A customizable Snack Ordering And Delivery App

1.INTRODUCTION

- A Snack delivery app that provides Snack delivery at your door in very less time and with the best packaging.
- Providing food from every famous food place near you. Order food with the best user experience.

over view

- A customizable snack ordering and delivery app is a software application that enables customers to order their favorite snacks and have them delivered to their doorstep. The app can be customized to meet the needs of various snack vendors, such as food trucks, cafes, and restaurants, allowing them to offer their products to customers online.
- The app typically has several features, including a user-friendly interface that allows customers to browse through the available snacks, add them to their cart, and make payments securely.
- The app may also allow customers to customize their orders, such as adding or removing specific ingredients, selecting a specific delivery time, and choosing their preferred delivery method.

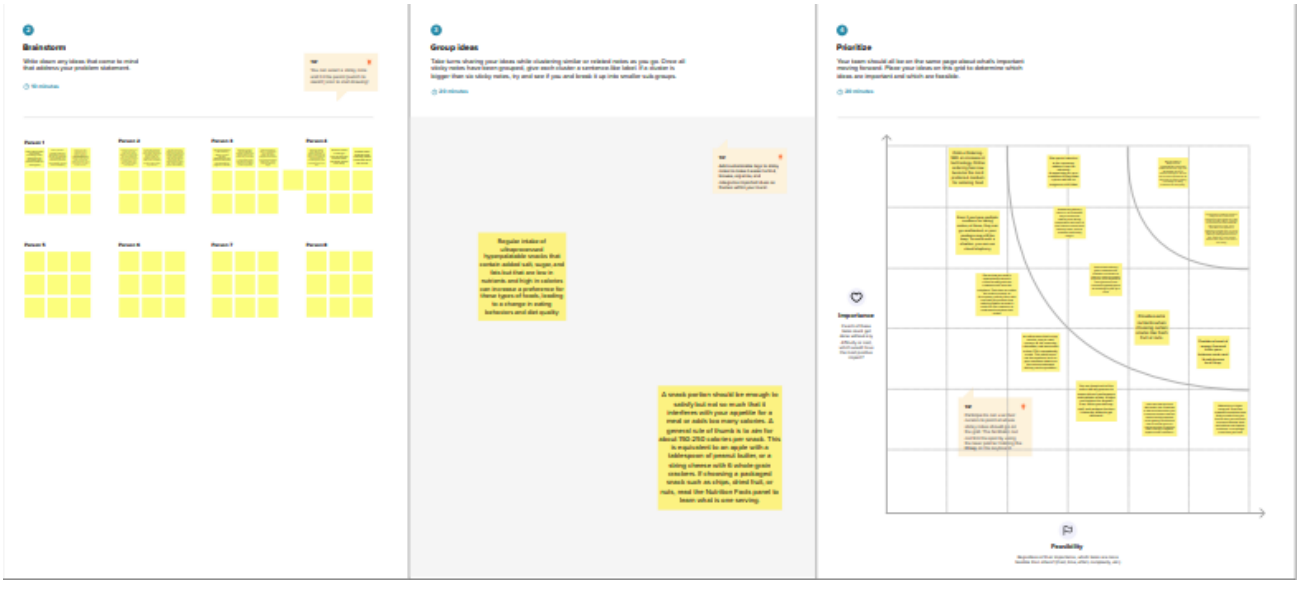
- Additionally, the app can provide real-time tracking of the customer's order, notifying them of the estimated delivery time and the status of their order.
- It can also provide delivery personnel with the necessary information to complete the delivery, such as the customer's address and contact details.
- Overall, a customizable snack ordering and delivery app can help snack vendors streamline their operations, reach more customers, and provide a more convenient ordering and delivery experience for their customers.

