A Customizable Snack Ordering and Delivery App

Introduction

1.1overview

A customizable snack ordering and delivery app is a mobile application that enables users to order snacks and have them delivered to their location. The app is designed to provide a convenient and user-friendly experience for customers, while also providing an efficient way for businesses to manage and fulfill orders.

The app typically includes a range of features such as a user-friendly interface, a menu of available snacks, customizable order options, payment processing, and real-time delivery tracking. Customers can browse the available snacks, select the items they want, and place an order for delivery to their location.

The app also provides an interface for businesses to manage their snack inventory, update menu items and pricing, receive and fulfill orders, and track delivery status. Business owners can customize the app to match their branding, choose delivery zones, and set delivery fees and minimum order amounts.

Some customizable snack ordering and delivery apps also offer additional features such as loyalty programs, discounts, and promotions to encourage repeat business and customer engagement.

Overall, a customizable snack ordering and delivery app can provide a streamlined and efficient way for snack businesses to manage orders and provide a convenient and user-friendly experience for customers.

Regenerate response

1.2 purpose

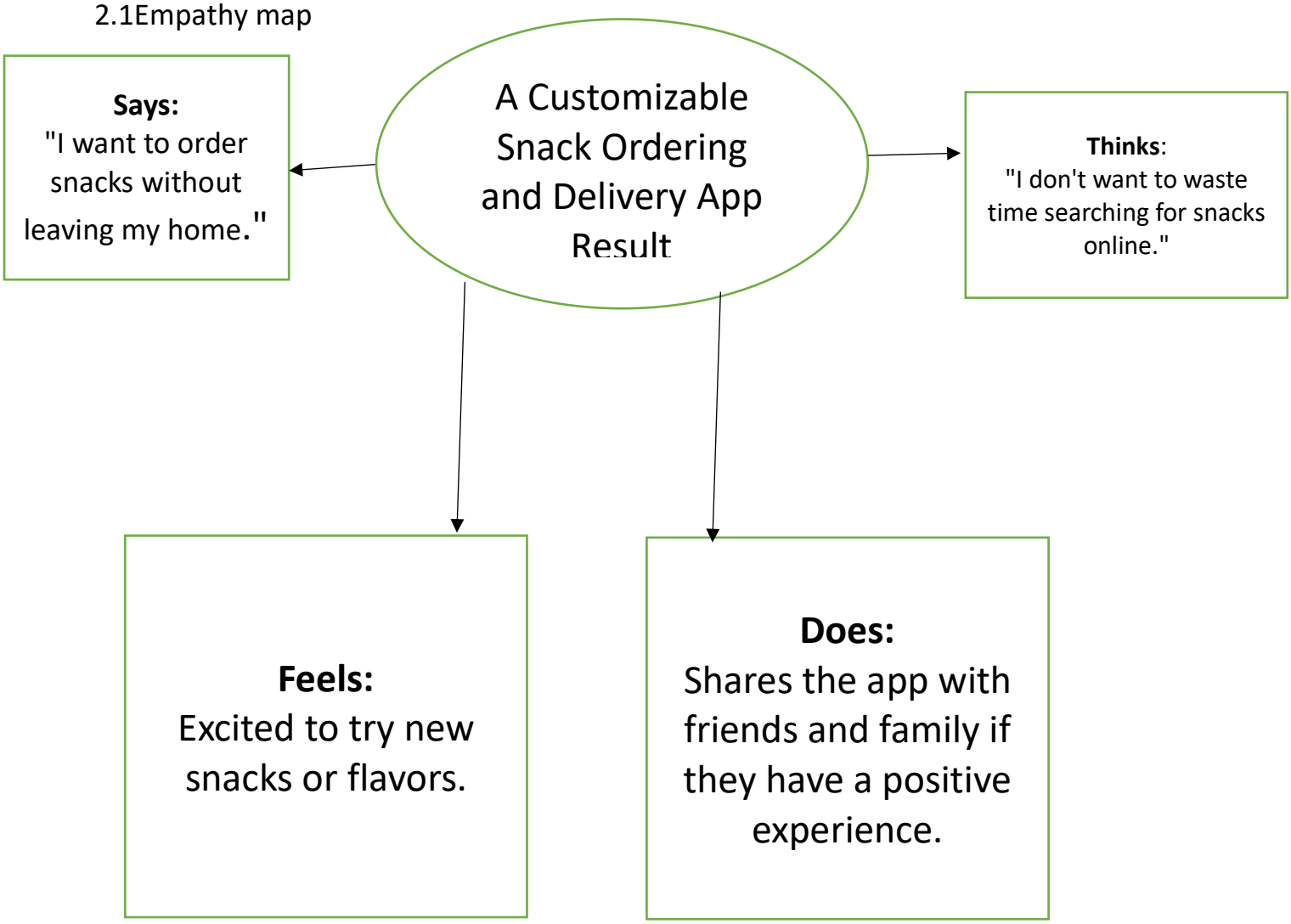
The purpose of a customizable snack ordering and delivery app is to provide a convenient and efficient way for customers to order snacks and have them delivered to their location. The app is designed to streamline the process of ordering and delivering snacks, providing a user-friendly interface and real-time delivery tracking.

