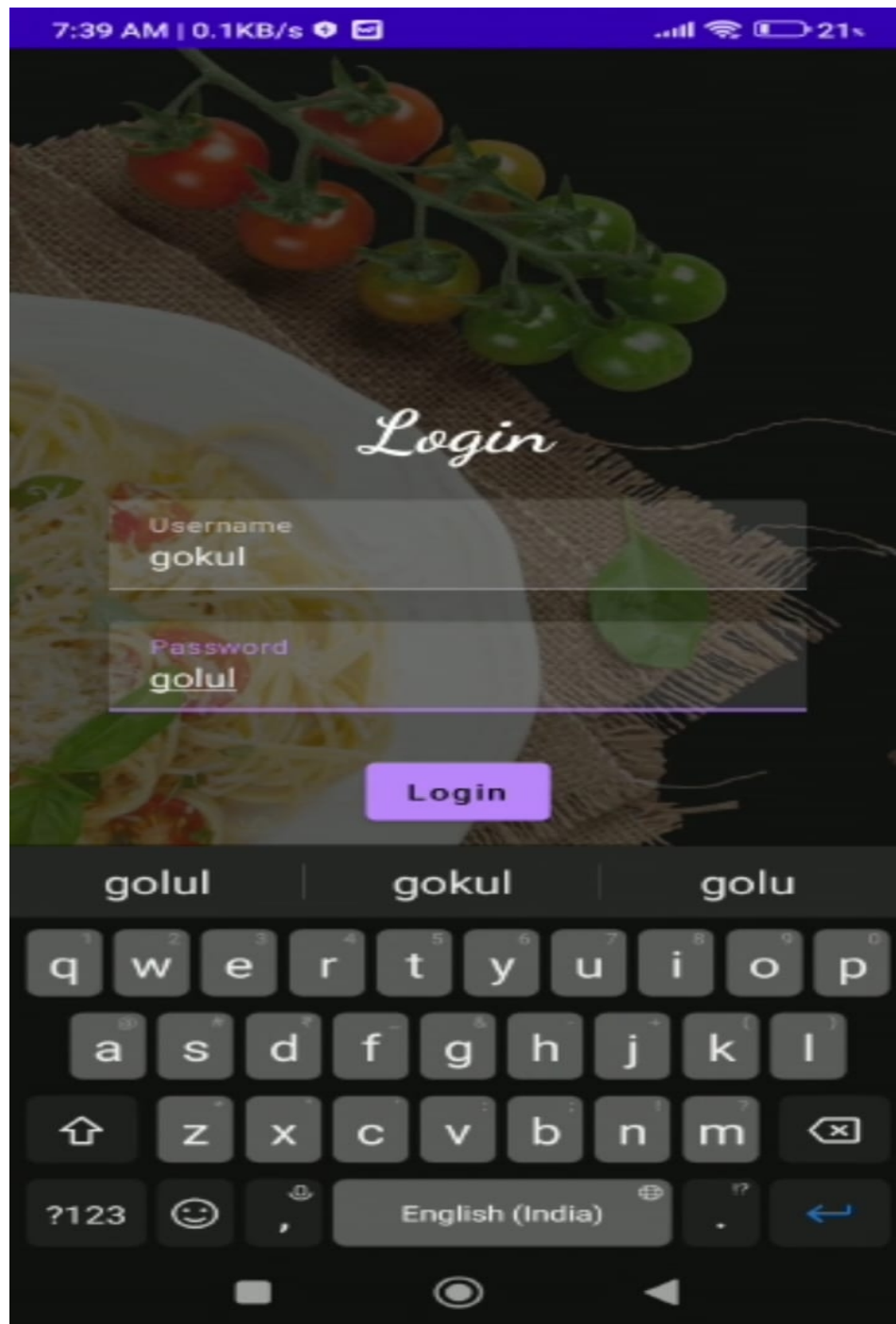
ideation and brainstorming map

3. RESULT

