4/23/2020

Socket Shopping App

PROG8480 Course Final Project Documentation

Team:

1. Hanan Younes (8684560)
2. Leo Carvalho Ramos Antunes (8642805)
3. Oleh Kosenko (8644350)
4. Xiaolong Li (8659081)

Table of Contents

1. Executive Summary:

The Socket App is an Android shopping application that provides an ultimate shopping experience for tech savvy users looking for the latest electronic products and gadgets. The Socket application allows users to:

- Shop electronic products anytime, anywhere
- Browse a built-in inventory of products ranging from laptops, phones and accessories
- Search for a specific product in the inventory of available stores
- Find the nearby Socket stores based on their location
- Socket stores contact information, and directions

2. Milestones & Timeline:

This project is a chance for the team to combine and practice some of the important concepts that have been introduced in the Android Class. The main objective is making a shopping application, Socket, to track a store's inventory and provide an easy shopping experience. The main components are

- Persist customer data using an SQLite database
- Keep track of the store's inventory of products using an SQLite database
- Find nearby Socket locations by working with Google Maps and the GPS functionality of the phone.
- Create classes to practice Object Oriented Programming principles, such as encapsulation, to protect the member variable and using getter and setter methods when needed.
- Follow the material design guidelines to make an overall appealing user interface with all graphical elements conveying a consistent message and eye-catching layout.
- Practice working and collaborating remotely between team members using a Version Control tool. Our team used Azure Devops as our collaborative workspace.
- Follow best coding practices by
    1. Choosing representative variable names, readable code,
    2. Validating and preventing runtime exceptions
    3. Creating methods for code reusability
    4. Adding comments for more code readability and in-code documentation

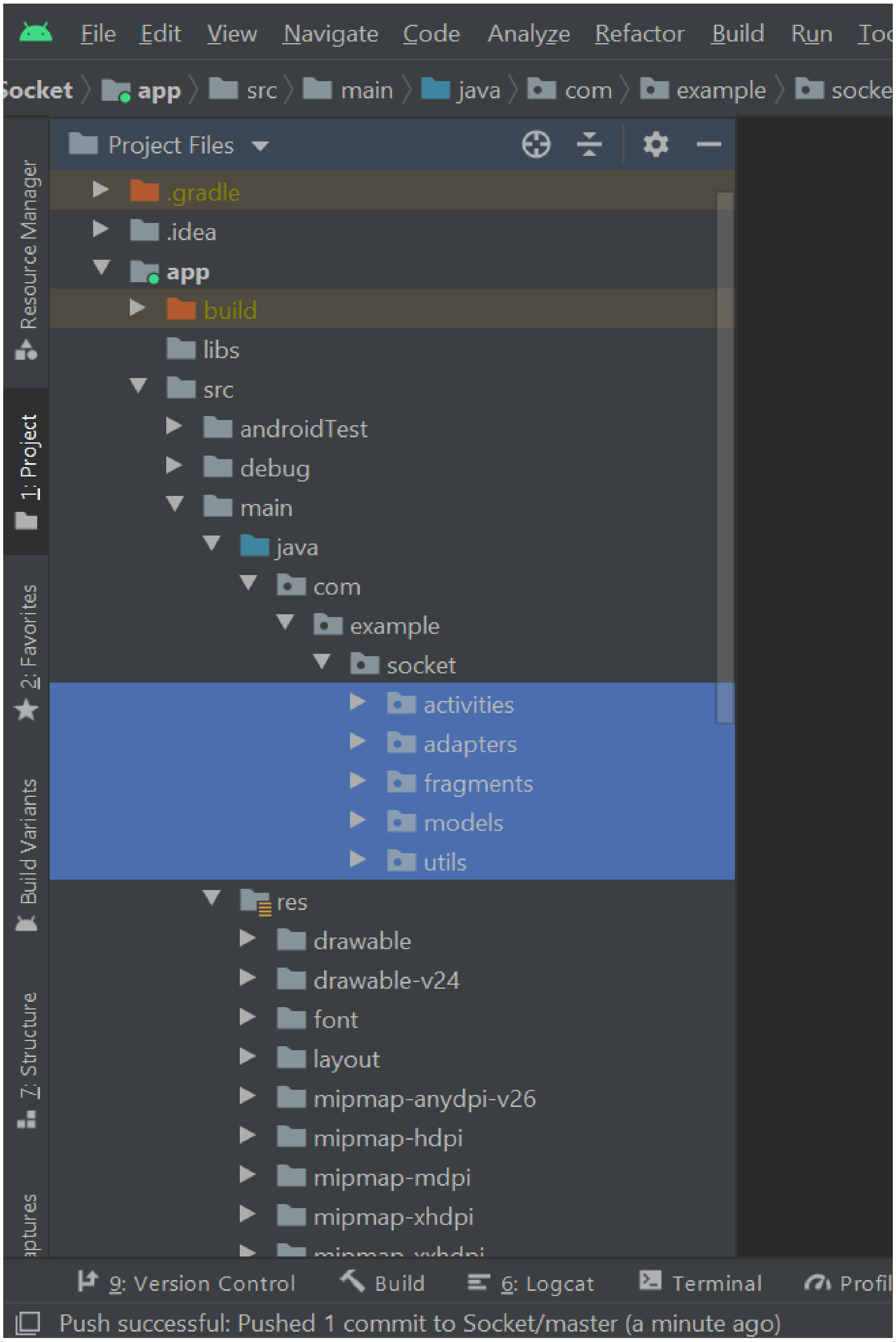
3. Team Work Items:

Team member	Work Item	Due Date
Hanan	1. Initial project set up with required classes 2. Version control set up 3. Calculate the nearest store location using the Haversine method 4. Project Documentation	Apr 18th, 2020
Leo	1. Final project set up 2. Implement all database and helper classes in the util/db folders 3. Creating a test suite for the project	
Oleh	1. Controllers for all fragments. 2. Calculate the nearest store location using the Haversine method	
Xiaolong	1. UI - Layouts creation for all activities and screens 2. Adding the Google Maps and store locator functionality	

4. **External Classes Used:** Please note that all implemented classes use getter and setter methods to preserve OOP principles.
   - **InventoryItem Class:** to keep track of products related information including product name, code, description, price, etc.
   - **Store Class:** to maintain store related information such as store id, full address, phone number, latitude and longitude.
   - **Province Class:** to track province name, id, and code. The province list is read from the database.
   - **Purchase Class:** to hold purchases' related data such as purchase date, total price, item sold, along with customer name and province.
5. **Extra Features:** As a team, we have implemented extra non-trivial features to enhance our applications' shopping experience including:
   - Maintaining some of the customer's data using SQLite database such as name, province and data of purchase.
   - Working with the Google Maps and the GPS functionality of the phone to display different store locations to users.
   - Using multiple Fragments to create reusable code
   - Implementing test suite to test the app under the AndroidTest folder
   - Organizing the project code into folders based on their functionality (see Figure 1.)

