```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.app.Activity
8  import android.content.Context
9  import android.content.Intent
10 import android.graphics.drawable.AnimatedVectorDrawable
11 import android.os.Bundle
12 import android.support.annotation.IntDef
13 import android.support.annotation.NonNull
14 import android.support.v4.content.ContextCompat
15 import android.support.v7.widget.LinearLayoutManager
16 import android.support.v7.widget.RecyclerView
17 import android.support.v7.widget.RecyclerView.ViewHolder
18 import android.text.TextUtils
19 import android.text.format.DateUtils
20 import android.view.LayoutInflater
21 import android.view.View
22 import android.view.ViewGroup
23 import android.widget.ImageView
24 import android.widget.ProgressBar
25 import android.widget.TextView
26 import com.bumptech.glide.load.resource.drawable.DrawableTransitionOptions.withCrossFade
27 import com.bumptech.glide.request.RequestOptions
28 import io.pergasus.R
29 import io.pergasus.api.DataLoadingSubject
30 import io.pergasus.api.FollowerListable
31 import io.pergasus.api.FollowersDataManager
32 import io.pergasus.data.Follow
33 import io.pergasus.data.Shop
34 import io.pergasus.ui.recyclerview.SlideInItemAnimator
35 import io.pergasus.ui.widget.BottomSheet
36 import io.pergasus.ui.widget.CircularImageView
37 import io.pergasus.util.AnimUtils.getLinearOutSlowInInterpolator
38 import io.pergasus.util.HtmlUtils
39 import io.pergasus.util.bindView
40 import io.pergasus.util.glide.GlideApp
41 import java.text.NumberFormat
42 import java.util.*
43
44
45 class UserSheet : Activity() {
46     companion object {
47         private const val MODE_FOLLOWERS = 2
48         private const val DISMISS_DOWN = 0
49         private const val DISMISS_CLOSE = 1
50         private const val EXTRA_MODE = "EXTRA_MODE"
51         private const val EXTRA_SHOP = "EXTRA_SHOP"
52
53         private const val TYPE_PLAYER = 7
54         private const val TYPE_LOADING = -1
55
56 ----------------------------------------------------------------------------------------------------
57         fun start(@NonNull launching: Activity, shop: Shop) {
58             val starter = Intent(launching, UserSheet::class.java)
59             starter.putExtra(EXTRA_MODE, MODE_FOLLOWERS)
60             starter.putExtra(EXTRA_SHOP, shop)
61             launching.startActivity(starter/*,
62 ------------------    ActivityOptions.makeSceneTransitionAnimation(launching).toBundle()*/)
63         }
64     }
65
66     private var dismissState = DISMISS_DOWN
```

```
67
68      private var shop: Shop? = null
69      private var dataManager: FollowersDataManager? = null
70      private var adapter: FollowerAdapter<Follow>? = null
71      private var layoutManager: LinearLayoutManager? = null
72
73      private val bottomSheet: BottomSheet by bindView(R.id.bottom_sheet)
74      private val content: ViewGroup by bindView(R.id.bottom_sheet_content)
75      private val titleBar: ViewGroup by bindView(R.id.title_bar)
76      private val close: ImageView by bindView(R.id.close)
77      private val title: TextView by bindView(R.id.title)
78      private val grid: RecyclerView by bindView(R.id.item_list)
79      private var largeAvatarSize: Int = 0
80
81      @kotlin.annotation.Retention(AnnotationRetention.SOURCE)
82      @IntDef(MODE_FOLLOWERS)
83      internal annotation class PlayerSheetMode
84
85      override fun onCreate(savedInstanceState: Bundle?) {
86          super.onCreate(savedInstanceState)
87          setContentView(R.layout.activity_sheet)
88
89          largeAvatarSize = resources.getDimension(R.dimen.large_avatar_size).toInt()
90
91          val intent = intent
92          @PlayerSheetMode val mode = intent.getIntExtra(EXTRA_MODE, -1)
93          when (mode) {
94              MODE_FOLLOWERS -> {
95                  shop = intent.getParcelableExtra(EXTRA_SHOP)
96                  title.text = resources.getQuantityString(
97                          R.plurals.followers,
98                          shop?.followers_count?.toInt()!!,
99                          NumberFormat.getInstance().format(shop?.followers_count!!))
100                 dataManager = object : FollowersDataManager(this@UserSheet, shop!!) {
101                     override fun onDataLoaded(data: List<Follow>) {
102                         adapter?.addItems(data)
103                     }
104
105                 }
106             }
107             else -> throw IllegalArgumentException("Unknown launch mode.")