PURPOSE

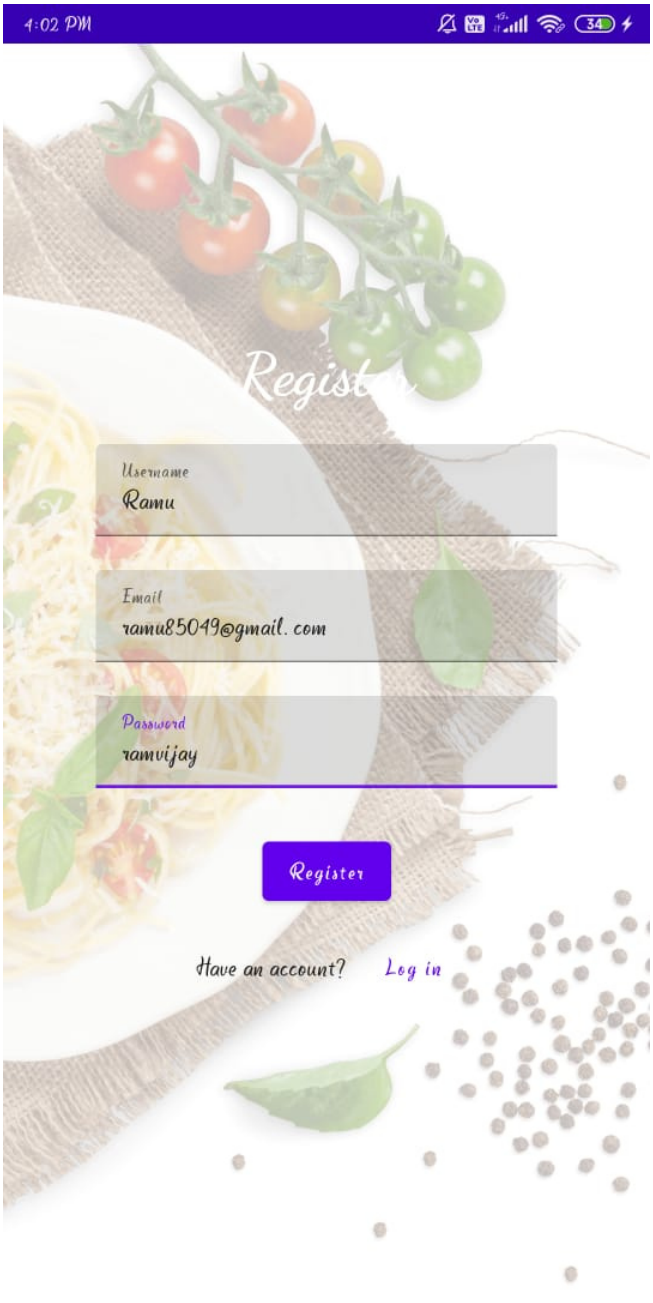
- The purpose of a customizable snack ordering and delivery app is to provide a convenient and efficient way for customers to order their favorite snacks from their preferred snack vendors and have them delivered to their doorstep. The app offers several benefits to both snack vendors and customers, including:
- Increased customer reach: The app enables snack vendors to reach a larger customer base beyond their physical location, which can help them grow their business.
- Convenience: Customers can easily order their favorite snacks from the comfort of their homes or offices, saving them time and effort.
- Customization: The app can be customized to meet the specific needs of different snack vendors, allowing them to offer unique snack options and pricing.
- Real-time tracking: Customers can track their orders in real-time, which increases transparency and reduces the chances of delivery errors.
- Increased efficiency: The app streamlines the ordering and delivery process, reducing the time and resources required

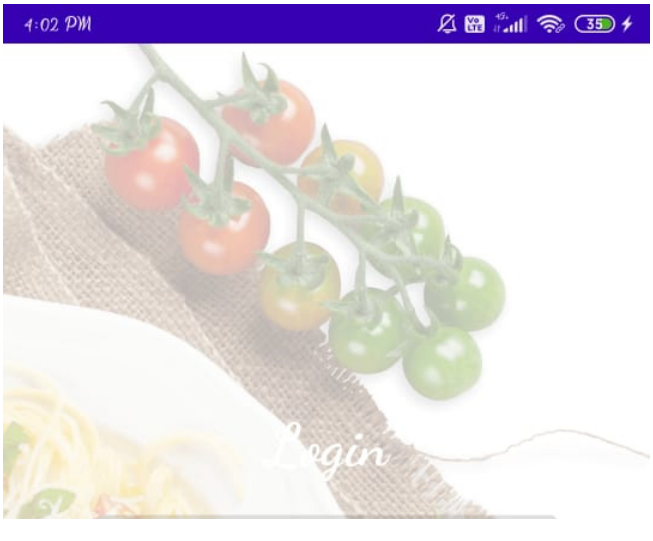
by snack vendors to fulfill orders.

