12/17/2021

Ché Alexander and Meshach Maynard

UNIVERSITY OF THE WEST INDIES

Store-Manager App

SWEEN3003: Web & Mobile

Contents

[About the Application 2](#_Toc90704573)

[Order Screen 2](#_Toc90704574)

[Show Details Screens 2](#_Toc90704575)

[Add Meals Screen 4](#_Toc90704576)

[Add Funds Screen 4](#_Toc90704577)

[Checkout Screen 5](#_Toc90704578)

[Statistics Screen 6](#_Toc90704579)

[Code of the Application 7](#_Toc90704580)

[Database Setup 7](#_Toc90704581)

[MealDbScheme 7](#_Toc90704582)

[Navigation Drawer 9](#_Toc90704583)

[Implementing Order 11](#_Toc90704584)

[Implementing ShowDetails 13](#_Toc90704585)

[OrderManager 15](#_Toc90704586)

[Implementing Add-Meals 16](#_Toc90704587)

[Implementing Add-Funds 17](#_Toc90704588)

[Implementing Checkout 18](#_Toc90704589)

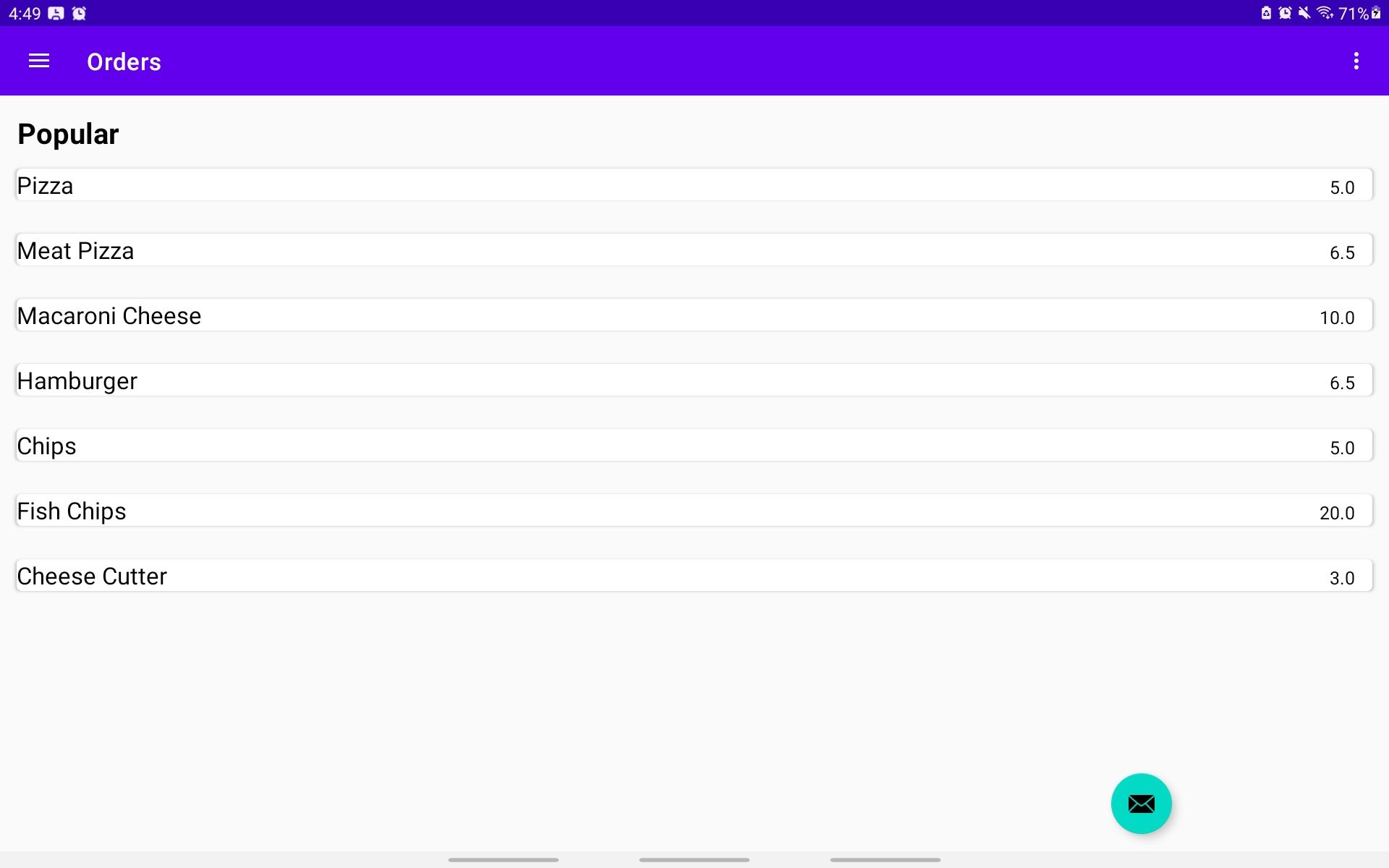
[Implementing Statistics 19](#_Toc90704590)

# About the Application

Store Manager is a database app the allows clients who own a shop/restaurant to monitor their business’ profits, which is done via entering data into the app. For instance, enter the money spent on products for the business. In addition, the app also acts as a cashier substitute as it allows users to track order requests. Furthermore, the app uses Navigation Drawer, where the main navigation is **Order, Add-Meal, Add-Funds, Checkout,** and **Statistics.**