108         }
109
110         bottomSheet.registerCallback(object : BottomSheet.Callbacks() {
111             override fun onSheetDismissed() {
112                 finishAfterTransition()
113             }
114
115             override fun onSheetPositionChanged(sheetTop: Int, userInteracted: Boolean) {
116                 if (userInteracted && close.visibility != View.VISIBLE) {
117                     close.visibility = View.VISIBLE
118                     close.alpha = 0f
119                     close.animate()
120                             .alpha(1f)
121                             .setDuration(400L)
122                             .setInterpolator(getLinearOutSlowInInterpolator(this@UserSheet))
123                             .start()
124                 }
125                 if (sheetTop == 0) {
126                     showClose()
127                 } else {
128                     showDown()
129                 }
130             }
131         })
132
133         layoutManager = LinearLayoutManager(this)
```

```
134         grid.layoutManager = layoutManager
135         grid.itemAnimator = SlideInItemAnimator()
136         adapter = FollowerAdapter(this@UserSheet)
137         dataManager?.registerCallback(adapter!!)
138         grid.adapter = adapter
139         grid.addOnScrollListener(titleElevation)
140         dataManager?.loadData() // kick off initial load
141
142     }
143 ____________________________________________________________________________
144     override fun onDestroy() {
145         dataManager?.cancelLoading()
146         super.onDestroy()
147     }
148 ____________________________________________________________________________
149     private val titleElevation: RecyclerView.OnScrollListener = object : RecyclerView.OnScrollListener() {
150         override fun onScrolled(recyclerView: RecyclerView?, dx: Int, dy: Int) {
151             val raiseTitleBar: Boolean = dy > 0 || grid.computeVerticalScrollOffset() != 0
152             titleBar.isActivated = raiseTitleBar // animated via a StateListAnimator
153         }
154     }
155 ____________________________________________________________________________
156     private fun showClose() {
157         if (dismissState == DISMISS_CLOSE) return
158         dismissState = DISMISS_CLOSE
159         val downToClose: AnimatedVectorDrawable =
160                 ContextCompat.getDrawable(this, R.drawable.avd_down_to_close) as AnimatedVectorDrawable
161         close.setImageDrawable(downToClose)
162         downToClose.start()
163     }
164 ____________________________________________________________________________
165     private fun showDown() {
166         if (dismissState == DISMISS_DOWN) return
167         dismissState = DISMISS_DOWN
168         val closeToDown: AnimatedVectorDrawable =
169                 ContextCompat.getDrawable(this, R.drawable.avd_close_to_down) as AnimatedVectorDrawable
170         close.setImageDrawable(closeToDown)
171         closeToDown.start()
172     }
173 ____________________________________________________________________________
174     fun dismiss(view: View) {
175         if (view.visibility != View.VISIBLE) return
176         bottomSheet.dismiss()
177     }
178
179     private inner class FollowerAdapter<T : FollowerListable>(context: Context) : RecyclerView.Adapter<
    RecyclerView.ViewHolder>(), DataLoadingSubject.DataLoadingCallbacks {
180         private var loading = true
181         private val layoutInflater: LayoutInflater = LayoutInflater.from(context)
182         internal var items: MutableList<T> = ArrayList(0)
183 ____________________________________________________________________________
184         internal val dataItemCount: Int
185             get() = items.size
186 ____________________________________________________________________________
187         private val loadingMoreItemPosition: Int
188             get() = if (loading) itemCount - 1 else RecyclerView.NO_POSITION
189 ____________________________________________________________________________
190         init {
191             setHasStableIds(true)
192         }
193 ____________________________________________________________________________
194         override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
195             return when (viewType) {
196                 TYPE_PLAYER -> createPlayerViewHolder(parent)
197                 else -> LoadingViewHolder(
198                         // TYPE_LOADING
199                         layoutInflater.inflate(R.layout.list_loading, parent, false))
```

```
200         }
201     }
202 
203     private fun createPlayerViewHolder(parent: ViewGroup): SheetViewHolder {
204         return SheetViewHolder(
205                 layoutInflater.inflate(R.layout.sheet_item, parent, false))
206     }