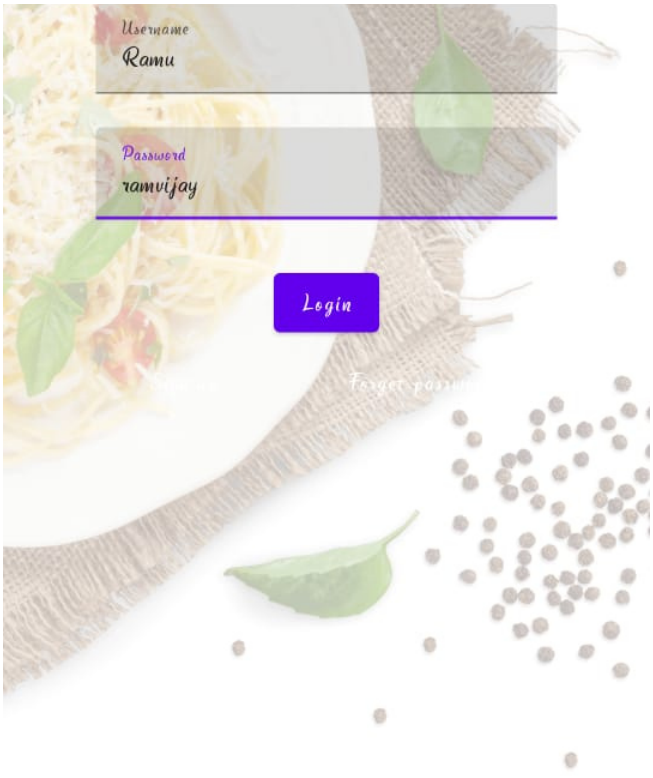
Overall, the purpose of a customizable snack ordering and delivery app is to create a seamless and enjoyable experience for customers, while also helping snack vendors increase their sales and expand their reach.