For snack businesses, the app offers an efficient way to manage and fulfill orders, update inventory, and track delivery status. Customizable options such as branding, delivery zones, and fees also allow businesses to tailor the app to their specific needs and preferences.

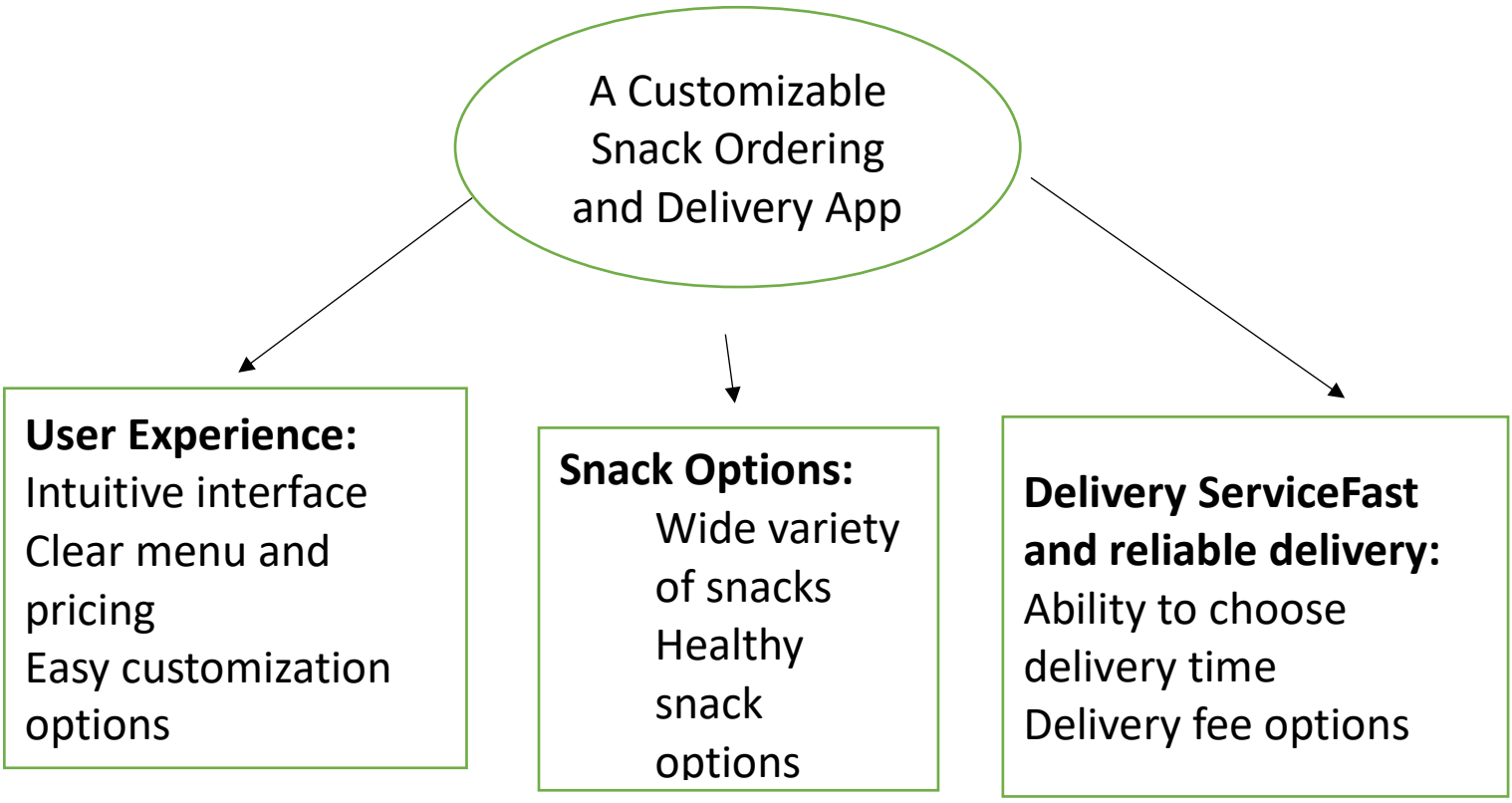
Overall, the app aims to enhance the customer experience by providing a fast and convenient way to order snacks while also improving the operational efficiency of snack businesses. By simplifying the ordering and delivery process, the app can also help to increase customer engagement and encourage repeat business.