207 
208     override fun onBindViewHolder(holder: ViewHolder, position: Int) {
209         if (position == loadingMoreItemPosition) return
210         bindPlayer(holder as SheetViewHolder, items[position])
211     }
212 
213     private fun bindPlayer(holder: SheetViewHolder, t: T) {
214         GlideApp.with(holder.itemView.context)
215                 .load(t.customer?.photo)
216                 .apply(RequestOptions.circleCropTransform())
217                 .apply(RequestOptions().placeholder(R.drawable.avatar_placeholder))
218                 .apply(RequestOptions().override(largeAvatarSize, largeAvatarSize))
219                 .transition(withCrossFade())
220                 .into(holder.playerAvatar)
221         holder.playerName.text = t.customer?.name?.toLowerCase(Locale.US)
222         if (!TextUtils.isEmpty(t.customer?.info)) {
223             HtmlUtils.parseAndSetText(holder.playerBio, t.customer?.info)
224         }
225         if (t.dateCreated != null) {
226             holder.timeAgo.text = DateUtils.getRelativeTimeSpanString(t.dateCreated?.time!!,
227                     System.currentTimeMillis(), DateUtils.SECOND_IN_MILLIS)
228                     .toString().toLowerCase(Locale.US)
229         }
230     }
231 
232     override fun getItemViewType(position: Int): Int {
233         return if (position < dataItemCount && dataItemCount > 0) {
234             TYPE_PLAYER
235         } else TYPE_LOADING
236     }
237 
238     override fun getItemId(position: Int): Long {
239         return if (getItemViewType(position) == TYPE_LOADING) {
240             -1L
241         } else items[position].id!!
242     }
243 
244     override fun getItemCount(): Int {
245         return dataItemCount + if (loading) 1 else 0
246     }
247 
248     override fun dataStartedLoading() {
249         if (loading) return
250         loading = true
251         notifyItemInserted(loadingMoreItemPosition)
252     }
253 
254     override fun dataFinishedLoading() {
255         if (!loading) return
256         val loadingPos = loadingMoreItemPosition
257         loading = false
258         notifyItemRemoved(loadingPos)
259     }
260 
261     internal fun addItems(newItems: List<T>) {
262         val insertRangeStart = dataItemCount
263         items.addAll(newItems)
264         notifyItemRangeInserted(insertRangeStart, newItems.size)
265     }
266 
```

```
267     }
268
269     internal class SheetViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
270         var playerAvatar: CircularImageView = itemView.findViewById(R.id.player_avatar)
271         var playerName: TextView = itemView.findViewById(R.id.player_name)
272         var playerBio: TextView = itemView.findViewById(R.id.player_bio)
273         var timeAgo: TextView = itemView.findViewById(R.id.time_ago)
274     }
275
276     internal class LoadingViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
277
278         var progress: ProgressBar = itemView as ProgressBar
279
280     }
281
282
283 }
284
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.app.Activity
8  import android.content.Intent
9  import android.os.Bundle
10 import android.support.v4.content.ContextCompat
11 import android.transition.TransitionManager
12 import android.view.View
13 import android.view.ViewGroup
14 import android.widget.Button
15 import android.widget.FrameLayout
16 import android.widget.TextView
17 import android.widget.Toast
18 import com.afollestad.materialdialogs.MaterialDialog
19 import com.google.firebase.auth.FirebaseAuth
20 import com.google.firebase.auth.FirebaseUser
21 import com.google.firebase.iid.FirebaseInstanceId
22 import io.pergasus.R
23 import io.pergasus.api.PhoenixClient
24 import io.pergasus.api.PhoenixUtils
25 import io.pergasus.data.Customer
26 import io.pergasus.ui.transitions.FabTransform
27 import io.pergasus.ui.transitions.MorphTransform
28 import io.pergasus.ui.widget.ConfirmationToastView
29 import io.pergasus.util.bindView
30
31
32 /** Login for basic users */
33 class AuthActivity : Activity() {
34
35     private val frame: FrameLayout  by bindView(R.id.frame_login)
36     private val container: ViewGroup  by bindView(R.id.container)
37     private val login: Button  by bindView(R.id.login)
38     private val message: TextView  by bindView(R.id.login_message)
39     private val loginFailed: TextView  by bindView(R.id.login_failed_message)
40
41     private var isDismissing = false
42     private var isLoginFailed = false
43
44     private lateinit var client: PhoenixClient
45     private lateinit var auth: FirebaseAuth
46     private lateinit var loading: MaterialDialog
47 ____________________________________________________________________________
48     override fun onCreate(savedInstanceState: Bundle?) {
49         super.onCreate(savedInstanceState)