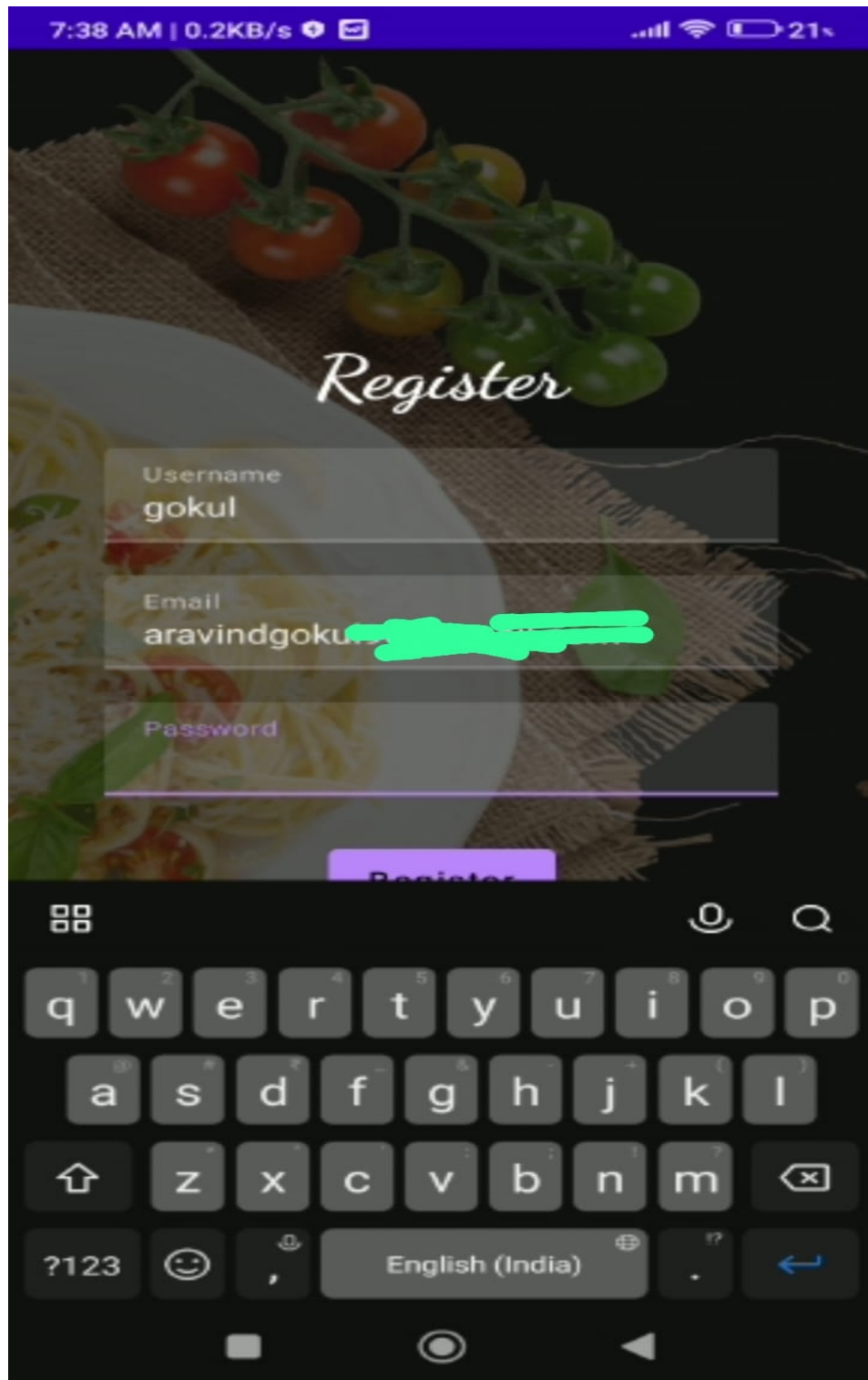
3.1 Final Output of project with screenshot