6. Project Structure Details:

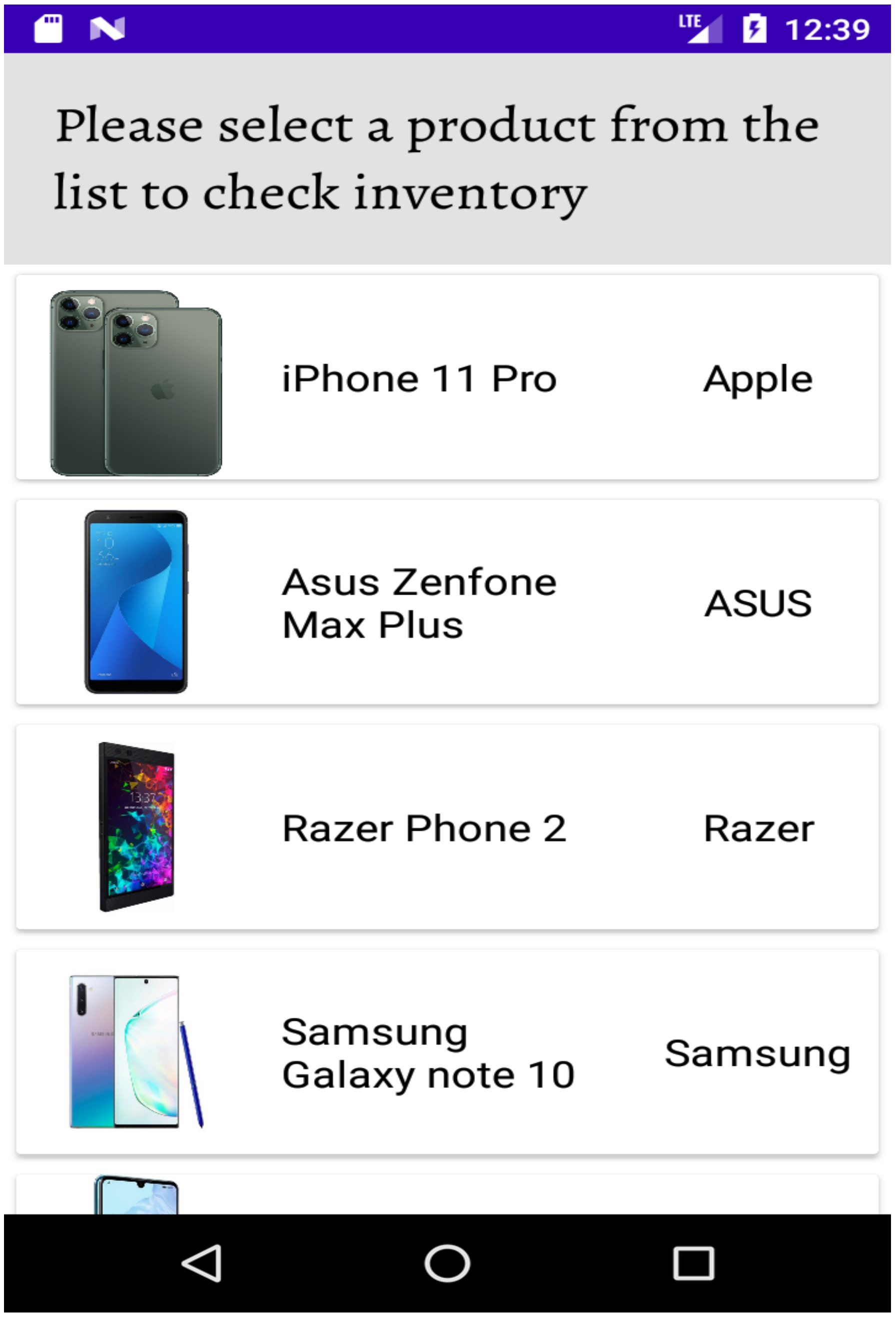
1. **Activities Folder:** this folder contains the MainActivity Class and the MapsActivity Class.
2. **Adapters Folder:** this folder contains two list adapters, storeListAdapter and ItemListAdapter, to populate the items inventory and store lists with data. The two adaptors are used with StoreFragment and ItemListFragment respectively.
3. **Fragment Folder:** this folder contains three implemented fragments, InventoryFragment, StoreFragment and ItemListFragment. The purpose of using fragments is code reusability and readability.
4. **Models Folder:** this folder contains classes used in the Socket application. Please refer to section 4 for more details about these classes.
5. **Utils/db Folders:** This folder is holding database helper and utility classes. It is further divided into:
   a. **Executer Folder:** This folder contains 5 database related classes, one of them is an abstract class, **SocketDbExecutor**, the defines the blueprint of methods to be used when retrieving data from the database. The other 4 database helper classes, which are mapped to the external classes detailed in section 4, implement these abstract methods in a way tailored to their own context and individual requirements.
   b. **Location Folder:** the location folder includes three database related classes, two of them are final: **SocketContract** Class to represents database contract with table names and columns; **SocketDbDefinition** Class which is an Inner class that defines InventoryItem, Store, Province, Purchase tables definition scripts. The last class is the **SocketDbhelper Class** which is a helper class to retrieve and save data to an SQLite database.
6. **Layout Folder:** This folder contains 11 layout xml files that have been used to build the different components of the application's UI.
7. **Res Folder:** this folder includes strings, colors, themes, and custom fonts we used in implementing the Socket application.