50         setContentView(R.layout.activity_auth)
51
52         if (!FabTransform.setup(this, container)) {
53             MorphTransform.setup(this, container,
54                     ContextCompat.getColor(this, R.color.background_light),
55                     resources.getDimensionPixelSize(R.dimen.dialog_corners))
56         }
57
58         //Shared preferences
59         client = PhoenixClient(this@AuthActivity)
60         auth = client.auth
61
62         loading = client.getDialog()
63
64         //Action for frame layout & login button
65         frame.setOnClickListener(dismiss)
66         login.setOnClickListener(doLogin)
```

```
67      container.setOnClickListener(null)
68
69  }
70
71  //Do login action
72  private val doLogin: View.OnClickListener = View.OnClickListener {
73      if (client.isConnected) {
74          //Launch AuthUI for Firebase
75          showLoading()
76          PhoenixUtils.firebaseAuthUI(this@AuthActivity, RC_AUTH_FIREBASE)
77      } else {
78          showLoginFailed(getString(R.string.no_internet_body))
79      }
80  }
81
82  override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
83      when (requestCode) {
84          RC_AUTH_FIREBASE -> when (resultCode) {
85              RESULT_OK -> {
86                  val firebaseUser = auth.currentUser
87                  updateUI(firebaseUser)
88              }
89              RESULT_CANCELED, RESULT_FIRST_USER -> showLoginFailed("Cancelled operation")
90          }
91      }
92  }
93
94  private fun updateUI(currentUser: FirebaseUser?) {
95      if (currentUser != null) {
96          client.db.collection("phoenix/mobile/${PhoenixUtils.CUSTOMER_REF}")
97                  .get()
98                  .addOnSuccessListener { querySnapshot ->
99                      if (querySnapshot.isEmpty) {
100                         //If no data exists in the db, create a new user
101                         createNewUser(currentUser)
102                     } else {
103                         var hasUser = false
104                         var customer = Customer()
105                         //Start a O(n2) query for all documents in the db
106                         for (item in querySnapshot.documents) {
107                             //If document snapshot's key is similar to the account id then
108                             // log user in else create a new instance of the user
109                             if (item.exists() && item.id == currentUser.uid) {
110                                 customer = item.toObject(Customer::class.java)
111                                 hasUser = true
112                             }
113                         }
114
115                         if (hasUser) {
116                             client.setCustomer(customer.key!!)
117                             if (client.isLoggedIn) {
118                                 client.setLoggedInUser(customer)
119                                 isLoginFailed = false
120                                 val toast = ConfirmationToastView(applicationContext,
121                                         client.customer.name,
122                                         client.customer.photo,
123                                         getString(R.string.app_logged_in_as))
124                                 toast.create()
125                                 setResult(Activity.RESULT_OK)
126                                 finishAfterTransition()
127                             }
128                         } else createNewUser(currentUser)
129                     }
130                 }
131     }
132 }
133
```

```
134     private fun createNewUser(user: FirebaseUser) {
135         //Build new customer instance
136         val customer = Customer.Builder()
137                 .setKey(user.uid)
138                 .setInfo(user.email!!)
139                 .setName(if (user.displayName.isNullOrEmpty()) "No Username" else user.displayName!!)
140                 .setPhoto(if (user.photoUrl != null) user.photoUrl?.toString()!! else "default")
141                 .setId(System.currentTimeMillis())
142                 .setAddressLat("")
143                 .setAddressLng("")
144                 .setTokenId(FirebaseInstanceId.getInstance().token!!)
145                 .build()
146
147         //Add data to database
148         client.db.collection("phoenix/mobile/${PhoenixUtils.CUSTOMER_REF}")
149                 .document(customer.key!!)
150                 .set(customer.toHashMap(customer))
151                 .addOnFailureListener { e ->
152                     showLoginFailed(e.localizedMessage)
153                 }
154                 .addOnCompleteListener { task ->
155                     if (task.isSuccessful) {
156                         client.setCustomer(customer.key!!)