Problem Definition&Design Thinking

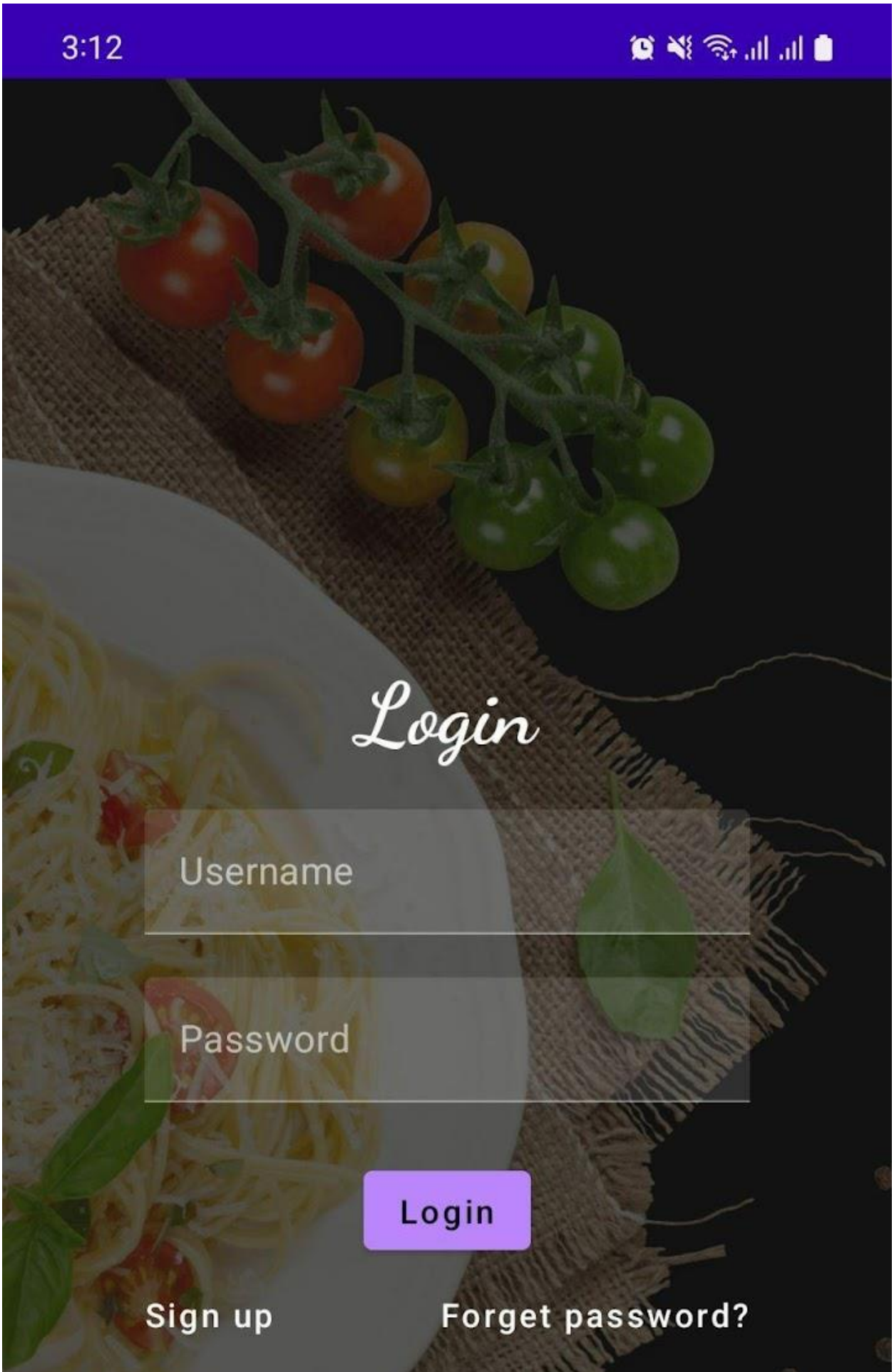
2.1Empathy map

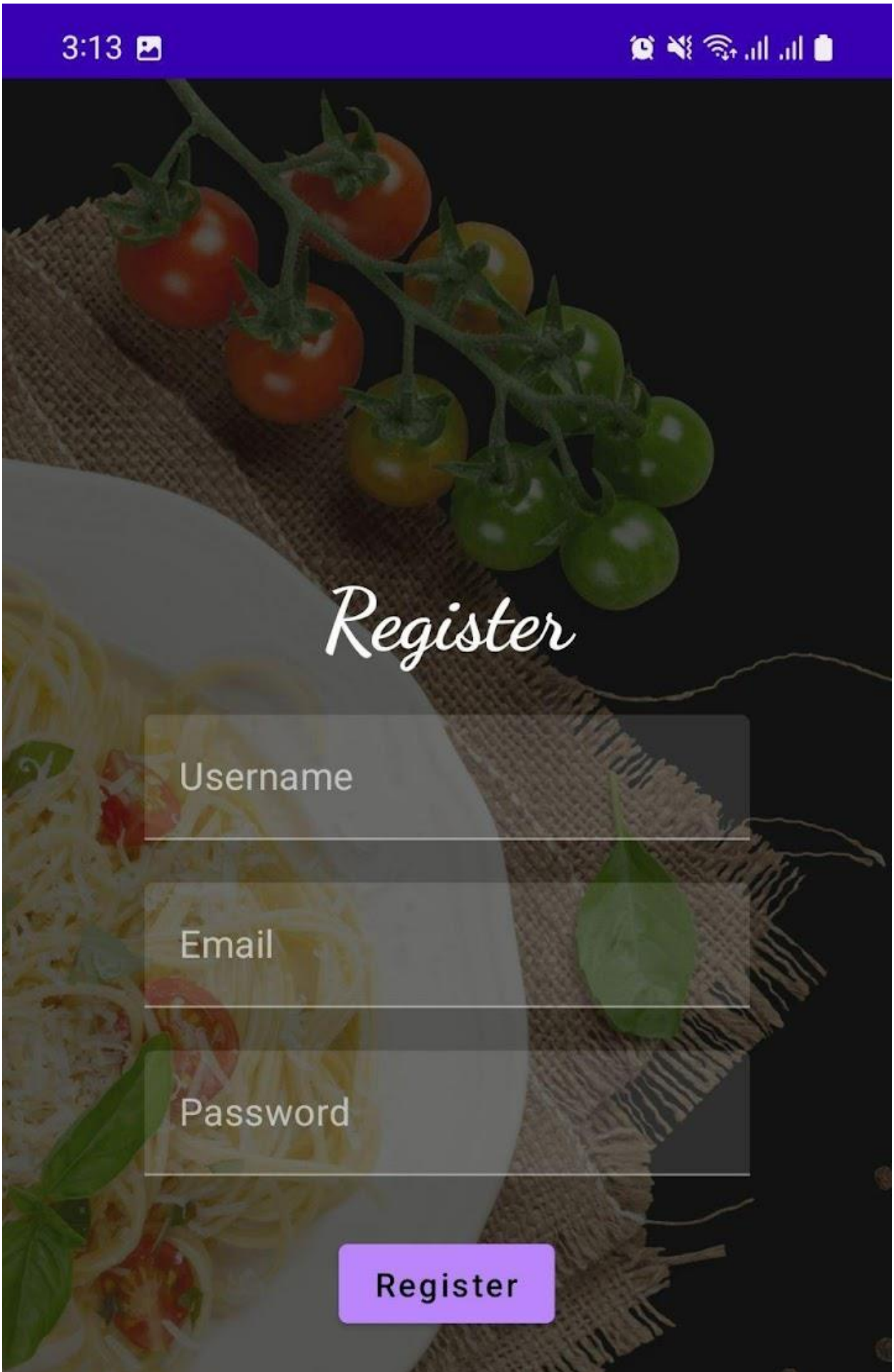


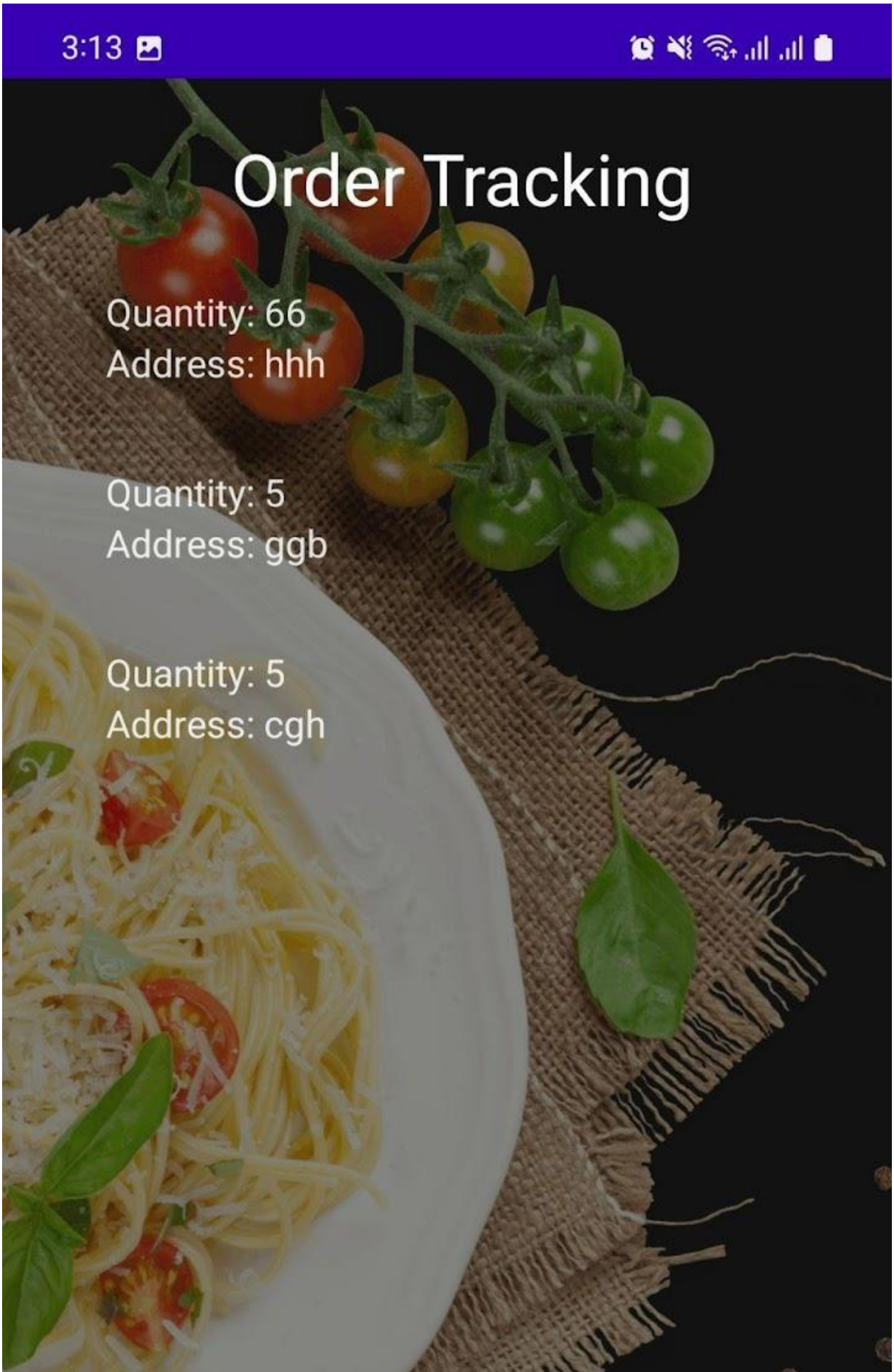
2.2ideation&Brainstorming map

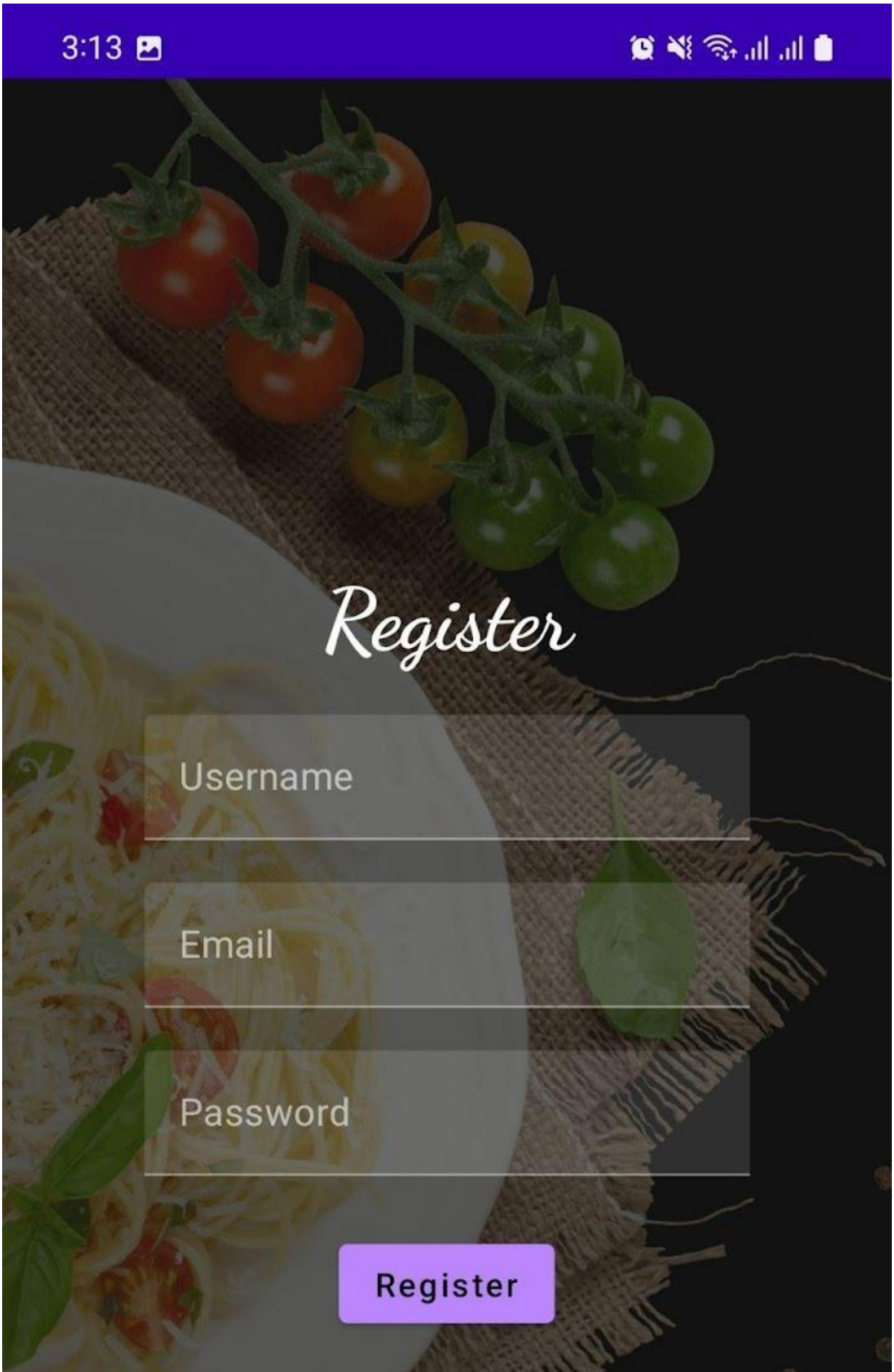


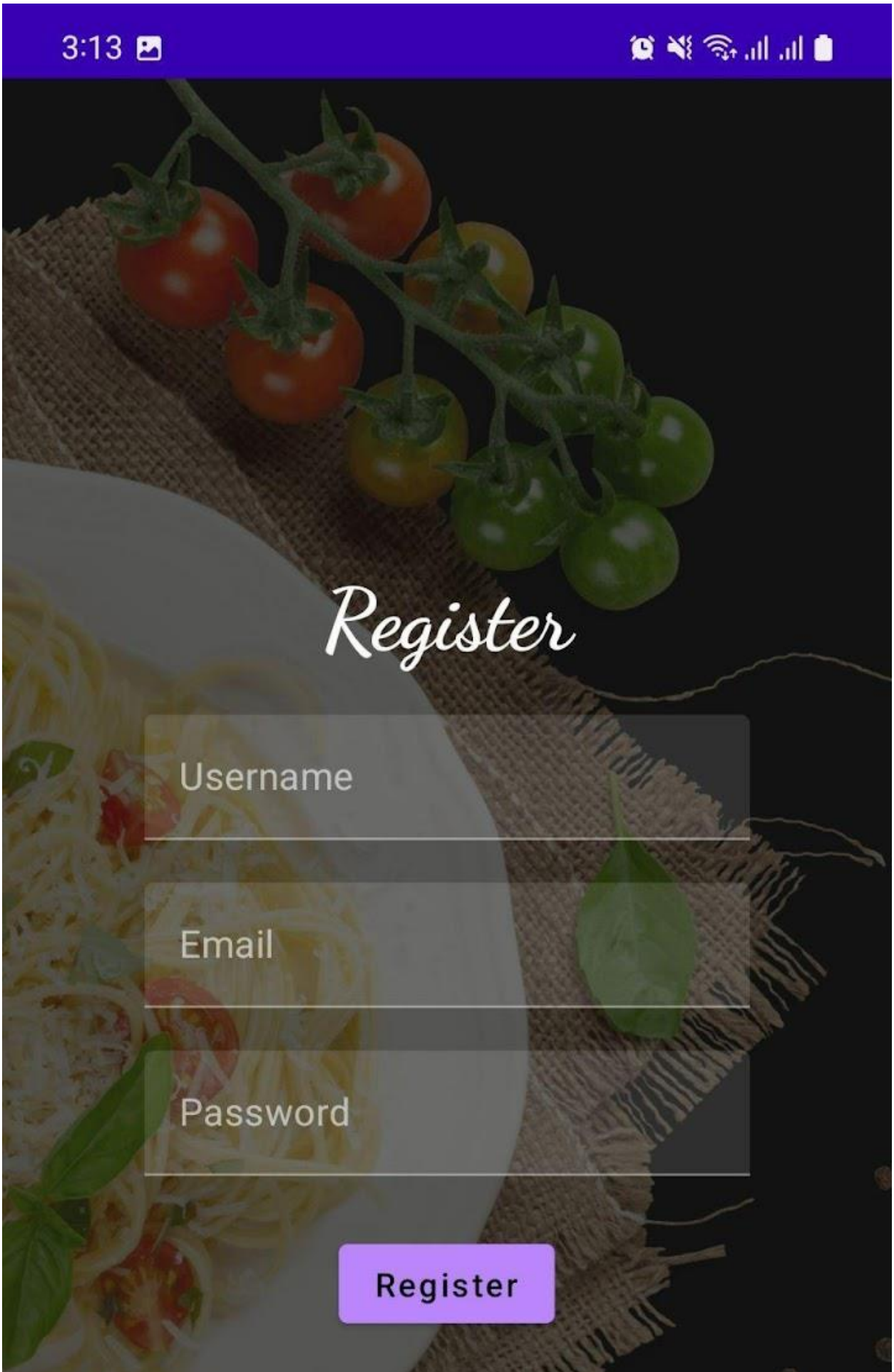
Result

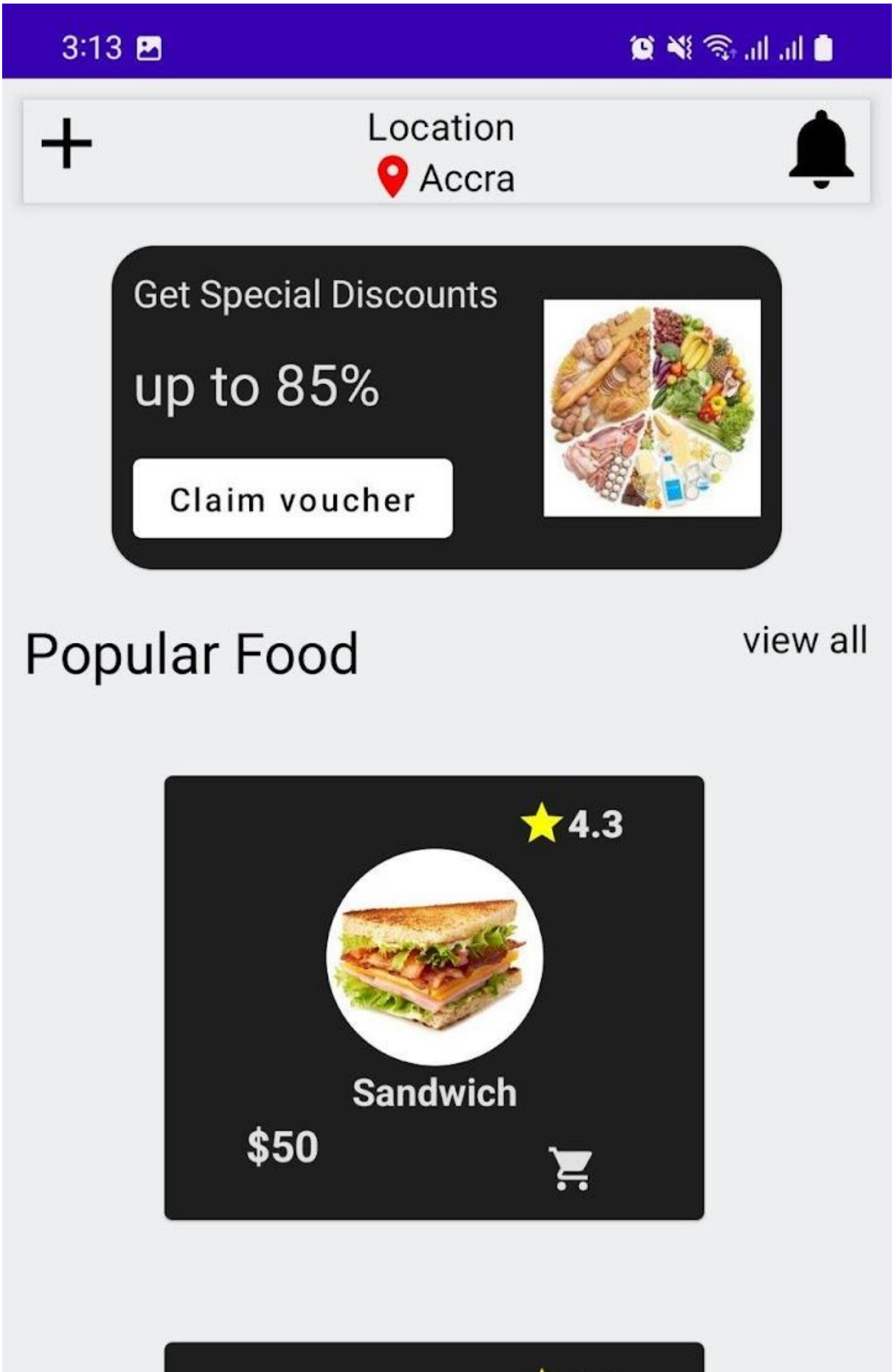

3:12
Login
Username
Password
Login
Sign up
Forget password?


3:13
Register
Username
Email
Password
Register


3:13
Order Tracking
Quantity: 66
Address: hhh
Quantity: 5
Address: ggb