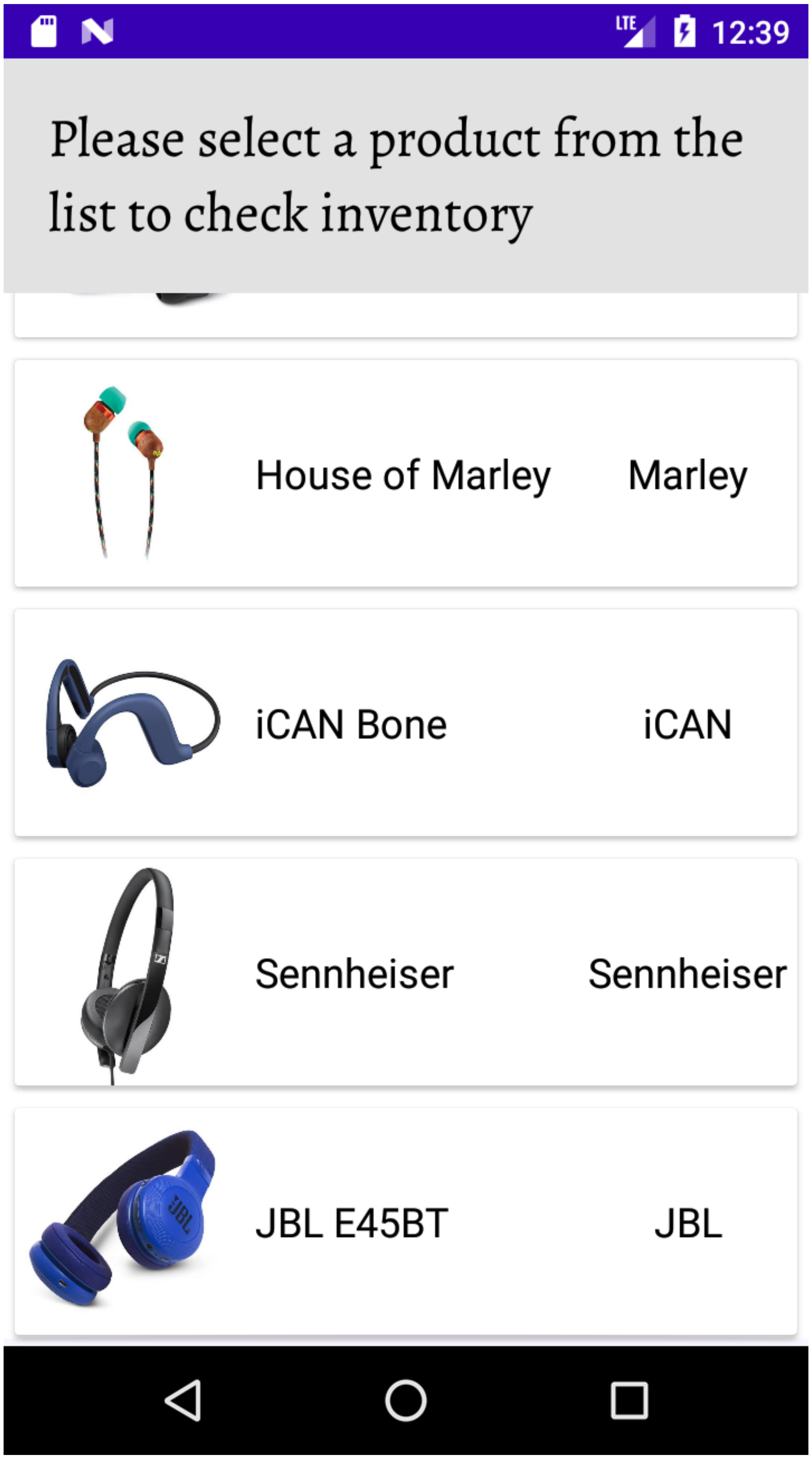


Push successful: Pushed 1 commit to Socket/master (a minute ago)

Figure 1. Project Folders

7. **Detailed Screenshots of the Application:** In this part, screenshots are added based on their sequence of appearance assuming a given user story. Also, toast messages are displayed whenever the user makes a selection.
    a. **Landing Screen (2 screenshots):** First the user launches the application, and a screen featuring a scrollable list of electronic products and accessories that are available in inventory. The user is advised to choose a product from the list. Once an item is selected, the user moves to the next screen that features the store locator.