157                         if (client.isLoggedIn) {
158                             client.setLoggedInUser(customer)
159                             isLoginFailed = false
160                             val toast = ConfirmationToastView(applicationContext,
161                                     client.customer.name,
162                                     client.customer.photo,
163                                     getString(R.string.app_logged_in_as))
164                             toast.create()
165                             setResult(Activity.RESULT_OK)
166                             finishAfterTransition()
167
168                         }
169                     } else {
170                         showLoginFailed(task.exception?.message)
171                     }
172                 }
173     }
174 ---------------------------------------------------------------------------
175     override fun onSaveInstanceState(savedInstanceState: Bundle) {
176         super.onSaveInstanceState(savedInstanceState)
177         savedInstanceState.putBoolean(STATE_LOGIN_FAILED, isLoginFailed)
178     }
179 ---------------------------------------------------------------------------
180     //Dismiss activity
181     private val dismiss: View.OnClickListener = View.OnClickListener {
182         isDismissing = true
183         setResult(Activity.RESULT_CANCELED)
184         finishAfterTransition()
185     }
186 ---------------------------------------------------------------------------
187     private fun showLoginFailed(info: String?) {
188         isLoginFailed = true
189         showLogin()
190         loginFailed.visibility = View.VISIBLE
191         Toast.makeText(this, info, Toast.LENGTH_SHORT).show()
192     }
193 ---------------------------------------------------------------------------
194     private fun showLogin() {
195         TransitionManager.beginDelayedTransition(container)
196         message.visibility = View.VISIBLE
197         login.visibility = View.VISIBLE
198         loading.hide()
199     }
200 ---------------------------------------------------------------------------
```

```
201     private fun showLoading() {
202         TransitionManager.beginDelayedTransition(container)
203         message.visibility = View.GONE
204         login.visibility = View.GONE
205         loginFailed.visibility = View.GONE
206         loading.show()
207     }
208 
209     /** Returns the activity to its previous state */
210     override fun onBackPressed() {
211         isDismissing = true
212         setResult(Activity.RESULT_CANCELED)
213         finishAfterTransition()
214     }
215 
216 
217     companion object {
218         private const val STATE_LOGIN_FAILED: String = "STATE_LOGIN_FAILED"
219         private const val RC_AUTH_FIREBASE: Int = 34234
220     }
221 
222 }
223 
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.animation.Animator
8  import android.animation.AnimatorListenerAdapter
9  import android.annotation.SuppressLint
10 import android.app.Activity
11 import android.app.ActivityOptions
12 import android.content.Intent
13 import android.graphics.drawable.AnimatedVectorDrawable
14 import android.os.Bundle
15 import android.support.v4.content.ContextCompat
16 import android.support.v7.widget.LinearLayoutManager
17 import android.support.v7.widget.RecyclerView
18 import android.text.format.DateUtils
19 import android.transition.TransitionManager
20 import android.view.*
21 import android.widget.*
22 import com.afollestad.materialdialogs.MaterialDialog
23 import com.bumptech.glide.load.engine.DiskCacheStrategy
24 import com.bumptech.glide.load.resource.drawable.DrawableTransitionOptions.withCrossFade
25 import com.bumptech.glide.request.target.Target
26 import com.google.firebase.firestore.EventListener
27 import com.google.firebase.firestore.QuerySnapshot
28 import io.pergasus.BuildConfig
29 import io.pergasus.R
30 import io.pergasus.R.string.price
31 import io.pergasus.api.PhoenixClient
32 import io.pergasus.api.PhoenixUtils
33 import io.pergasus.data.Order
34 import io.pergasus.data.PhoenixNotification
35 import io.pergasus.data.Purchase
36 import io.pergasus.ui.recyclerview.Divided
37 import io.pergasus.ui.recyclerview.GridItemDividerDecoration
38 import io.pergasus.ui.recyclerview.SlideInItemAnimator
39 import io.pergasus.ui.transitions.FabTransform
40 import io.pergasus.ui.transitions.MorphTransform
41 import io.pergasus.ui.widget.BadgedFourThreeImageView
42 import io.pergasus.util.AnimUtils
43 import io.pergasus.util.ShortcutHelper
44 import io.pergasus.util.bindView
45 import io.pergasus.util.glide.GlideApp
46 import timber.log.Timber
47 import java.text.NumberFormat
48 import java.util.*
49
50
51 /** User's Cart Activity */
52 @SuppressLint("LogConditional")
53 class CartActivity : Activity() {
54     //Widgets
55     private val frame: FrameLayout by bindView(R.id.draggable_frame)