Quantity: 5
Address: cgh


3:13
Register
Username
Email
Password
Register


3:13
Register
Username
Email
Password
Register


3:13
Location
Accra
Get Special Discounts
up to 85%
Claim voucher
Popular Food
view all
4.3
Sandwich
$50

Advantages&Disadvantages

Advantages:

1. Convenience: A customizable snack ordering and delivery app provides users with the convenience of ordering snacks from anywhere at any time without having to physically visit a store or restaurant.
2. Customization: A customizable snack ordering and delivery app allows users to customize their orders according to their preferences, such as choosing toppings, sauces, or sides.
3. Increased sales: A customizable snack ordering and delivery app can help businesses increase their sales by providing customers with an additional ordering channel.
4. Increased efficiency: A customizable snack ordering and delivery app can help businesses increase their efficiency by streamlining the ordering and delivery process.
5. Improved customer satisfaction: A customizable snack ordering and delivery app can improve customer satisfaction by providing a seamless and hassle-free ordering and delivery experience.
6. Technical issues: A customizable snack ordering and delivery app may face technical issues such as server downtime, connectivity issues, or bugs that may affect the ordering and delivery process.
7. Cost: Developing and maintaining a customizable snack ordering and delivery app can be expensive, particularly for small businesses with limited resources.
8. Dependence on technology: A customizable snack ordering and delivery app depends on technology, and any disruption to the technology can affect the ordering and delivery process.
9. Limited reach: A customizable snack ordering and delivery app may have limited reach, particularly in areas with low smartphone penetration or poor internet connectivity.
10. Competition: The market for customizable snack ordering and delivery apps is highly competitive, and businesses need to continually innovate to stay ahead of the competition.