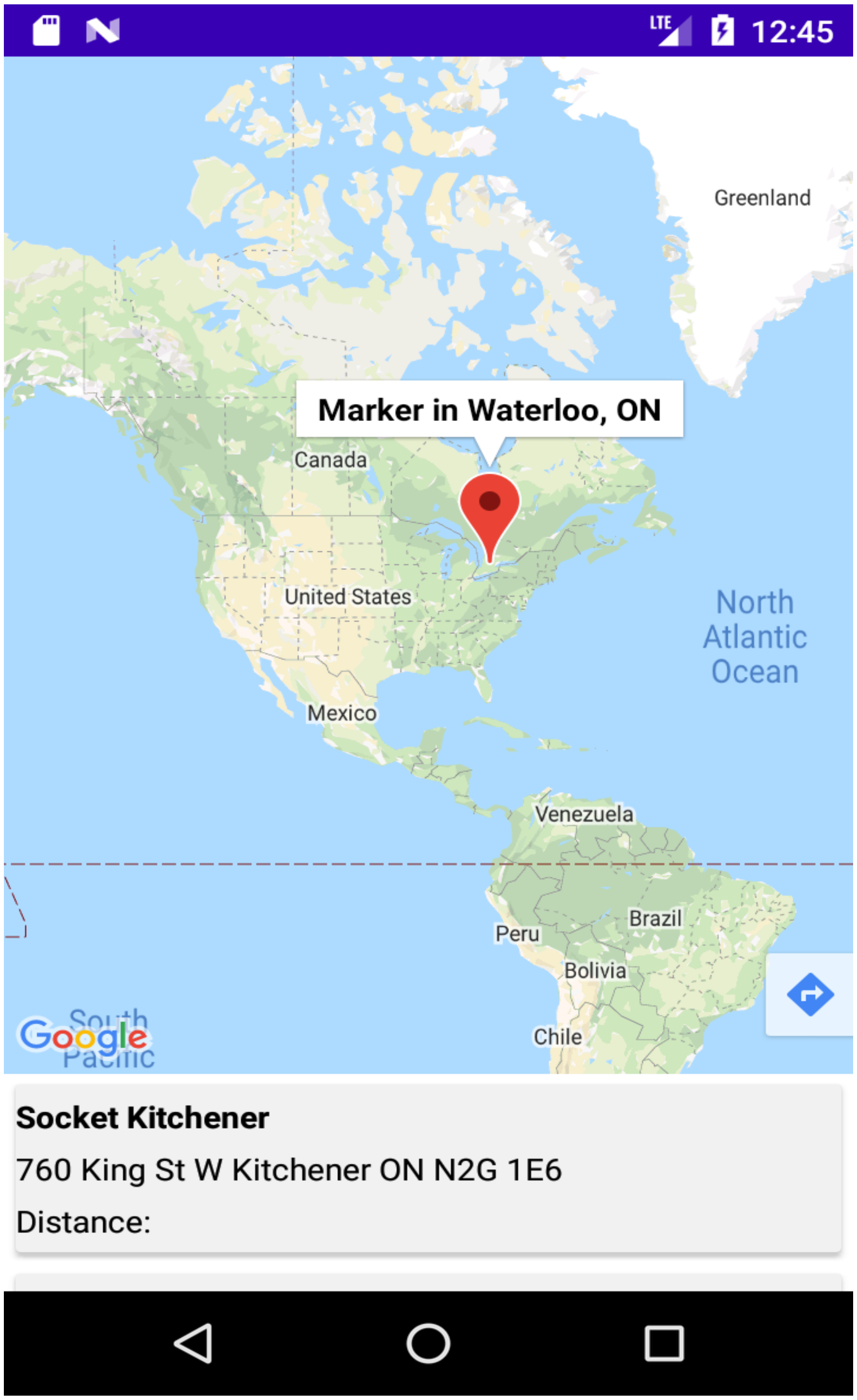


12:39

Please select a product from the list to check inventory

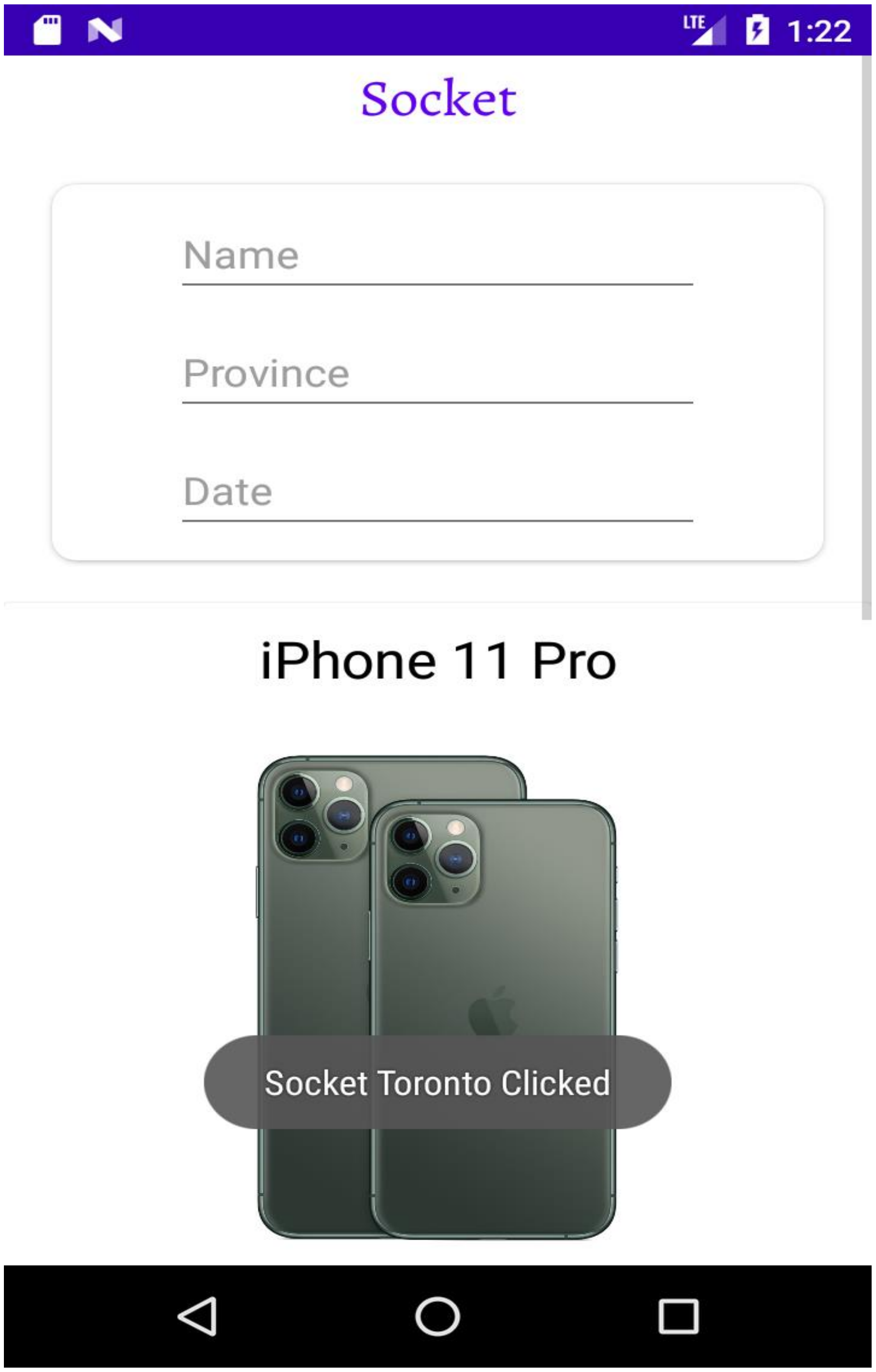
	House of Marley	Marley
	iCAN Bone	iCAN
	Sennheiser	Sennheiser
	JBL E45BT	JBL