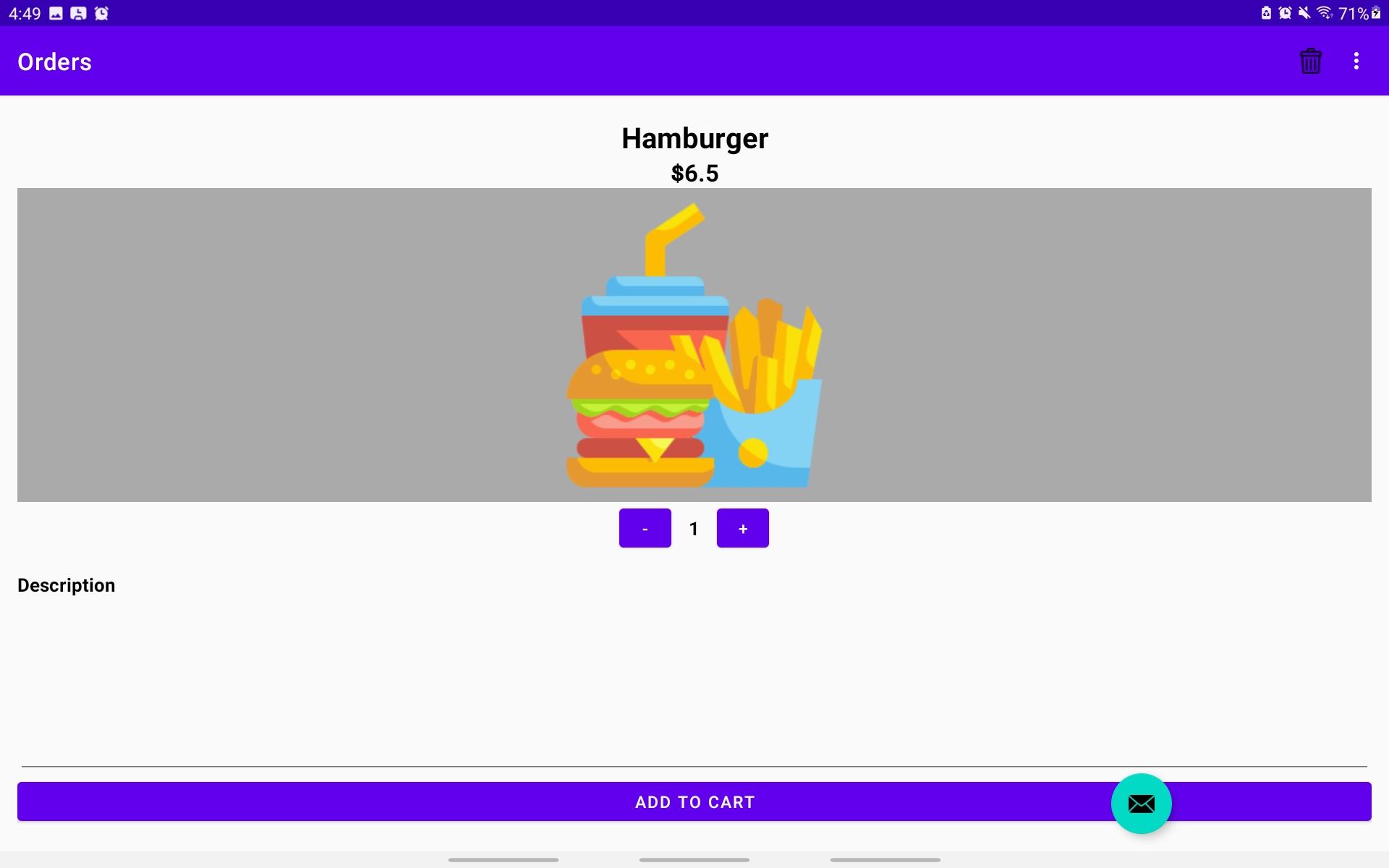
## Order Screen

The Order Screen is the first screen the user will see. It contains the menu list of the store, depicting both the name and price of each dish.



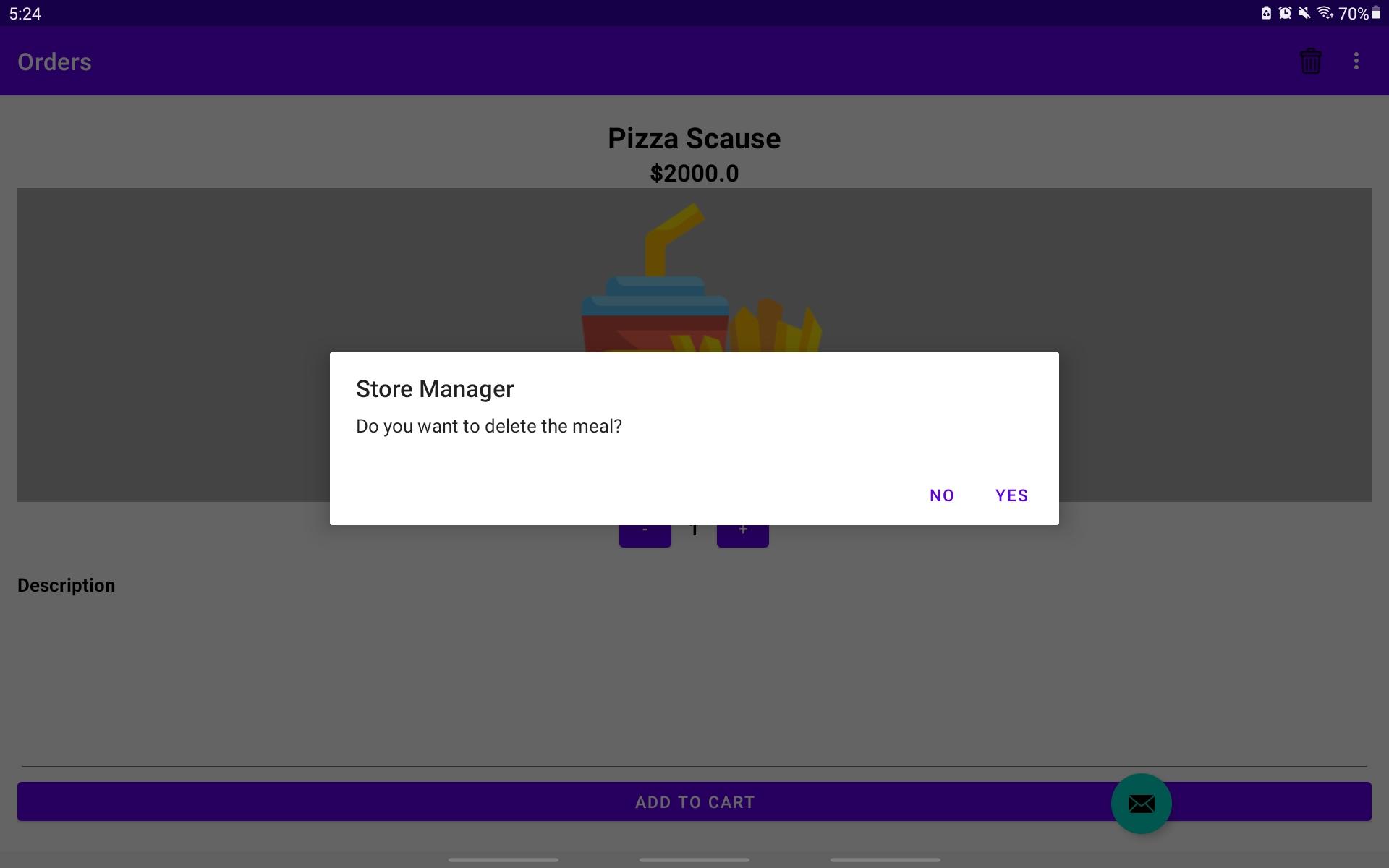
### Show Details Screens

This screen allows the user (cashier) to place the meal in the customer cart. However, to access this screen the user just has to press the desired meal. Additionally, the screen displays the **Name** and **Price** of the meal, and **Add-to-Cart** button, and a **Counter**. The **counter** allows the user to input and tracks the amount of that meal the customer is purchasing. Once completed, the user will have to press the **Add-to-Cart** button to place the dish in the **Checkout** list.