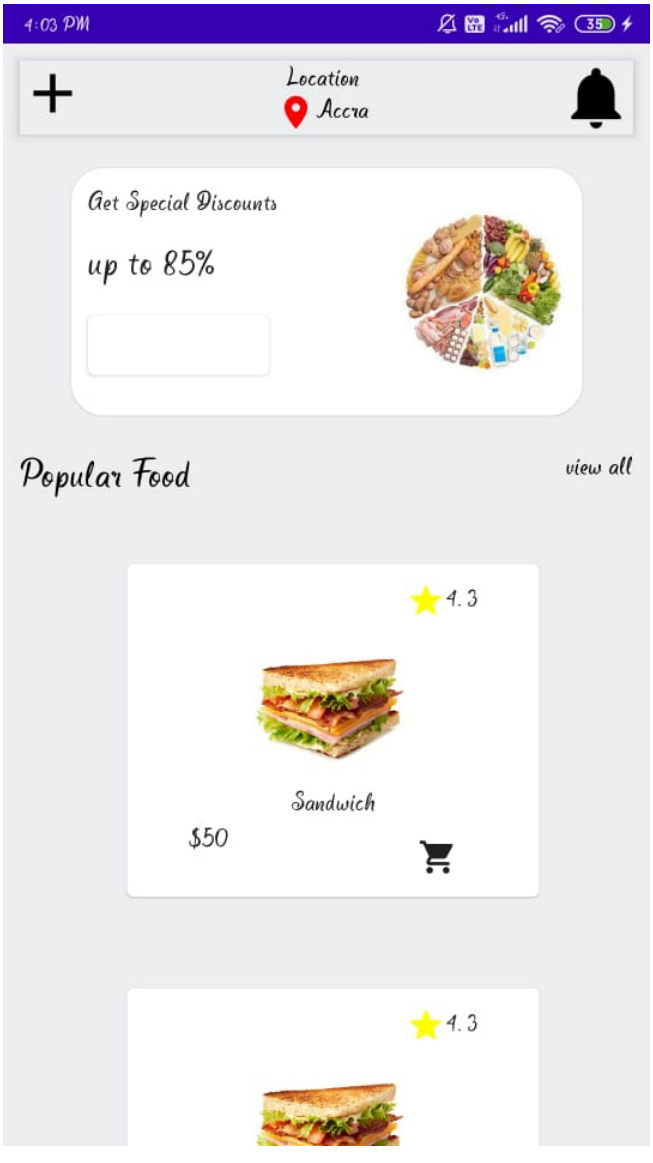


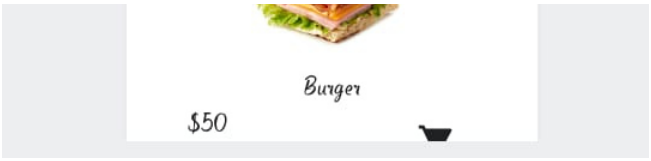
Result

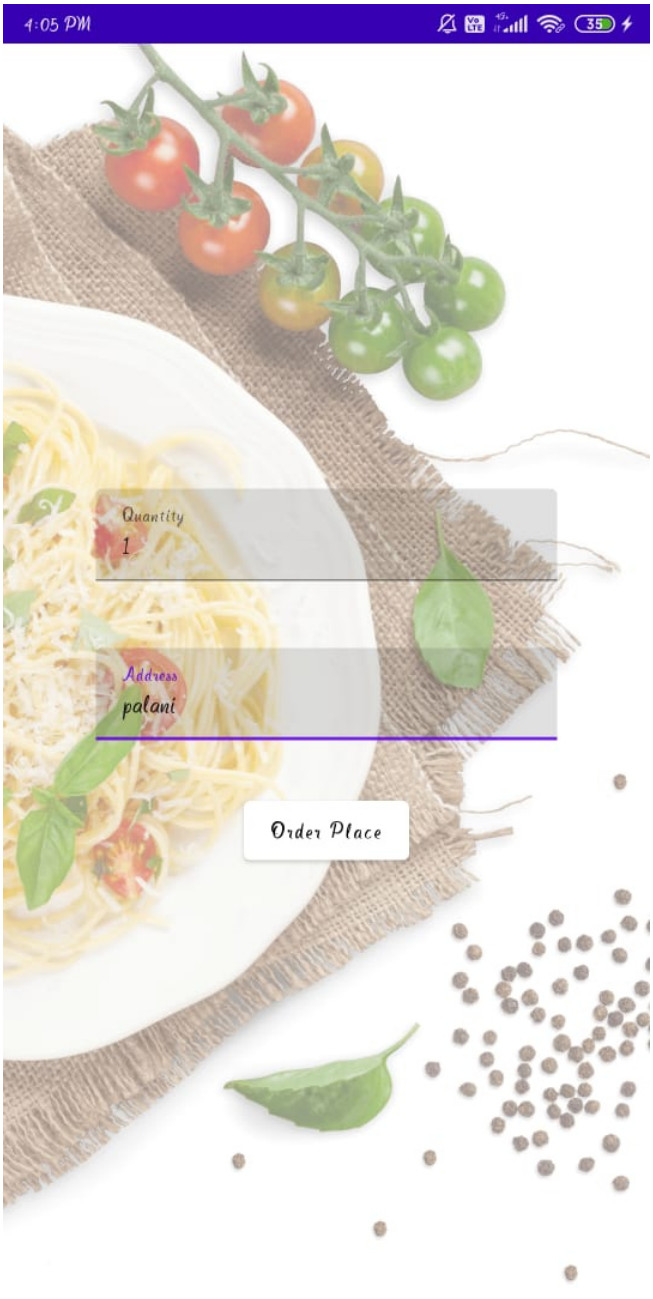

4:02 PM
Register
Username
Ramu
Email
ramu85049@gmail.com
Password
ramvijay
Register
Have an account?
Log in


4:02 PM
Login


Username
Ramu
Password
ramvijay
Login


4:03 PM
Location
Accra
Get Special Discounts
up to 85%
Popular Food
view all
4.3
Sandwich
$50
4.3





ADVANTAGE

- Helps curb your appetite to prevent overeating at the next meal.
- Provides extra nutrients when choosing certain snacks like fresh fruit or nuts.

- Can help maintain adequate nutrition if one has a poor appetite but cannot eat full meals, such as due to an illness.
- It is called advantages of snack ordering.
- Increases Sustained Energy. Sugar-laden processed foods, snacks full of trans fats, or a mighty carb-overload can result in awful energy crashes come mid-afternoon. ...
- Improves Cognitive Function.
- Better Nutrition for a Better Mood.
- Physical Health Boosts Mental Health.

Disadvantages

- **Delivery men in danger:**

Delivery men deliver the food, if it is sunny or rainy, he is waiting outside the restaurant to take the order and deliver your order on time.