56     private val frameContent: ViewGroup by bindView(R.id.bottom_sheet_content)
57     private val grid: RecyclerView by bindView(R.id.grid)
58     private val toolbar: Toolbar by bindView(R.id.toolbar)
59     private val fab: ImageButton by bindView(R.id.fab)
60     private val noCart: TextView by bindView(R.id.no_cart)
61     private val itemsCost: TextView by bindView(R.id.total_cost)
62     var fabPosting: ImageButton? = null
63
64     //Others
65     private lateinit var client: PhoenixClient  //Shared preferences
66     private lateinit var adapter: CartItemAdapter   //Adapter for recyclerview
```

```
67    private lateinit var layoutManager: LinearLayoutManager //LayoutManager for recyclerview
68    private val orders: ArrayList<Order> = ArrayList(0) //Empty arrayList for Orders
69
70    override fun onCreate(savedInstanceState: Bundle?) {
71        super.onCreate(savedInstanceState)
72        setContentView(R.layout.activity_cart)
73        //Setup Toolbar support
74        setActionBar(toolbar)
75
76        //Animate the toolbar if the user ios launching the activity for the first time
77        if (savedInstanceState == null) animateToolbar()
78
79        //close the activity when user clicks on the back button
80        toolbar.setNavigationOnClickListener { setResultAndFinish() }
81
82        //Allow enter transition from FAB transformation
83        if (!FabTransform.setup(this, frameContent)) {
84            MorphTransform.setup(this, frameContent,
85                    ContextCompat.getColor(this, R.color.background_dark), 0)
86        }
87
88        //Setup SharedPreferences
89        client = PhoenixClient(this@CartActivity)
90
91        //Setup recyclerview
92        adapter = CartItemAdapter(this@CartActivity)    //init adapter
93        grid.adapter = adapter  //set adapter for recyclerview
94        layoutManager = LinearLayoutManager(this)   //init layout manager
95        grid.layoutManager = layoutManager  //set layout manager for recyclerview
96        grid.itemAnimator = SlideInItemAnimator()   //set item animator (when items are entering in)
97
98        //Add separator for items in the recyclerview
99        grid.addItemDecoration(GridItemDividerDecoration(this, R.dimen.divider_height, R.color.divider))
100
101       //Enable shortcut to cart in Android 8.0+
102       ShortcutHelper.reportCartUsed(this)
103   }
104
105   //Function to return results to the calling activity, if any
106   private fun setResultAndFinish() {
107       setResult(RESULT_OK)
108       finishAfterTransition()
109   }
110
111   //Function to get data from the Orders database reference
112   private fun getData() {
113       //ref: phoenix/orders/{userKey}/**
114       if (client.isLoggedIn) {
115           if (orders.isNotEmpty()) orders.clear()
116           client.db.document(PhoenixUtils.ORDER_REF)
117                   .collection(client.customer.key!!)
118                   .addSnapshotListener(this@CartActivity, EventListener<QuerySnapshot?> { p0, p1 ->
119                       if (p1 != null) {
120                           Toast.makeText(this@CartActivity, p1.localizedMessage,
121                                   Toast.LENGTH_LONG).show()
122                           noCart.visibility = View.VISIBLE
123                           Timber.d(p1.localizedMessage)
124                           return@EventListener
125                       }
126                       if (p0 != null) {
127                           p0.documents
128                                   .filter { it.exists() }
129                                   .mapTo(orders) { it.toObject(Order::class.java) }
130                           adapter.addOrders(orders)
131                           //Hide no cart label if there are items in the database
132                           noCart.visibility = if (p0.isEmpty) View.VISIBLE else View.GONE
133
```

```
134                     val total: Double = getTotalValue(orders)
135                     val quantity: Double = getTotalQuantity(orders)
136                     //Set total cost
137                     itemsCost.visibility = View.VISIBLE
138                     itemsCost.text = String.format(Locale.US, "Total Cost: %s",
139                         NumberFormat.getCurrencyInstance(Locale.US).format(total))
140                     checkEmptyState()
141
142                     //Add click action
143                     fab.setOnClickListener {
144                         if (adapter.itemCount > 0) {
145                             val order = Intent(this, OrderActivity::class.java)
146                             FabTransform.addExtras(order,
147                                 ContextCompat.getColor(this, R.color.accent),
148                                 R.drawable.ic_attach_money_black_24dp)
149                             order.putExtra(OrderActivity.EXTRA_CART_PRICE, total)