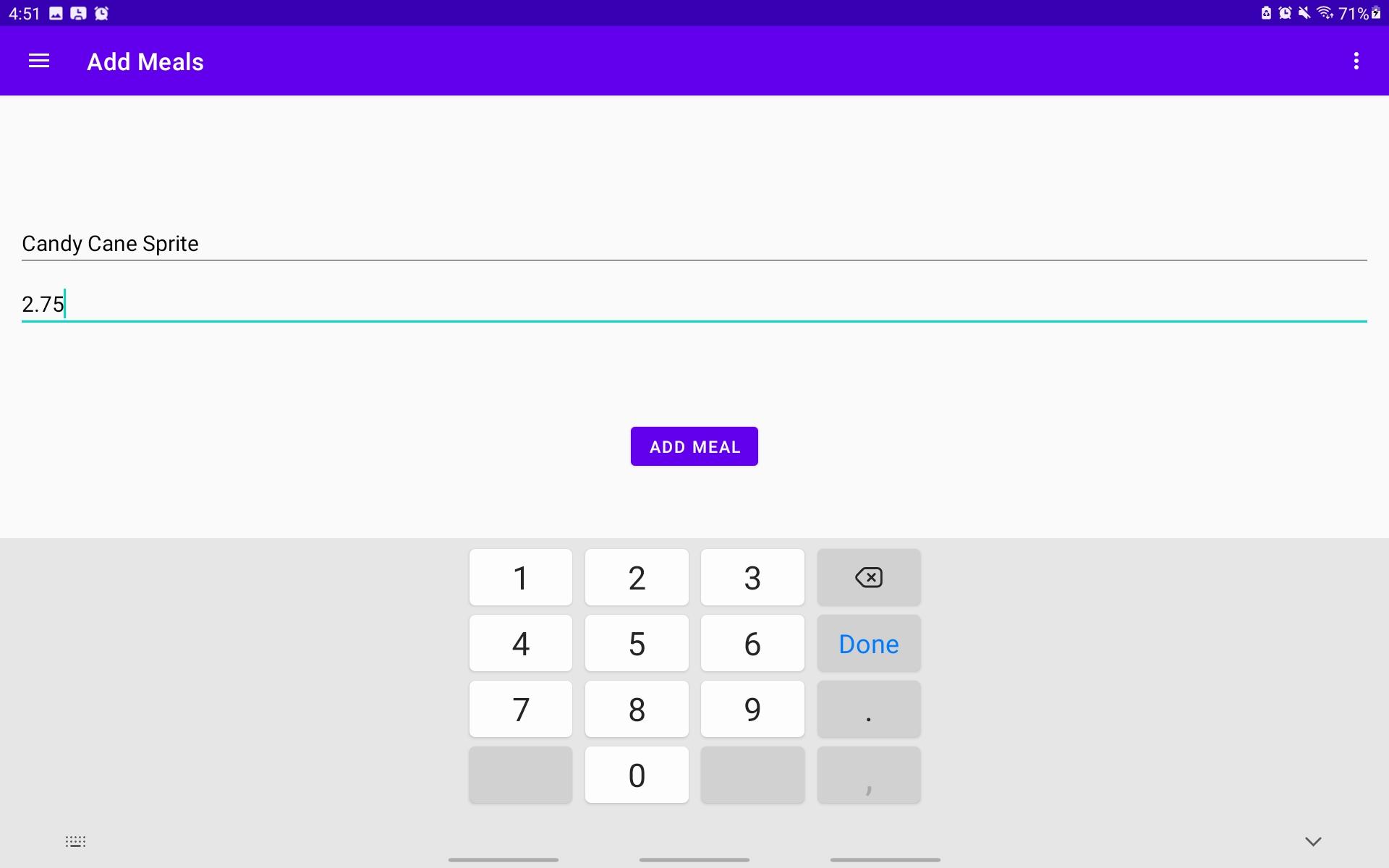
To exit the **Show Details** screen the user can only press the back button as Navigation Drawer is disabled. This is due to fragments overlaying on each other.

Additionally, there is a delete icon in the menu bar in this fragment. The icon once press allows the user to delete a dish from the database.