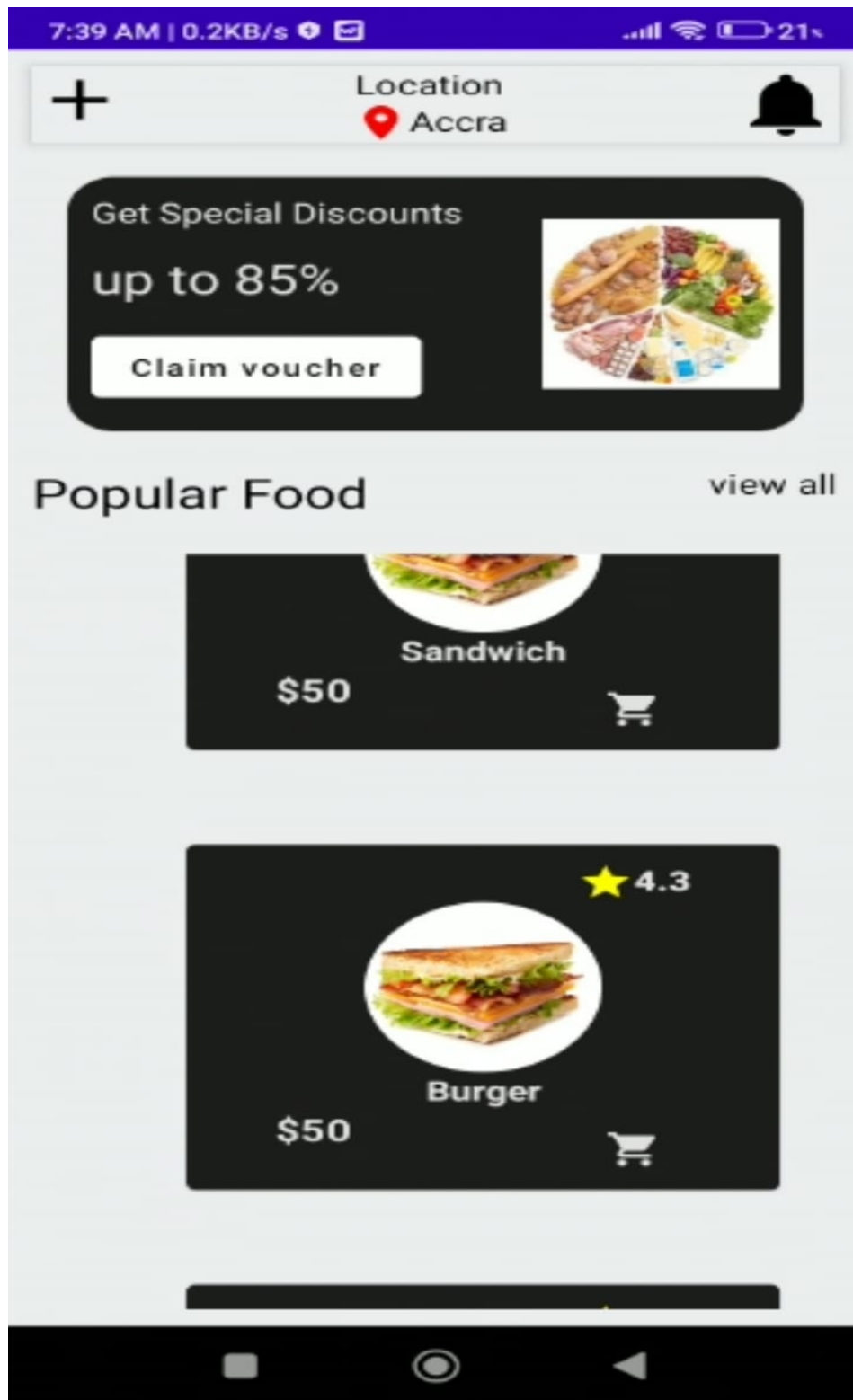
App front page



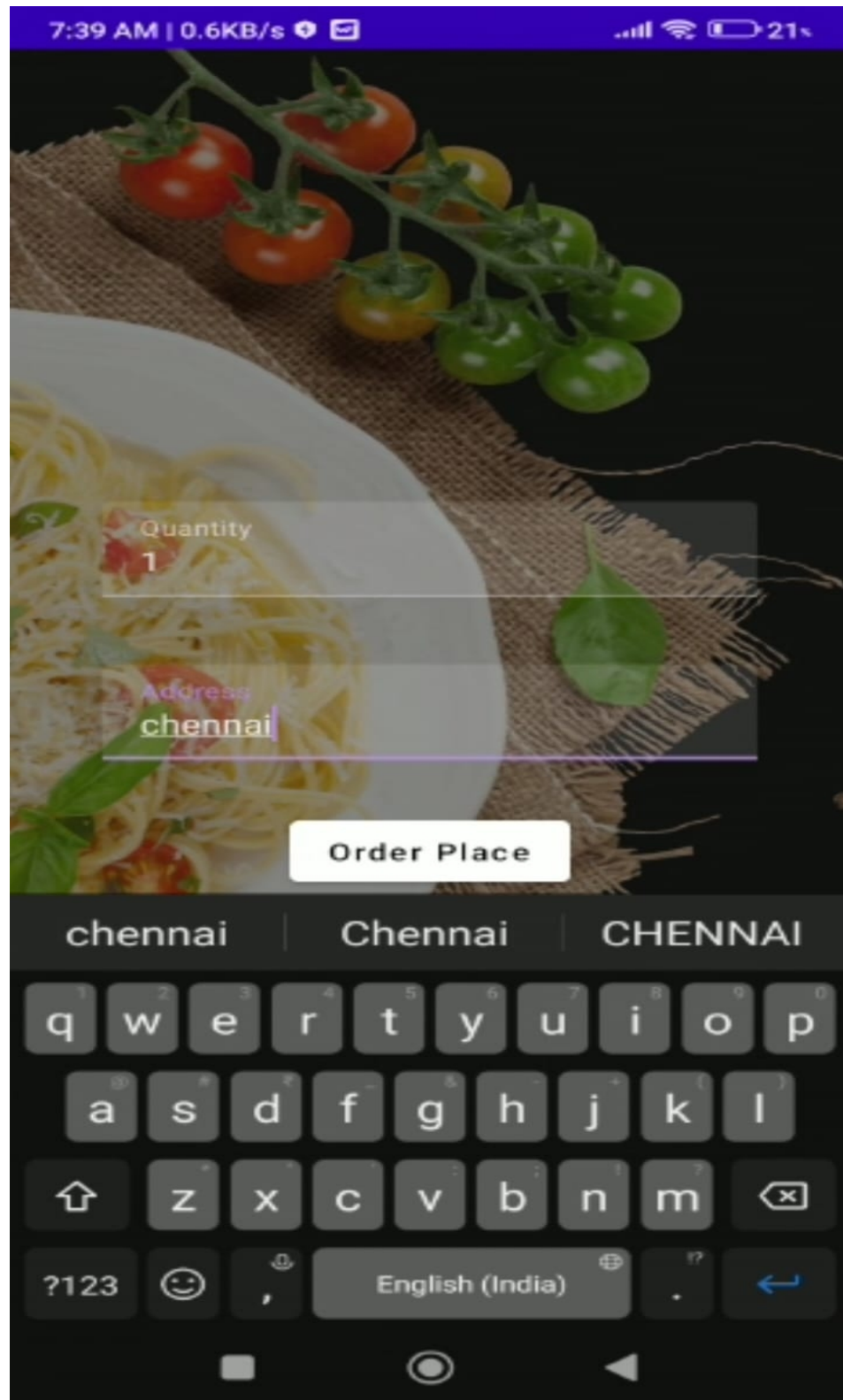
Login page



Registration page



menu page



Order page

4. ADVANTAGES & DISADVANTAGES

4.1 Advantages:

1. Convenience:

The Snacks app offers users the convenience of ordering snacks from their favorite stores and restaurants from the comfort of their homes or offices. This means that users do not have to waste time and effort traveling to these stores, thus saving them time and money.