150                             order.putExtra(OrderActivity.EXTRA_CART_QUANTITY, quantity)
151                             order.putExtra(OrderActivity.EXTRA_CART_TITLE,
152                                 String.format("For the purchase of ${adapter.itemCount} items"))
153                             val options = ActivityOptions.makeSceneTransitionAnimation(this,
154                                 fab, getString(R.string.transition_new_designer_news_post))
155                             startActivityForResult(order, CODE_ORDER, options.toBundle())
156                         }
157                     }
158                 }
159             })
160
161         } else {
162             val intent = Intent(this@CartActivity, AuthActivity::class.java)
163             FabTransform.addExtras(intent,
164                 ContextCompat.getColor(this, R.color.button_accent),
165                 R.drawable.ic_attach_money_black_24dp)
166             val options = ActivityOptions.makeSceneTransitionAnimation(this, fab,
167                 getString(R.string.transition_dribbble_login))
168             startActivity(intent, options.toBundle())
169         }
170
171     }
172 ---------------------------------------------------------------------------
173     //Called once the activity starts up after onCreate has been called
174     override fun onStart() {
175         super.onStart()
176         orders.clear()
177         getData()
178     }
179 ---------------------------------------------------------------------------
180     //Called once the activity has stopped
181     override fun onStop() {
182         orders.clear()
183         super.onStop()
184     }
185 ---------------------------------------------------------------------------
186     //Check the number of items in the adapter and update the UI respectively
187     private fun checkEmptyState() {
188         if (adapter.itemCount > 0) {
189             TransitionManager.beginDelayedTransition(frame)
190             grid.visibility = View.VISIBLE
191             noCart.visibility = View.GONE
192             fab.visibility = View.VISIBLE
193         } else {
194             TransitionManager.beginDelayedTransition(frame)
195             grid.visibility = View.GONE
196             noCart.visibility = View.VISIBLE
197             fab.visibility = View.GONE
198         }
199     }
200 ---------------------------------------------------------------------------
```

```
201     //Calculates the total sum of all items' price in the user's shopping cart
202     private fun getTotalValue(orders: ArrayList<Order>): Double {
203         var total = 0.00
204         if (orders.size > 0) {
205             (0 until orders.size)
206                     .map { orders[it] }
207                     //Multiply the price by the quantity to get the total value
208                     .forEach { total += (it.price?.toDouble()!!) * (it.quantity?.toInt()!!) }
209         }
210         return total
211
212     }
213 ------------------------------------------------------------------------------------------
214     //Calculates the total sum of all items' quantity in the user's shopping cart
215     private fun getTotalQuantity(orders: ArrayList<Order>): Double {
216         var total = 0.00
217         if (orders.size > 0) {
218             (0 until orders.size)
219                     .map { orders[it] }
220                     //Multiply the price by the quantity to get the total value
221                     .forEach { total += it.quantity?.toDouble()!! }
222         }
223         return total
224
225     }
226 ------------------------------------------------------------------------------------------
227     //Handle results from intent actions
228     override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
229         when (requestCode) {
230             CODE_ORDER -> {
231                 when (resultCode) {
232                     OrderActivity.RESULT_DRAG_DISMISSED -> {
233                         showFab()
234                     }
235                     OrderActivity.RESULT_PAYING -> {
236                         if (data != null) {
237                             showPostingProgress()
238                             //Build new Purchase Object
239                             val builder = Purchase.Builder()
240                                     .setId(System.currentTimeMillis())
241                                     .setKey(client.customer.key!!)
242                                     .setPrice(price.toString())
243
244                             //Add each item in orders list
245                             for (item in orders) {
246                                 builder.setOrderItems(item)
247                             }
248                             client.db.document(PhoenixUtils.PURCHASE_REF)
249                                     .collection(client.customer.key!!)