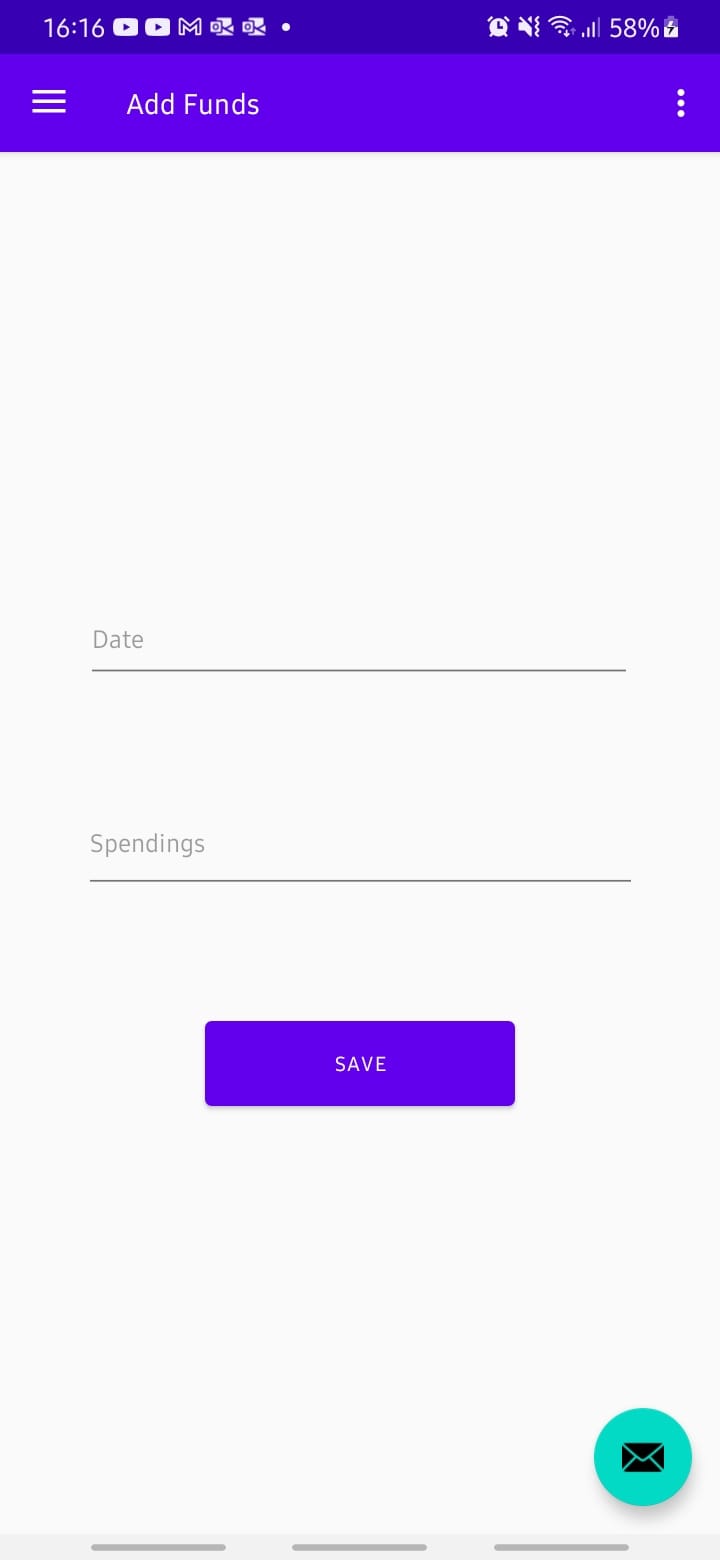
## Add Meals Screen

Add-Meals screen allows the user to add new meals to the menu. All the user has to do is place the name and price of the dish in their respected fields and click **Add Meal**. Doing this will save the dish to the database.



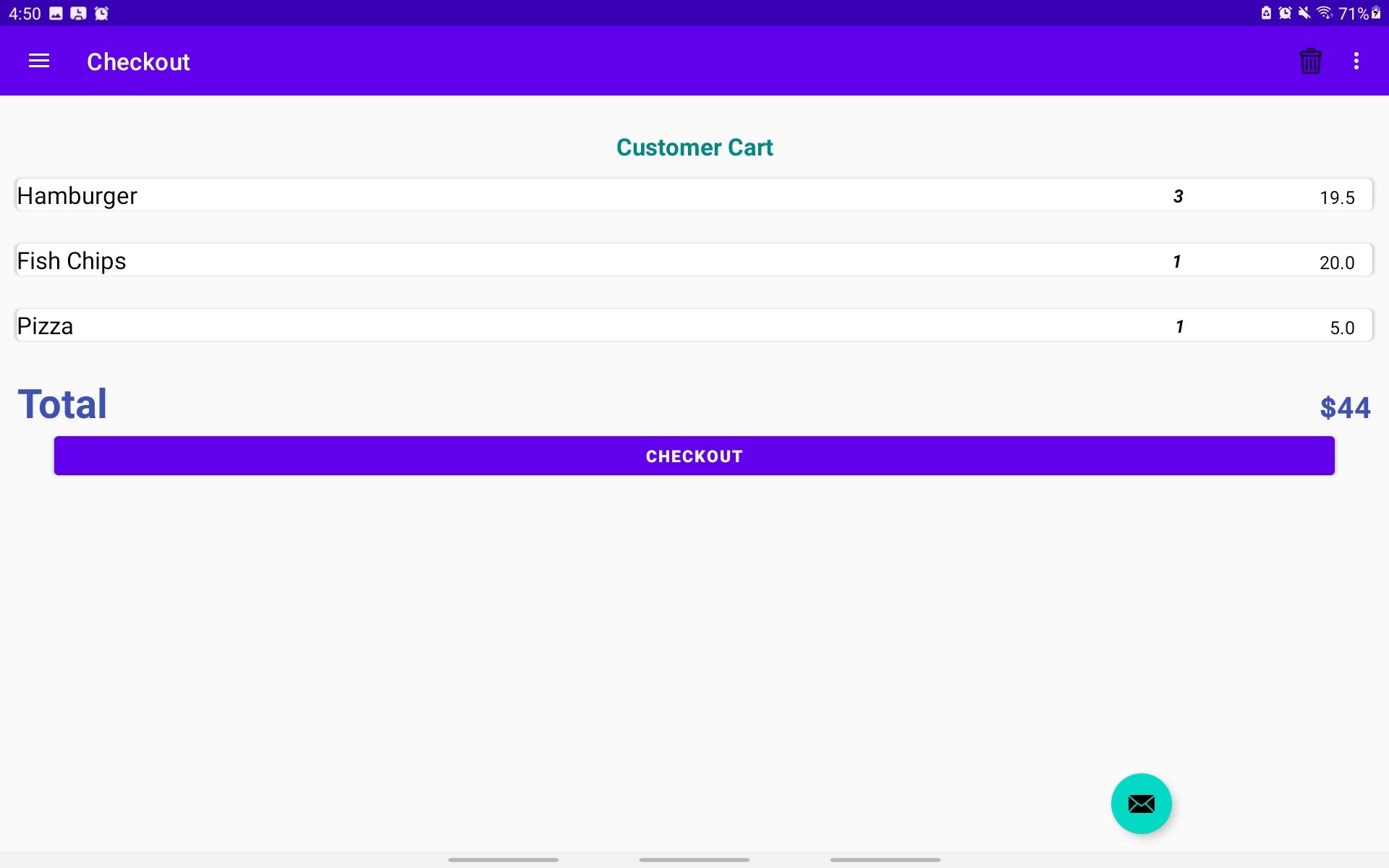
## Add Funds Screen

This screen is where spending for the month goes because when the user is buying groceries to be able to prepare the dishes, they will spend money, and this money needs to be accounted for to help calculate the profits for that month and help manage the money for next month.