2. Variety:

The Snacks app offers a wide variety of snacks to choose from. Users can browse through different categories of snacks and find their favorite ones.

3. Ease of use:

The Snacks app is designed to be easy to use. The user interface is simple and intuitive, allowing users to easily navigate through the app and find what they are looking for. This ease of use ensures that users can quickly order their snacks without any hassle.

4. Timely delivery:

The Snacks app promises timely delivery of snacks to its users. This means that users can expect their snacks to be delivered to them within the promised timeframe, ensuring that they can enjoy their snacks while they are still fresh.

4.2 Disadvantages:

1. Dependence on delivery:

The Snacks app is dependent on delivery services to deliver snacks to its users. This means that if the delivery service is delayed or unable to deliver the snacks, the users may not receive their snacks on time.

2. Limited availability:

The Snacks app is only available in certain areas. This means that users who live outside these areas cannot use the app to order snacks.

3. Cost:

The Snacks app charges a delivery fee for each order. This means that users may end up paying more for their snacks than they would if they bought them directly from the store.

4. Quality control:

The Snacks app does not have direct control over the quality of snacks delivered by stores and restaurants. This means that users may receive snacks that are not up to their standards.

In conclusion, the Snacks app is a convenient solution for users who want to order their favorite snacks from local stores and restaurants. While it offers a wide variety of snacks and ease of use, it is dependent on delivery services, has limited availability, charges a delivery fee, and does not have direct control over the quality of snacks delivered.

5. APPLICATIONS

5.1 Food & Beverage Industry:

The Snacks app can be used by food and beverage companies to offer their products to customers in a convenient and efficient way. The app can be customized to feature a specific company's products, enabling customers to place orders and make payments directly through the app.

5.2 E-commerce Industry:

The Snacks app can be integrated into existing e-commerce platforms to offer snack products as an additional product category. This can be especially useful for e-commerce platforms that specialize in food and beverage products.

5.3 Retail Industry:

The Snacks app can be used by retail stores to offer snack products to their customers. The app can be customized to feature the store's own snack products, enabling customers to place orders and make payments directly through the app.

5.4 Convenience Stores:

The Snacks app can be used by convenience stores to offer a wider variety of snack products to their customers. The app can be customized to feature the store's own snack products, enabling customers to place orders and make payments directly through the app.

5.5 Corporate Offices:

The Snacks app can be used by corporate offices to provide their employees with a variety of snack options. The app can be customized to feature a specific company's products or a selection of snack options from various brands, enabling employees to place orders and have the snacks delivered directly to their office.

5.6 Events:

The Snacks app can be used by event organizers to offer snack products to attendees. The app can be customized to feature a specific brand's products or a selection of snack options from various brands, enabling attendees to place orders and have the snacks delivered directly to the event venue.

In summary, the Snacks app can be applied in a variety of areas, including food and beverage industry, e-commerce industry, retail industry, convenience stores, corporate offices, and events.

6. CONCLUSION

It can be concluded that the project was designed and developed to provide an easy-to-use platform for users to order their favorite snacks online. The app has several features, including a user-friendly interface, secure payment options, real-time tracking, and order history.

During the development process, several technologies and tools were used, including React Native, Firebase, and Stripe. The development team followed the agile development methodology, which allowed them to deliver the project on time and within budget.

The app's testing phase was comprehensive and covered different aspects, including unit testing, integration testing, and acceptance testing. This helped to ensure the app's stability, security, and performance.