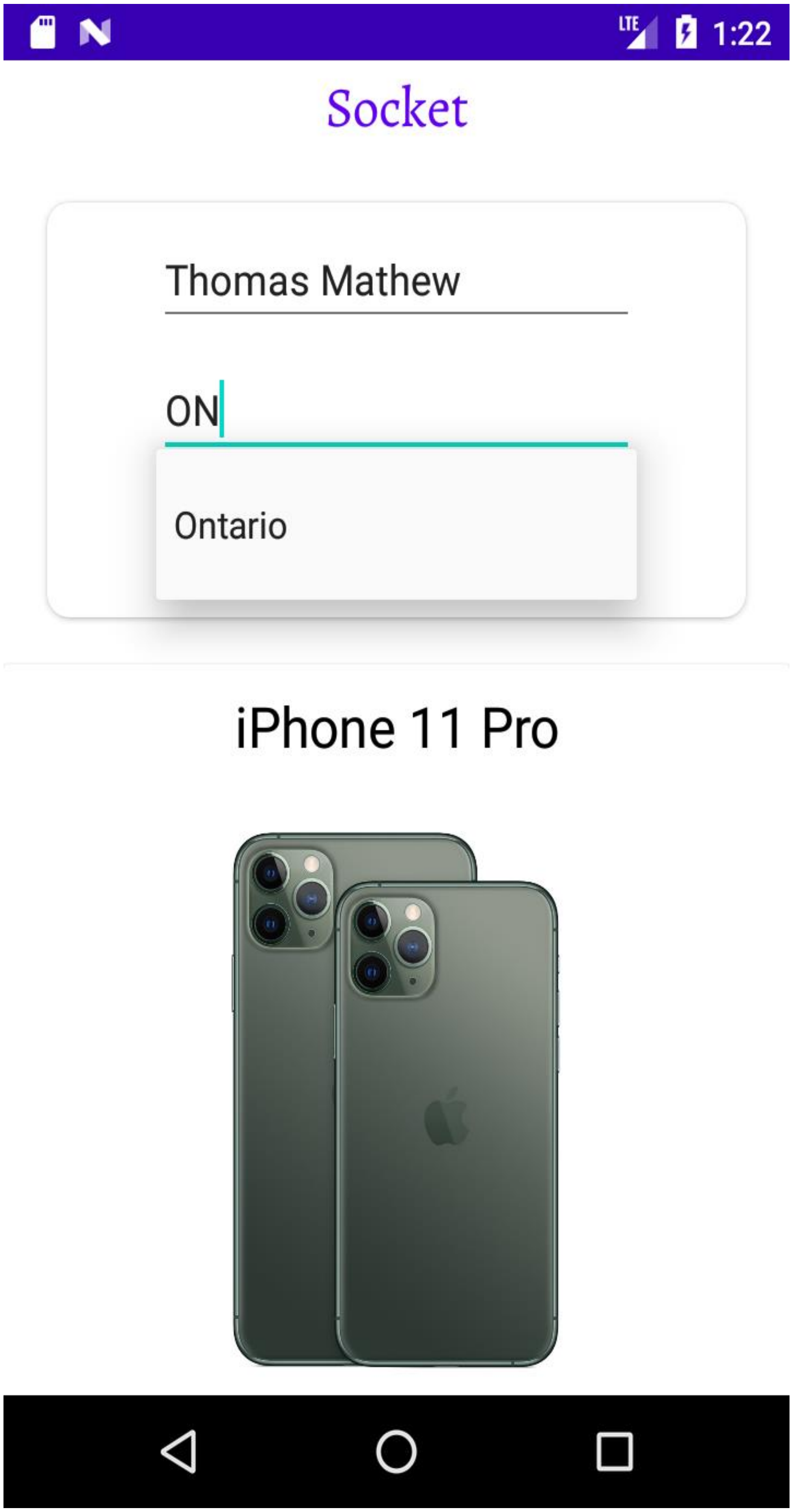
b. **Map Screen:** The user can see world map with a default location as the main Socket store location at Waterloo, On. At the bottom of the map, a list of store locations is presented along with their Distance to the user's current location.



Socket Kitchener
760 King St W Kitchener ON N2G 1E6
Distance:

c. **Shopping Screen:** The user selects a store to be directed to the store's online shopping screen to fill in their information, and to choose if any extra peripherals or accessories are needed. Once all information is filled in, an invoice will be displayed with the subtotal including taxes (Ontario taxes). The user can pick up the items purchased at the nearest Socket store selected using the store location list. Last, the user can go back to shop the inventory again by clicking on homepage button.