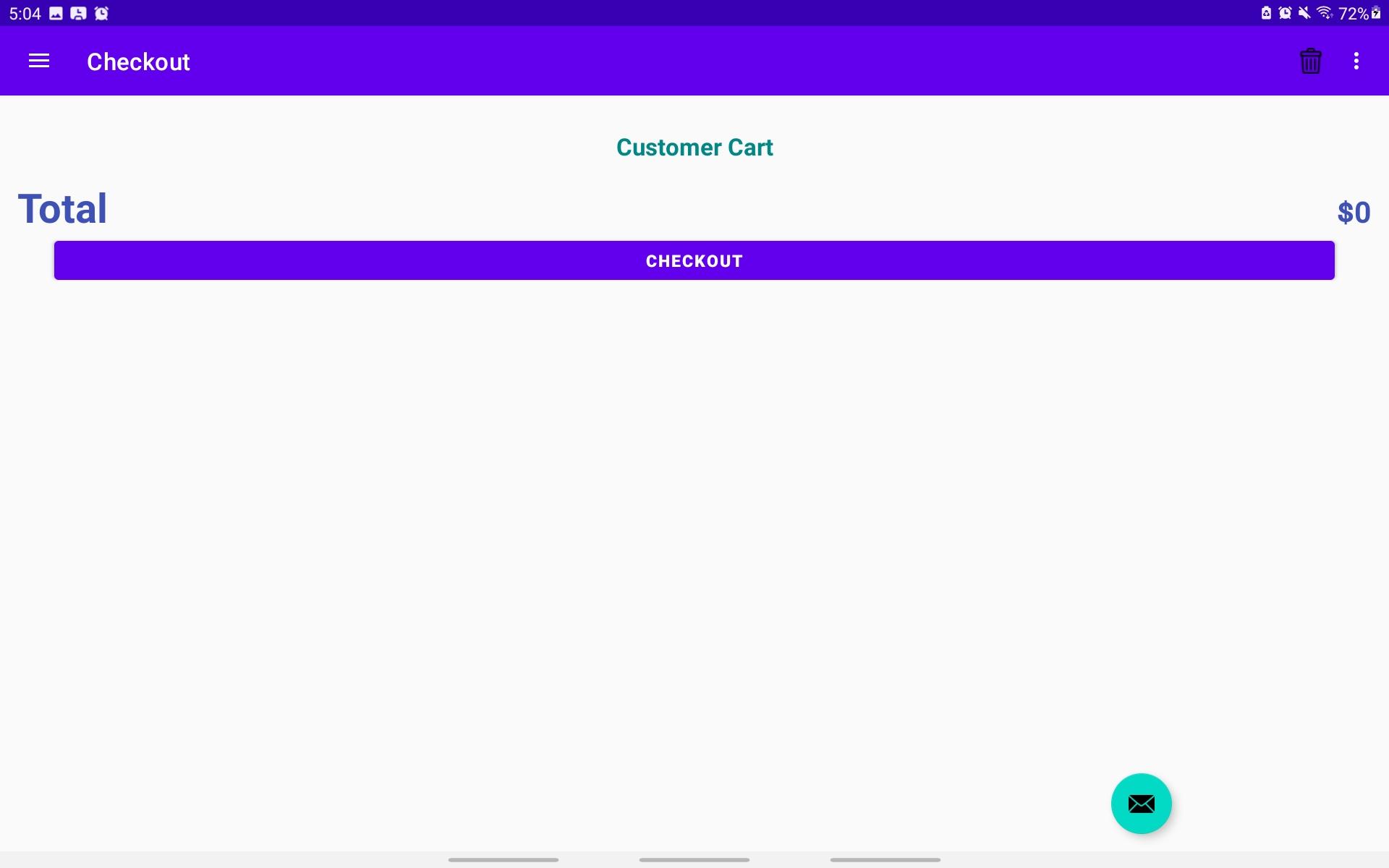


## Checkout Screen

The Checkout screen holds the list of customer requested dishes and the sum total of them. The customer and user can both see the total price and the list of items on the screen, moreover, clicking the **Checkout** button clear the screen.

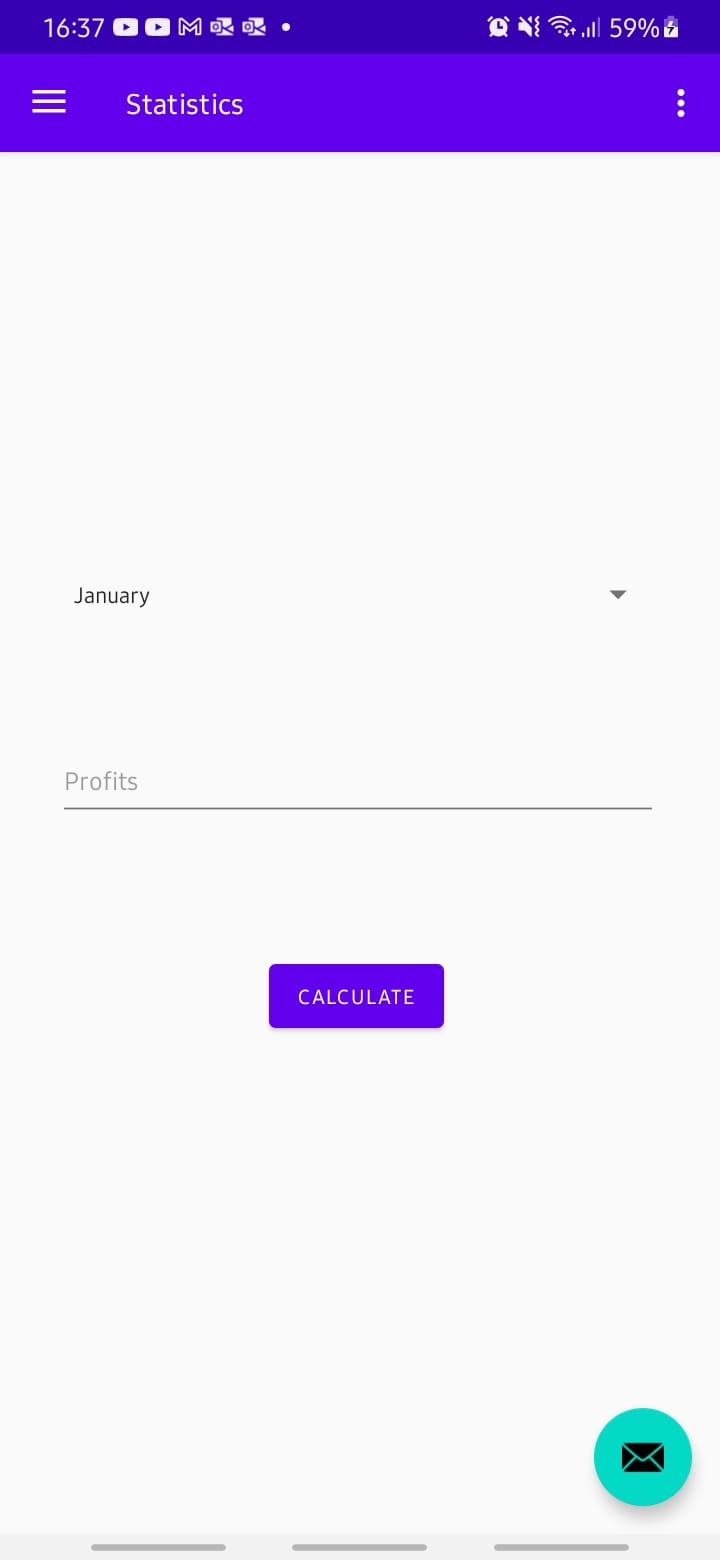


After Pressing **Checkout**

****

## Statistics Screen

From this screen, the user can see the profits that were made just by selecting which month they want.



# 

# Code of the Application

This section of the report will detail the code implementation of the application.

## Database Setup

Since this application needs a database to operate, a local SQLite table was created.