Applications

1. User registration and login: Users should be able to create an account and log in to the app to place orders.

2. Menu and ordering system: The app should have a menu of snacks available for ordering, with prices and descriptions. Users should be able to add items to their cart, specify any customization requests, and check out when they are ready to place their order.
3. Payment processing: The app should be able to process payments securely and efficiently. Multiple payment options, such as credit card or PayPal, should be available.
4. Delivery tracking: Once an order has been placed, users should be able to track the progress of their delivery in real-time. This could include features such as a map showing the delivery driver's location and estimated time of arrival.
5. Customization options: To make the app stand out, it could offer customization options such as the ability to create custom snack boxes or choose from a variety of dipping sauces and toppings.
6. Loyalty program: To encourage repeat business, the app could offer a loyalty program that rewards users with points or discounts for each purchase.
7. Reviews and ratings: Users should be able to leave reviews and ratings for snacks and the delivery service. This could help other users make informed decisions about what to order.
8. Push notifications: The app could send push notifications to users to notify them about special promotions, new menu items, or updates about their orders.
9. Social media integration: Users should be able to share their orders on social media, and the app could also have social media pages to showcase its products and engage with customers.
10. Customer support: Finally, the app should have a customer support system in place to help users with any issues or questions they may have. This could include a chatbot, FAQ section, or email support.

These are just a few ideas for features that could be included in a customizable snack ordering and delivery app. By offering a convenient, user-friendly service with plenty of customization options, such an app could quickly become a popular choice for snack lovers.

Regenerate response

Conclusion

In conclusion, a customizable snack ordering and delivery app has the potential to be a successful business idea or startup. By including features such as user registration and login, menu and ordering system, payment processing, delivery

tracking, customization options, loyalty program, reviews and ratings, push notifications, social media integration, and customer support, the app can provide a convenient and user-friendly service that can quickly gain popularity among snack lovers. It's important to keep in mind that the success of the app will depend on factors such as marketing, pricing, and customer service, but by offering a unique and customizable service, the app can stand out from competitors and attract a loyal user base.

Future scope

1. Expansion to new markets: The app could expand to new markets, both domestically and internationally, to reach more customers and increase its user base.
2. Integration with smart devices: The app could integrate with smart devices such as Amazon Echo or Google Home, allowing users to place orders using voice commands.
3. Use of AI and machine learning: The app could use AI and machine learning to analyze customer data and make personalized recommendations for snacks or promotions based on their ordering history and preferences.
4. Partnership with snack brands: The app could partner with snack brands to offer exclusive products or discounts to users, further increasing customer loyalty.
5. Sustainability initiatives: The app could incorporate sustainability initiatives such as eco-friendly packaging or partnering with local farmers and producers to source ingredients, appealing to environmentally conscious consumers.
6. Expansion to other food categories: The app could expand beyond snacks to include other food categories such as meals or beverages, offering a wider range of options to users.

Appendix

Admin activity.kt

```
=package
com.example.snackorde
ring


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.SnackOrderin
gTheme
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}")
                    }
                }
            }
        }


    }
}
```

Login activity.kt

```
package
com.example.snackorde
ring


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.SnackOrderin
gTheme


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,
ss              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.snackord
ering


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.RoundedCorne
rShape
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.SnackOrderin
gTheme


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




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

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.snackord
ering


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()
    }





    @SuppressLint("Range")
    fun getOrderByQuantity(quantity: String): Order? {
```

```
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE $COLUMN_QUANTITY = ?", arrayOf(quantity))
        var order: Order? = null
        if (cursor.moveToFirst()) {
            order = Order(
                id = cursor.getInt(cursor.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_A
DDRESS)),
        )
    }
    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_Q
UANTITY)),
                address                                =
cursor.getString(cursor.getColumnIndex(COLUMN_A
DDRESS)),
            )
            orders.add(order)
        } while (cursor.moveToNext())
    }
    cursor.close()
    db.close()
    return orders
}
```

```
}
```

RegisterActivity.kt

```
package
com.example.snackorde
ring


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.SnackOrderin
gTheme


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(
        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.snackorde
ring


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.KeyboardAction
s
import
androidx.compose.foundation.text.KeyboardOption
s
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.textInputServiceFact
ory
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.SnackOrderin
gTheme


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)
}
```

UserActivity.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()
    }


    @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_FI
RST_NAME)),
            lastName                                     =
cursor.getString(cursor.getColumnIndex(COLUMN_LA
ST_NAME)),
            email                                        =
cursor.getString(cursor.getColumnIndex(COLUMN_E
MAIL)),
            password                                     =
cursor.getString(cursor.getColumnIndex(COLUMN_PA
SSWORD)),
        )
    }
    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_FI
RST_NAME)),
            lastName                                     =
cursor.getString(cursor.getColumnIndex(COLUMN_LA
ST_NAME)),
```

```
            email                                    =
cursor.getString(cursor.getColumnIndex(COLUMN_E
MAIL)),
            password                                 =
cursor.getString(cursor.getColumnIndex(COLUMN_PA
SSWORD)),
        )
    }
    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_FI
RST_NAME)),
                lastName                             =
cursor.getString(cursor.getColumnIndex(COLUMN_LA
ST_NAME)),
                email                                =
cursor.getString(cursor.getColumnIndex(COLUMN_E
MAIL)),
                password                             =
cursor.getString(cursor.getColumnIndex(COLUMN_PA
SSWORD)),
```

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


}
```