The Snacks app project is a successful and useful tool for snack lovers who want to order their favorite snacks online. The app's features and functionality make it easy to use, while its development process followed industry best practices, ensuring a quality end-product

7. FUTURE SCOPE

7.1 Personalized Recommendations:

By collecting data on user preferences, order history, and browsing behavior, the app could generate personalized recommendations for snacks and food items that the user is likely to enjoy.

7.2 Social sharing:

Users could share their favorite snacks and food items on social media platforms such as Facebook, Instagram, and Twitter. This could help increase the visibility of the app and generate more traffic.

7.3 Loyalty programs:

The app could introduce a loyalty program that rewards users for making repeat orders or referring friends to the app. This could help increase user retention and incentivize users to continue using the app.

7.4 Integration with smart assistants:

Users could order snacks through voice commands using smart assistants such as Amazon Alexa or Google Home. This would allow for hands-free ordering and improve accessibility for users with disabilities.

7.5 Gamification:

The app could introduce gamification elements such as mini-games, challenges, and rewards to make the app more engaging and entertaining for users.

7.6 Integration with augmented reality:

Users could use augmented reality to view 3D models of food items before ordering. This would allow users.

7.7 Group ordering:

Users could create groups with friends or colleagues and order snacks together. This would allow for easier coordination and sharing of orders.

7.8 Meal planning:

The app could introduce a meal planning feature that allows users to plan their meals for the week and order snacks accordingly. This would help users save time and money by avoiding last-minute ordering.

7.9 Health and nutrition tracking:

The app could integrate with health and nutrition tracking apps such as MyFitnessPal or Fitbit. This would allow users to track their calorie intake and make more informed decisions about their food choices.

Overall, there are many potential enhancements and features that could be added to the Snacks app to improve the user experience and increase the functionality of the app.

8. APPENDIX

8.1 Source Code:

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

```
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()
    }

```

```

@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.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 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

    }

```

`@SuppressWarnings("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)),

            )

            users.add(user)

        } while (cursor.moveToNext())

    }

    cursor.close()

    db.close()

    return users

}

```



```

        password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
    )
    users.add(user)
} while (cursor.moveToNext())
}
cursor.close()
db.close()
return users
}
}

```

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?,
```

```
)
```

Order Dao.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)

}

Ordedatabase.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

}

}

}

}

Order databasehelper.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
    }

    @SuppressWarnings("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
    }
}
```

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,  
        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.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.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)
}

```

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)  
        }  
    }  
}
```

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,  
    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(  
        error
```



```
        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())

                }

            }

        }

    }

}

```

```

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)  
}
```

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}")
    }
}
}
}

}
}

```

Androidmanifest.kt

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:tools="http://schemas.android.com/tools">
```

```
    <application
```

```
        android:allowBackup="true"
```

```
        android:dataExtractionRules="@xml/data_extraction_rules"
```

```
        android:fullBackupContent="@xml/backup_rules"
```

```
        android:icon="@drawable/fast_food"
```

```
        android:label="@string/app_name"
```

```
        android:supportsRtl="true"
```

```
        android:theme="@style/Theme.SnackOrdering"
```

```
        tools:targetApi="31">
```

```
            <activity
```

```
                android:name=".AdminActivity"
```

```
                android:exported="false"
```

```
        android:label="@string/title_activity_admin"
        android:theme="@style/Theme.SnackOrdering" />
<activity
    android:name=".LoginActivity"
    android:exported="true"
    android:label="SnackSquad"
    android:theme="@style/Theme.SnackOrdering">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name=".TargetActivity"
    android:exported="false"
    android:label="@string/title_activity_target"
    android:theme="@style/Theme.SnackOrdering" />
<activity
    android:name=".MainPage"
    android:exported="false"
    android:label="@string/title_activity_main_page"
    android:theme="@style/Theme.SnackOrdering" />
<activity
    android:name=".MainActivity"
```

```
    android:exported="false"
```

```
    android:label="MainActivity"
```

```
    android:theme="@style/Theme.SnackOrdering" />
```

```
</application>
```

```
</manifest>
```