### MealDbScheme

public class MealDbSchema {  
 public static final class MealTable{  
 public static final String *NAME* = "meals";  
  
 public static final class Cols{  
 public static final String *UUID* = "uuid";  
 public static final String *MealName* ="name";  
 public static final String *MealCost* = "price";  
 public static final String *TimesOrder* = "timesOrder";  
 }  
 }  
}

However, for the database to be fully used both **MealBaseHelper and MealCusorWrapper** are needed.

public class MealBaseHelper extends SQLiteOpenHelper {  
 private static final int *VERSION* = 1;  
 private static final String *DATABASE\_NAME* ="mealBase.db";  
  
  
 public MealBaseHelper(Context context){  
 super(context,*DATABASE\_NAME*,null,*VERSION*);  
 }  
  
 @Override  
 public void onCreate(SQLiteDatabase db) {  
 db.execSQL("create table " + MealTable.*NAME* + "(" +  
 " \_id integer primary key autoincrement, " +  
 MealTable.Cols.*UUID* + ", " +  
 MealTable.Cols.*MealName* + ", " +  
 MealTable.Cols.*MealCost* + ", " +  
 MealTable.Cols.*TimesOrder* +  
 ")"  
 );  
 }

public class MealCursorWrapper extends CursorWrapper {  
 public MealCursorWrapper(Cursor cursor) {  
 super(cursor);  
 }  
  
 public Meal getMeal() {  
 String uuidString = getString(getColumnIndex(MealDbSchema.MealTable.Cols.*UUID*));  
 String name = getString(getColumnIndex(MealDbSchema.MealTable.Cols.*MealName*));  
 double price = getDouble(getColumnIndex(MealDbSchema.MealTable.Cols.*MealCost*));  
 int timesOrdered = getInt(getColumnIndex(MealDbSchema.MealTable.Cols.*TimesOrder*));  
  
 Meal meal = new Meal(UUID.*fromString*(uuidString));  
 meal.setName(name);  
 meal.setPrice(price);  
 meal.setTimesOrder(timesOrdered);  
 return meal;  
 }  
}

The **Menulab** class allows the application to easily perform database queries, some of which include fetching meal lists, updating meals, adding meals and deleting meals from the table.

public class MenuLab {  
 private static MenuLab *sMenuLab*;  
 private Context mContext;  
 private SQLiteDatabase mDatabase;  
  
 public static MenuLab get(Context context) {  
 if (*sMenuLab* == null) {  
 *sMenuLab* = new MenuLab(context);  
 }  
 return *sMenuLab*;  
 }  
  
 private MenuLab(Context context) {  
 mContext = context.getApplicationContext();  
 mDatabase = new MealBaseHelper(mContext)  
 .getWritableDatabase();  
 }

// Add Meals in Database  
public void addMeal(Meal m) {  
 ContentValues values = *getContentValues*(m);  
 mDatabase.insert(MealDbSchema.MealTable.*NAME*, null, values);  
}  
  
public void deleteMeal(Meal m) {  
 String uuidString = m.getId().toString();  
 mDatabase.delete(MealDbSchema.MealTable.*NAME*,  
 MealDbSchema.MealTable.Cols.*UUID* + " = ?",  
 new String[]{uuidString});  
}  
  
// Update Meals, Meanly for updating Times ordered  
public void updateCrime(Meal meal) {  
 String uuidString = meal.getId().toString();  
 ContentValues values = *getContentValues*(meal);  
 mDatabase.update(MealDbSchema.MealTable.*NAME*, values,  
 MealDbSchema.MealTable.Cols.*UUID* + " = ?",  
 new String[] { uuidString }); // String is not placed directly in the clause to prevent SQL Injection  
}

## Navigation Drawer

Navigation Drawer is the main component of this program, as it allows for easy navigation. The navigation drawer was created via android studios’ template, however, modifications still needed to occur for implementing new fragments to navigate to.

The icons in the drawer were created in the **activity\_nav\_drawer.xml**

<group android:checkableBehavior="single">  
 <item  
 android:id="@+id/nav\_orders"  
 android:icon="@drawable/ic\_order\_icon"  
 android:title="@string/menu\_orders" />  
 <item  
 android:id="@+id/nav\_addFoods"  
 android:icon="@drawable/add\_meal\_icon"  
 android:title="@string/menu\_addMeal" />  
 <item  
 android:id="@+id/nav\_funds"  
 android:icon="@android:drawable/ic\_input\_add"  
 android:title="@string/add\_funds" />  
 <item  
 android:id="@+id/nav\_cart"  
 android:icon="@drawable/checkout"  
 android:title="@string/outBtn"/>  
 <item  
 android:id="@+id/nav\_stats"  
 android:icon="@drawable/icon\_trend"  
 android:title="@string/menu\_stats" />  
</group>

The method used to implement the navigation feature was to add references to the fragments in **mobile\_navigati.xml** file.

<navigation xmlns:android="http://schemas.android.com/apk/res/android"  
 ///  
 android:id="@+id/mobile\_navigation"  
 app:startDestination="@+id/nav\_orders">  
  
 <fragment  
 android:id="@+id/nav\_orders"  
 android:name="com.example.storemanager.ui.order.OrdersFragment"  
 android:label="@string/menu\_orders"  
 tools:layout="@layout/fragment\_order">  
 </fragment>  
 <fragment  
 android:id="@+id/nav\_addFoods"  
 android:name="com.example.storemanager.ui.add\_meals.AddMealFragment"  
 android:label="@string/menu\_addMeal"  
 tools:layout="@layout/fragment\_add\_meal"/>  
 <fragment  
 android:id="@+id/nav\_stats"  
 android:name="com.example.storemanager.ui.statistics.StatisticFragment"  
 android:label="@string/menu\_stats"  
 tools:layout="@layout/fragment\_statistics"/>  
 <fragment  
 android:id="@+id/nav\_food\_details"  
 android:name="com.example.storemanager.ui.order.ShowFoodDetailsFragment"  
 android:label="@string/menu\_foodDetails"  
 tools:layout="@layout/fragment\_show\_food\_details"/>  
 <fragment  
 android:id="@+id/nav\_funds"  
 android:name="com.example.storemanager.ui.add\_meals.FundsFragment"  
 android:label="@string/add\_funds"  
 tools:layout="@layout/fragment\_funds"/>  
 <fragment  
 android:id="@+id/nav\_cart"  
 android:name="com.example.storemanager.ui.checkout.CheckoutFragment"  
 android:label="@string/menu\_out"  
 tools:layout="@layout/fragment\_checkout"/>  
</navigation>

The next step was to link the references to the class **NavActivity.** In its onCreate() method.

mAppBarConfiguration = new AppBarConfiguration.Builder(  
 R.id.*nav\_orders*, R.id.*nav\_addFoods*, R.id.*nav\_stats*,R.id.*nav\_funds*, R.id.*nav\_cart*)  
 .setOpenableLayout(drawer)  
 .build();

Additionally, **NavActivity** was made to implement a custom interface that allowed it to call the navigation lock and unlock feature.