1:22

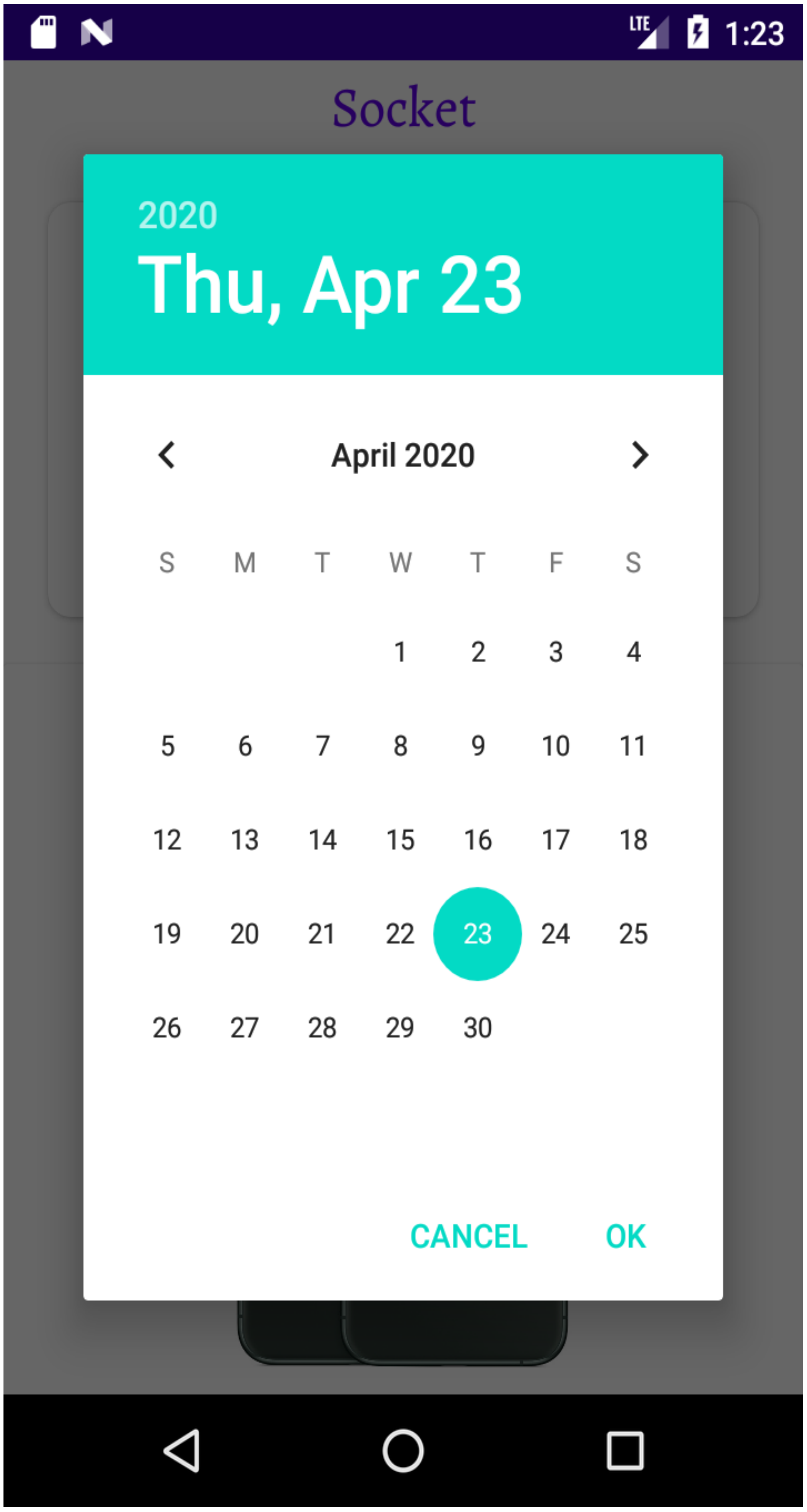
Socket

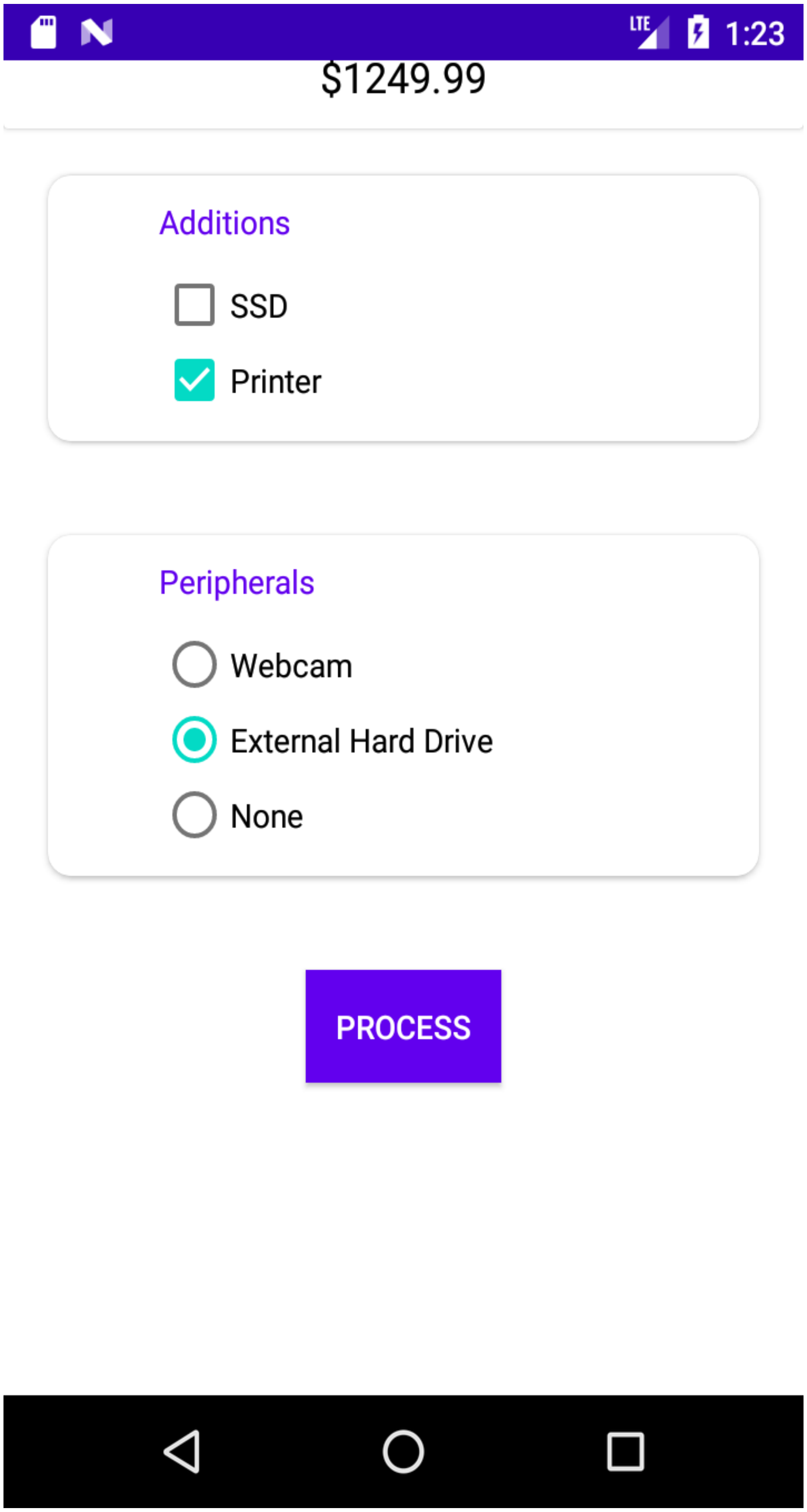
Thomas Mathew

ON

Ontario

iPhone 11 Pro


1:23
Socket
2020
Thu, Apr 23
April 2020
S M T W T F S
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
CANCEL
OK