- **Health issues:**

The attractive dishes sometimes make health issues due to their ingredients, and the hot food packed in plastic bags or boxes leads to health issues. If you get this type of food on regular basis, it may cause food poisoning and makes you obese too.

- It is called disadvantages of snack ordering.

APPLICATIONS

- ## Ordering process is easy

A WebApp is fast, easy and comfortable to use. There are no misunderstandings or frustrations as can happen with ordering over the phone. In a nutshell, your customers choose

to order food through the WebApp because it’s at their fingertips.

- **Exposure to new customers**

Online ordering through a food delivery WebApp can help you reach new customers outside your regulars and locals. By enhancing your brand’s online presence in the market, you can boost your sales with new and returning customers.

- **Online ordering is convenient**

A WebApp allows customers to order anytime, anywhere using their mobiles, tablets or other handheld devices. With a food delivery WebApp, the customer can quietly place an order without the hassle of talking over the phone.

- **More business opportunities**

Sometimes customers want your food but in the comfort of their own home. Whether that’s due to ongoing restrictions, bad weather or simply the desire to stay home, by offering delivery you’re able to serve a wider range of customers.

- **Stay ahead of the competition**

The percentage of restaurants and takeaway establishments that have a food delivery WebApp is surprisingly small. By making your restaurant available to customers at the touch of a screen immediately puts you ahead of the game.

- **Greater reach**

Your restaurant seating capacity may be 100 at a time, or perhaps less, but with a food delivery service app you can

reach thousands of people. All you need is an integrated ordering system, and you are good to go!

- **Better customer data**

Who are your regular customers? Which food items are popular? Are they aware of your promotions and offers? These and many other related questions can be answered using analytics from your app. Once you know what your customers like, you can keep them coming back with targeted offers.

CONCLUSION:

A lot of innovations have been introduced and/or being under development to achieve this goal, for example:

•preparation of crunchy and lighter snacks with different shapes and size;

•introduction of extrusion cooking;

•inclusion of nonmeat ingredients in traditional meat products;

•incorporation of meat into traditional cereal based snacks;

•incorporation of various ingredients having functional values as high dietary fiber, natural antioxidants and preservatives;

•improved packaging conditions and technologies as MAP, vacuum packaging, active and intelligent packaging systems etc.

FEATURES SCOPE:

Around the world, adults consume energy outside of traditional meals such as breakfast, lunch, and dinner. However, because

there is no consistent definition of a “snack,” it is unclear whether those extra eating occasions represent additional meals or snacks. The manner in which an eating occasion is labeled (e.g., as a meal or a snack) may influence other food choices an individual makes on the same day and satiety after consumption. Therefore, a clear distinction between “meals” and “snacks” is important. This review aims to assess the definition of extra eating occasions, to understand why eating is initiated at these occasions, and to determine what food choices are common at these eating occasions in order to identify areas for dietary intervention and improvement. Part I of this review discusses how snacking is defined and the social, environmental, and individual influences on the desire to snack and choice of snack. The section concludes with a brief discussion of the associations of snacking with cardiometabolic health markers, especially lipid profiles and weight. Part II addresses popular snack choices, overall snacking frequencies, and the demographic characteristics of frequent snackers in several different countries. This review concludes with a recommendation for nutrition policymakers to encourage specific health-promoting snacks that address nutrient insufficiencies and excesses.

APPENDIX

SOURCE CODE

Database 1

Step 1 : Create User data class

```
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?,

    )
```

Step 2 : Create an UserDao interface

```
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)
}
```

Step 3 : Create an UserDatabase class

```
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
            }
        }
    }
```

```
}
```

Step 4 : Create an UserDatabaseHelper class

```
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
}

@SuppressLint("Range")
fun getAllUsers(): List<User> {
    val users = mutableListOf<User>()
    val db = readableDatabase
    val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
    if (cursor.moveToFirst()) {
        do {
            val user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName = cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName = cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
            users.add(user)
        } while (cursor.moveToNext())
    }
    cursor.close()
    db.close()
```

```
        return users
    }
}
```

Database 2

Step 1 : Create Order data class

```
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?,
)
```

Step 2 : Create OrderDao interface

```
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)
}
```

Step 3 : Create OrderDatabase class

```
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
            }
        }
    }
}
```

Step 4 : Create OrderDatabaseHelper class

```
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()
    }


    @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_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
    }

}
```