@Override  
public void setDrawer\_lock() {  
 // Lock Drawer  
 drawer.setDrawerLockMode(DrawerLayout.*LOCK\_MODE\_LOCKED\_CLOSED*);  
 binding.appBarNav.toolbar.setNavigationIcon(null);  
}  
  
@Override  
public void setDrawer\_unlock() {  
 // Unlock Drawer  
 drawer.setDrawerLockMode(DrawerLayout.*LOCK\_MODE\_UNLOCKED*);  
 setSupportActionBar(binding.appBarNav.toolbar);  
}

## Implementing Order

XML files are:

* fragment\_order (for layout)
* cardview\_menu\_row (for recyclerview layout.)

The order screen displays the currently available menu which is in a card view. This is accomplished by using **DishAdapter and MealsHolder.** The bind method will call each time a new Meal is displayed in the Mealholder this will the card view to update with the desired text.

private class MealsHolder extends RecyclerView.ViewHolder implements View.OnClickListener {  
 private Meal mMeal;  
 private TextView mTextViewTitle;  
 private TextView mTextViewPrice;  
  
 public MealsHolder(LayoutInflater inflater, ViewGroup parent) {  
 super(inflater.inflate(R.layout.*cardview\_menu\_row*, parent, false));  
 itemView.setOnClickListener(this);  
 mTextViewTitle = itemView.findViewById(R.id.*tv\_meal\_title*);  
 mTextViewPrice = itemView.findViewById(R.id.*tv\_meal\_price*);  
 }  
  
 //This will be called each time a new Meal is displayed in MelHolder  
 //When given a Meal, MealHolder will now update the title TextView and price TextView to reflect the  
 //state of the Meal.  
 public void bind(Meal meal){  
 mMeal = meal;  
 mTextViewTitle.setText(mMeal.getName());  
 mTextViewPrice.setText(String.*valueOf*(mMeal.getPrice()));  
 }// bind  
  
 @Override  
 public void onClick(View view) {   
 }  
}// MealsHolder  
  
private class DishAdapter extends RecyclerView.Adapter<MealsHolder> {  
 private List<Meal> mMenu;  
  
 public DishAdapter(List<Meal> meals) {mMenu = meals;}  
  
 // inflates the row layout from xml when needed  
 @Override  
 public MealsHolder onCreateViewHolder(ViewGroup parent, int viewType) {  
 LayoutInflater layoutInflater = LayoutInflater.*from*(getActivity());  
 return new MealsHolder(layoutInflater, parent);  
 }  
  
 // binds the data into TexView for each row  
 @Override  
 public void onBindViewHolder(MealsHolder holder, int position) {  
 Meal meals = mMenu.get(position);  
 holder.bind(meals);  
 }  
  
 @Override  
 public int getItemCount() {return mMenu.size();}  
  
 public void setMenu(List<Meal> meals){mMenu = meals;}  
  
}// MealAdapter

Moreover, to implement the click event to open the **Show Details screen** from pressing an item on the list. An onClick listener was placed in **MealsHolder.**

@Override  
 public void onClick(View view) {  
 String dish\_name = mMeal.getName();  
 Toast.*makeText*(getActivity(),  
 dish\_name + " clicked!", Toast.*LENGTH\_SHORT*).show();  
  
 Bundle bundle = new Bundle();  
 bundle.putString("meal\_ID", String.*valueOf*(mMeal.getId()));

Fragment showDetails = new ShowFoodDetailsFragment();  
 showDetails.setArguments(bundle);  
 FragmentManager fragmentManager = getActivity().getSupportFragmentManager();  
 FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();  
 fragmentTransaction.replace(R.id.*nav\_host\_fragment\_content\_nav*, showDetails)  
 .addToBackStack(null)  
 .commit();  
 }  
}// MealsHolder

A bundle is needed to send the meal’s id to the **ShowDetailsFragment.**

### Implementing ShowDetails

In **ShowDetailsFragment** the bundle is retrieved in the onCreateView method. Moreover, the Navigation set drawer lock is performed to prevent screen overlay.

@Override  
public View onCreateView(LayoutInflater inflater, ViewGroup container,  
 Bundle savedInstanceState) {  
 // Inflate the layout for this fragment  
 binding = FragmentShowFoodDetailsBinding.*inflate*(getLayoutInflater());  
 amount = Integer.*parseInt*(String.*valueOf*(binding.textAmount.getText())); // Setting Amount  
  
 // Retrieve Bundle  
 Bundle bundle = this.getArguments();  
 if(bundle != null){  
 UUID dishId = UUID.*fromString*((String) bundle.get("meal\_ID"));  
 mMeal = MenuLab.*get*(getActivity()).getMeal(dishId);  
 totalBrought = mMeal.getTimesOrder(); // Gets the total number of that sold  
 }  
 binding.mealTitle.setText(mMeal.getName());  
 binding.mealPrice.setText("$"+ mMeal.getPrice());  
  
 ((NavActivity) getActivity()).setDrawer\_lock();  
 return binding.getRoot();  
}

To implement the Navigation unlock:

// To prevent memory leak  
@Override  
public void onDestroyView() {  
 super.onDestroyView();  
 ((NavActivity)getActivity()).setDrawer\_unlock();// UnLock Drawer  
 binding = null;  
}

Delete Meal:

@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
 switch (item.getItemId()) {  
 case R.id.*delete\_meal*:  
 AlertDialog.Builder delete = new AlertDialog.Builder(getContext());  
 delete.setTitle(R.string.*app\_name*);  
 delete.setMessage("Do you want to delete the meal?");  
 delete.setPositiveButton("Yes", new DialogInterface.OnClickListener() {  
 public void onClick(DialogInterface dialog, int id) {  
 dialog.dismiss();  
 MenuLab.*get*(getActivity()).deleteMeal(mMeal);  
 getActivity().onBackPressed();  
 }  
 });  
 delete.setNegativeButton("No", new DialogInterface.OnClickListener() {  
 public void onClick(DialogInterface dialog, int id) {  
 dialog.dismiss();  
 }  
 });  
 AlertDialog alertDelete = delete.create();  
 alertDelete.show();  
 return true;  
 default:  
 return super.onOptionsItemSelected(item);  
 }  
}

Meal Counter: These are the orange buttons that track the number of goods ordered of that type.