$1249.99

Additions

☐ SSD

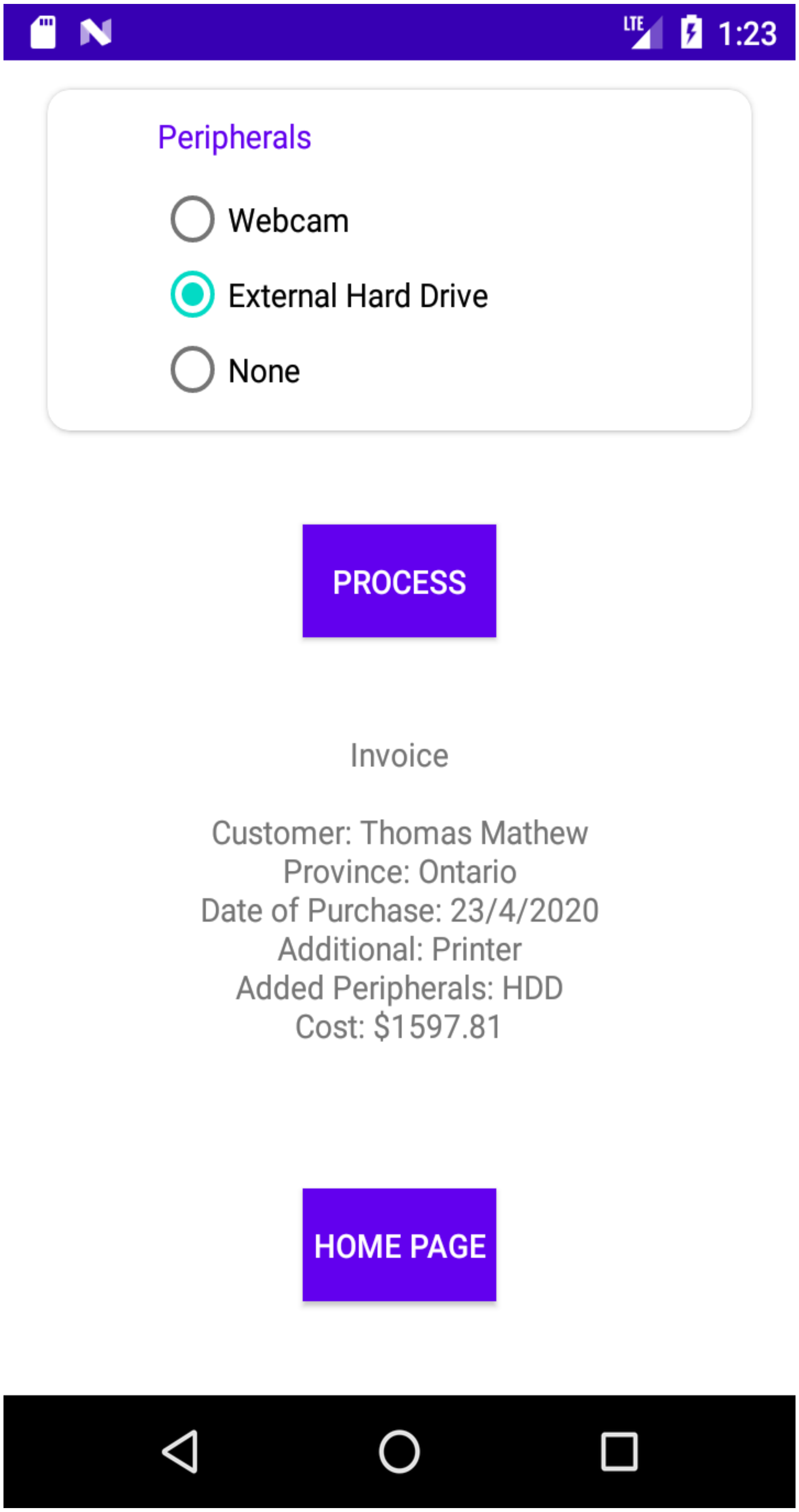
☑ Printer

Peripherals

○ Webcam

● External Hard Drive

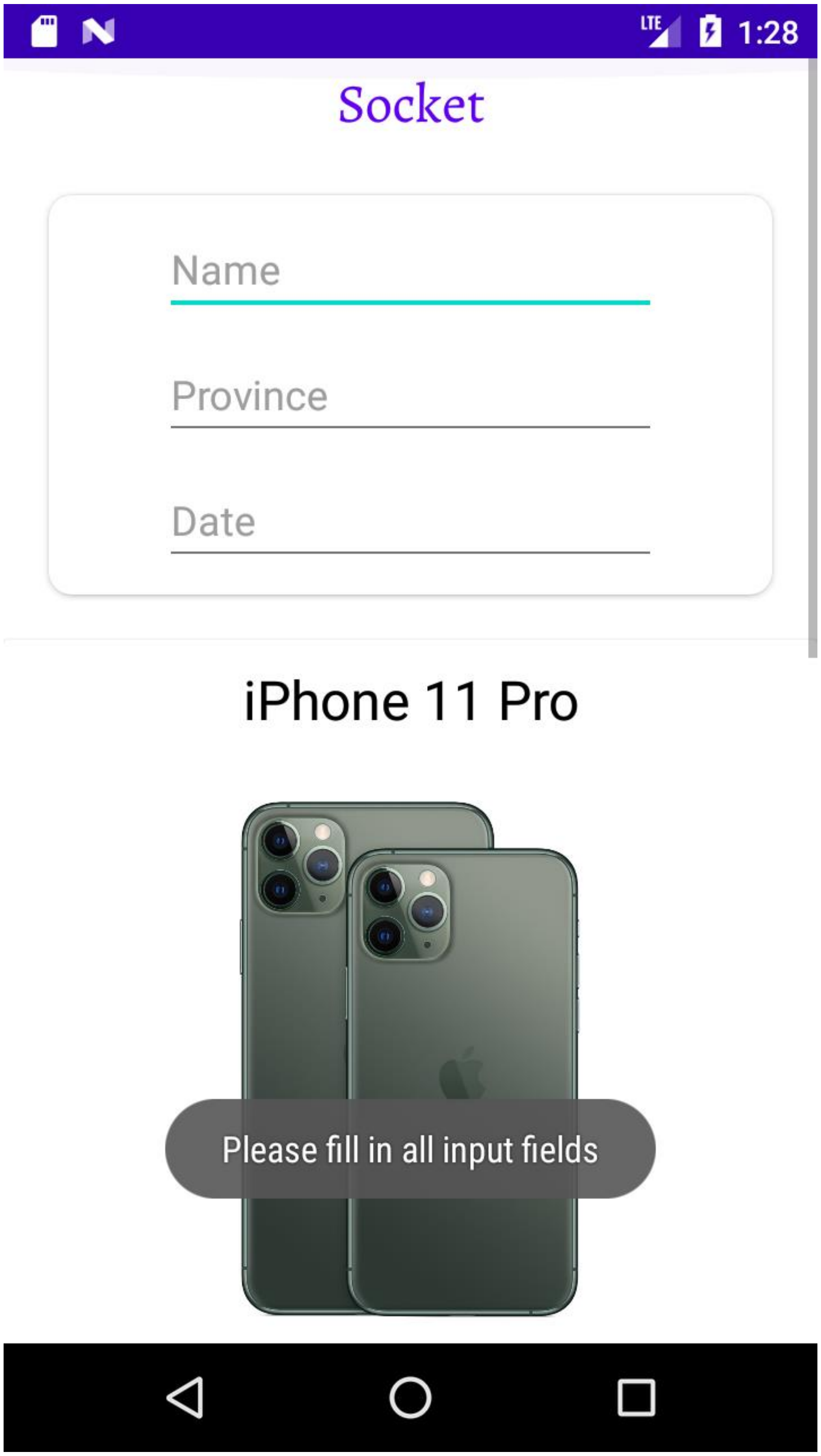
○ None



Invoice

Customer: Thomas Mathew
Province: Ontario
Date of Purchase: 23/4/2020
Additional: Printer
Added Peripherals: HDD
Cost: $1597.81

- If any of the fields is missing; the user is notified to fill in all fields.



Please fill in all input fields

8. Challenges:

The main challenge our team faces was the current unusual COVID-19 situation that obligated us to practice social distancing and collaborate remotely using a version control tool, Azure Devops. Being unable to meet face to face increased the collaboration and coordination costs as we spent more time and energy to coordinate schedules, arrange appropriate virtual meeting times, brainstorm, reach agreements, make decisions together as a team, integrate the contributions of group members, and still work on other courses' assignments and projects. The time spent on each of these tasks added together is significant. However, working remotely was a great opportunity to cope with the current circumstances, hone our version control skills, and develop our multitasking skills.

9. Application Code:

- **MainActivity:**

```java
package com.example.socket.activities;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;

import android.os.Bundle;

import com.example.socket.fragments.ItemListFragment;
import com.example.socket.R;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        loadFragment(new ItemListFragment());
    }

    private void loadFragment(Fragment fragment) {
        FragmentManager fm = getSupportFragmentManager();
        FragmentTransaction ft = fm.beginTransaction();
        ft.replace(R.id.frameLayout, fragment);
        ft.commit();
    }
}
```

- **MapsActivity:**

```
package com.example.socket.activities;
import androidx.fragment.app.FragmentActivity;
import android.os.Bundle;

import com.example.socket.R;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified when the map is ready to be
used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }

    /**
     * Manipulates the map once available.
     * This callback is triggered when the map is ready to be used.
     * This is where we can add markers or lines, add listeners or move the camera.
In this case,
     * we just add a marker near Sydney, Australia.
     * If Google Play services is not installed on the device, the user will be
prompted to install
     * it inside the SupportMapFragment. This method will only be triggered once the
user has
     * installed Google Play services and returned to the app.
     */
    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        // Add a marker in Sydney and move the camera
        LatLng sydney = new LatLng(-34, 151);
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in
Sydney"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
}
```

- **SocketInstrumentedTest:**

```
package com.example.socket;

import android.content.Context;

import androidx.test.platform.app.InstrumentationRegistry;
import androidx.test.ext.junit.runners.AndroidJUnit4;

import com.example.socket.models.InventoryItem;
import com.example.socket.models.Province;
import com.example.socket.models.Purchase;
import com.example.socket.models.Store;
import com.example.socket.utils.db.SocketDbHelper;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;

import java.util.Date;
import java.util.List;

import static org.hamcrest.Matchers.is;
import static org.hamcrest.Matchers.not;
import static org.hamcrest.Matchers.notNullValue;
import static org.junit.Assert.*;

/**
 * Instrumented test, which will execute on an Android device.
 *
 * @see <a href="http://d.android.com/tools/testing">Testing documentation</a>
 */
@RunWith(AndroidJUnit4.class)
public class SocketInstrumentedTest {
    // DbHelper instance declaration
    private SocketDbHelper dbHelper;

    @Before
    public void setup() {
        // Context of the app under test.
        Context appContext =
InstrumentationRegistry.getInstrumentation().getTargetContext();
        // Initialize Socket Database Helper
        dbHelper = new SocketDbHelper(appContext);

        List<Purchase> purchases = dbHelper.getAllPurchases();
        for (Purchase p : purchases) {
            dbHelper.removePurchase(p.getId());
        }
    }

    @Test
    public void testGetAllProvinces() {
        List<Province> provinces = dbHelper.getAllProvinces();
```

```java
        assertThat(provinces.size(), is(13));
        assertThat(provinces.get(0).getCode(), is("AL"));
    }

    @Test
    public void testGetProvinceByCode() {
        Province province = dbHelper.getProvinceByCode("ON");
        assertThat(province, notNullValue());
        assertThat(province.getCode(), is("ON"));
        assertThat(province.getName(), is("Ontario"));
    }

    @Test
    public void testGetAllStores() {
        List<Store> stores = dbHelper.getAllStores();
        assertThat(stores.size(), is(3));
        assertThat(stores.get(0).getCode(), is("SK"));
        assertThat(stores.get(1).getCode(), is("ST"));
        assertThat(stores.get(2).getCode(), is("SW"));
    }

    @Test
    public void testGetStoreByCode() {
        Store store = dbHelper.getStoreByCode("ST");
        assertThat(store, notNullValue());
        assertThat(store.getCode(), is("ST"));
        assertThat(store.getName(), is("Socket Toronto"));
        assertThat(store.getLocationLatitude(), is(43.6541737));
        assertThat(store.getLocationLongitude(), is(-79.38081165));
    }

    @Test
    public void testAllInventoryItems() {
        List<InventoryItem> inventoryItems = dbHelper.getAllInventoryItems();
        assertThat(inventoryItems.size(), is(18));
        assertThat(inventoryItems.get(0).getCode(), is("SKP1"));
    }

    @Test
    public void testInventoryItemsByStoreId() {
        Store store = dbHelper.getStoreByCode("ST");
        List<InventoryItem> inventoryItems =
                dbHelper.getInventoryItemsByStoreId(store.getId());

        assertThat(inventoryItems.size(), is(6));
        assertThat(inventoryItems.get(0).getCode(), is("STP1"));
    }

    @Test
    public void testGetInventoryItemByCode() {
        InventoryItem inventoryItem = dbHelper.getInventoryItemByCode("STP1");
        assertThat(inventoryItem, notNullValue());
        assertThat(inventoryItem.getCode(), is("STP1"));
        assertThat(inventoryItem.getName(), is("MSI Gaming Notebook"));
        assertThat(inventoryItem.getPrice(), is(1299.99));
    }

    @Test
    public void testGetPurchases() {
        List<Purchase> purchases = dbHelper.getAllPurchases();
        assertThat(purchases.size(), is(0));
    }

    @Test
    public void testPurchaseItem() {
        Province province = dbHelper.getProvinceByCode("ON");
        InventoryItem inventoryItem = dbHelper.getInventoryItemByCode("STP1");
        String customerName = "John Smith";
        double calculatedTotalPrice = inventoryItem.getPrice(); // simplified

        Purchase result = dbHelper.insertPurchase(customerName, province.getCode(),
                new Date().toString(), calculatedTotalPrice, inventoryItem.getId());

        Purchase purchase = dbHelper.getPurchaseByCode(result.getCode());
        assertThat(purchase, notNullValue());
        assertThat(purchase.getCode(), is("P1"));
        assertThat(purchase.getCustomerName(), is(customerName));
        assertThat(purchase.getCustomerProvinceCode(), is(province.getCode()));
        assertThat(purchase.getTotalPrice(), is(calculatedTotalPrice));
    }
}
```

-