// add another instance  
binding.plusBtn.setOnClickListener(new View.OnClickListener(){  
 @Override  
 public void onClick(View view) {  
 amount += 1;  
 binding.textAmount.setText(String.*valueOf*(amount));  
 }  
});  
  
// remove an instance  
binding.minusBtn.setOnClickListener(new View.OnClickListener(){  
 @Override  
 public void onClick(View view) {  
 amount -= 1;  
 binding.textAmount.setText(String.*valueOf*(amount));  
 }  
});

Add Cart: Insterts the ordered food into a checkout list via **OrderManager.**

binding.addCartBtn.setOnClickListener(new View.OnClickListener() {  
 OrderManager orderManager = OrderManager.*get*(getActivity());  
 @Override  
 public void onClick(View view) {  
 totalBrought += amount;  
 mMeal.setTimesOrder(totalBrought);  
 //Set cart  
 mCart = new CartFood(mMeal.name, mMeal.price, amount);  
 orderManager.insertFood(mCart);  
 Toast.*makeText*(getActivity(),"Total sold: " + mMeal.getTimesOrder(),Toast.*LENGTH\_LONG*).show();  
 }  
});

### OrderManager

This class stores the temporary data for the customer's requested products.

public class OrderManager {  
 private static final DecimalFormat *df* = new DecimalFormat("0.00");  
 private static OrderManager *sOrder*;  
 public double totalMealPrice = 0.00;  
 public ArrayList<CartFood> mCart;  
  
 public static OrderManager get(Context context) {  
 if (*sOrder* == null) {  
 *sOrder* = new OrderManager(context);  
 }  
 return *sOrder*;  
 }  
  
 private OrderManager(Context context) {mCart = new ArrayList<>(); }  
  
 public List<CartFood> getCartItems() {return mCart;}  
  
 public void destroy(){mCart.clear();}  
 public double getTotalCartPrice(List<CartFood> c) {  
 double fee = 0.00;  
 for (CartFood food : c) {  
 fee += food.getTotal();  
 }  
 return Double.*parseDouble*(*df*.format(fee));  
 }// calculateTotalMealPrice  
  
  
 public void insertFood(CartFood item) { mCart.add(item); }  
  
 public void setTotalCartPrice(double i) { totalMealPrice = i; }  
}

## Implementing Add-Meals

Takes the name and price and saves them into the Menu table.

@Override  
public View onCreateView(LayoutInflater inflater, ViewGroup container,  
 Bundle savedInstanceState) {  
 binding = FragmentAddMealBinding.*inflate*(getLayoutInflater());  
  
  
 //Button Click Saves meal  
 binding.sendBtn.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 newMeal = new Meal();  
 dish = String.*valueOf*(binding.dishName.getText());  
 price = String.*valueOf*(binding.price.getText());  
 if (dish.isEmpty() || price.isEmpty()){  
 Toast.*makeText*(getActivity(),"Complete Forum",Toast.*LENGTH\_LONG*).show();  
 }  
 else{  
 newMeal.setName(dish); // Meal Name  
 newMeal.setPrice(Double.*parseDouble*(price)); // Meal Price  
 MenuLab.*get*(getActivity()).addMeal(newMeal);  
 Toast.*makeText*(getActivity(),"SAVED ",Toast.*LENGTH\_LONG*).show();  
 binding.dishName.setText("");  
 binding.price.setText("");  
 }  
 }  
 });  
 return binding.getRoot();  
}

## 

## Implementing Add-Funds

This screen takes the spending and the date that the user used during a month time period and store it, so that when it coming to calculating the profits it will be total earn minus total spent.

Sadly, this function has yet to be implemented and due to it not being implemented the statistic screen will not function as it should.

## 

## Implementing Checkout

XML files are:

* fragment\_checkout (for layout)
* cardview\_checkout (for recyclerview layout.)

The **CheckoutFragment** contains 2 other classes which handle the Recyclerview items similar to **OrderFragment**, **CartAdapter,** and **CartHolder**.

To display the dynamic textview in the Checkout screen, **CartHolder and its function bind** were used.

private class CartHolder extends RecyclerView.ViewHolder {  
 private CartFood mCart;  
 private TextView mTextViewTitle;  
 private TextView mTextViewPrice;  
 private TextView mTextViewAmount;  
  
 public CartHolder(LayoutInflater inflater, ViewGroup parent) {  
 super(inflater.inflate(R.layout.*cardview\_checkout*, parent, false));  
 mTextViewTitle = itemView.findViewById(R.id.*cart\_meal\_title*);  
 mTextViewAmount = itemView.findViewById(R.id.*cart\_amount*);  
 mTextViewPrice = itemView.findViewById(R.id.*cart\_meal\_price*);  
 }  
  
 public void bind(CartFood items){  
 mCart = items;  
 mTextViewTitle.setText(mCart.getCart\_name());  
 mTextViewAmount.setText(String.*valueOf*(mCart.getCart\_amount()));  
 mTextViewPrice.setText(String.*valueOf*(mCart.getTotal()));  
  
 }// bind  
}// MealsHolder

Saves the total save (which is equivalent to money earned) to **MonthlyBussinessReview’s** setTotalEarned method

public void on view created(@NonNull View view, @Nullable Bundle saved instance state) {  
 super.onViewCreated(view, savedInstanceState);  
  
 // Saves total cart money and clear screen  
 binding.checkOutBtn.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 MonthlyBussinessReview saveTotal = new MonthlyBussinessReview();  
 saveTotal.setTotalEarned(grandTotal);  
 orderManager.destroy();  
 binding.tvTotal.setText("$0");  
 mCartRecyclerView.setAdapter(null);  
 }  
 });  
}

## Implementing Statistics

public View onCreateView(LayoutInflater inflater, ViewGroup container,  
 Bundle savedInstanceState) {  
 View view = inflater.inflate(R.layout.*fragment\_statistics*, container, false);  
  
 // Spinner element  
 Spinner spinner = view.findViewById(R.id.*spinner*);  
  
 // Spinner click listener  
 spinner.setOnItemSelectedListener(this);  
  
 // Spinner Drop down elements  
 List<String> categories = new ArrayList<>();  
 categories.add("January");  
 categories.add("February");  
 categories.add("March");  
 categories.add("April");  
 categories.add("May");  
 categories.add("June");  
 categories.add("July");  
 categories.add("August");  
 categories.add("September");  
 categories.add("November");  
 categories.add("December");  
  
 // Creating adapter for spinner  
 ArrayAdapter<String> dataAdapter = new ArrayAdapter<>(getActivity(), android.R.layout.*simple\_spinner\_item*, categories);  
  
 // Drop down layout style - list view with radio button  
 dataAdapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);  
  
 // attaching data adapter to spinner  
 spinner.setAdapter(dataAdapter);

This block of code setups the date spinner on the statistic screen, so that the user can pick a date to find the total profits for that month.

double x = monthlyBussinessReview.getTotalEarned();  
double y = monthlyBussinessReview.getSpent();  
calculate = x - y ;  
  
// Will display the profits when click  
button.setOnClickListener(v -> {  
 textView.setText((int) calculate);  
  
});

The code above takes the total earn and spent then subtract them and store them into calculate, after that is done the button is set to display calculate in the taxt field on click.