250                                     .document()
251                                     .set(builder.build())
252                                     .addOnCompleteListener { task ->
253                                         if (task.isSuccessful) {
254                                             //adapter.clear()
255                                             doSuccessAnimation()   //Do animation
256                                         } else {
257                                             doFailureAnimation()   //Do animation
258                                         }
259                                     }
260                         }
261                     }
262                     else -> {
263                         super.onActivityResult(requestCode, resultCode, data)
264                     }
265                 }
266             }
267         }
```

```
268     }
269
270     private fun doSuccessAnimation() {
271         if (client.isConnected) clearCart()
272         // success animation
273         val complete: AnimatedVectorDrawable? =
274                 getDrawable(R.drawable.avd_upload_complete) as AnimatedVectorDrawable
275         if (complete != null) {
276             fabPosting?.setImageDrawable(complete)
277             complete.start()
278             // length of R.drawable.avd_upload_complete
279             fabPosting?.postDelayed({
280                 fabPosting?.visibility = View.GONE
281                 sendNotification()  //Send notification to user
282                 //Show success screen
283                 val intent = Intent(this@CartActivity, SuccessActivity::class.java)
284                 intent.putExtra(SuccessActivity.EXTRA_PAGE, SuccessActivity.PAGE_SUCCESS)
285                 startActivity(intent)
286                 setResultAndFinish()
287             }, 2000)
288         } else {
289             sendNotification()  //Send notification to user
290             //Show success screen
291             val intent = Intent(this@CartActivity, SuccessActivity::class.java)
292             intent.putExtra(SuccessActivity.EXTRA_PAGE, SuccessActivity.PAGE_SUCCESS)
293             startActivity(intent)
294             setResultAndFinish()
295         }
296     }
297
298     private fun sendNotification() {
299         //Send notification to database
300         val notification = PhoenixNotification(
301                 "Congratulations",
302                 "Your recent purchase was successful",
303                 "dummy",    //Dummy text
304                 client.customer.key!!,
305                 "dummy" //Dummy text
306         )
307         client.db.collection(PhoenixUtils.NOTIFICATIONS_REF)
308                 .document()
309                 .set(notification.toHashMap(notification))
310                 .addOnFailureListener { exception ->
311                     if (BuildConfig.DEBUG) Timber.d(exception.localizedMessage)
312                 }.addOnCompleteListener { task ->
313                     if (task.isSuccessful) {
314                         if (BuildConfig.DEBUG) Timber.d("Notification sent")
315                         //Firebase functions will notify the user of new notification
316                     } else {
317                         if (BuildConfig.DEBUG) Timber.d(task.exception?.localizedMessage)
318                     }
319                 }
320     }
321
322     private fun clearCart() {
323         client.db.document(PhoenixUtils.ORDER_REF)
324                 .collection(client.customer.key!!)
325                 .get()
326                 .addOnSuccessListener { querySnapshot ->
327                     if (!querySnapshot.isEmpty) {
328                         querySnapshot.documents
329                                 .filter { it.exists() }
330                                 .forEach { it.reference.delete() }
331                         adapter.notifyDataSetChanged()
332                         checkEmptyState()
333                     }
334                 }
```

```
335     }
336
337     //Show failure animation
338     private fun doFailureAnimation() {
339         // failure animation
340         val failed: AnimatedVectorDrawable? =
341                 getDrawable(R.drawable.avd_upload_error) as AnimatedVectorDrawable
342         if (failed != null) {
343             fabPosting?.setImageDrawable(failed)
344             failed.start()
345         }
346         // remove the upload progress 'fab' and reshow the regular one
347         fabPosting?.animate()
348                 ?.alpha(0f)
349                 ?.rotation(90f)
350                 ?.setStartDelay(2000L) // leave error on screen briefly
351                 ?.setDuration(300L)
352                 ?.setInterpolator(AnimUtils
353                         .getFastOutSlowInInterpolator(this@CartActivity))
354                 ?.setListener(object : AnimatorListenerAdapter() {
355                     override fun onAnimationEnd(animation: Animator?) {
356                         fabPosting?.visibility = View.GONE
357                         fabPosting?.alpha = 1f
358                         fabPosting?.rotation = 0f
359                     }
360                 })
361     }
362
363     //Show posting progress once data has been
364     private fun showPostingProgress() {
365         ensurePostingProgressInflated()
366         fabPosting?.visibility = View.VISIBLE
367         // if stub has just been inflated then it will not have been laid out yet
368         if (fabPosting?.isLaidOut!!) {
369             revealPostingProgress()
370         } else {
371             fabPosting?.addOnLayoutChangeListener(object : View.OnLayoutChangeListener {
372                 override fun onLayoutChange(p0: View?, p1: Int, p2: Int, p3: Int, p4: Int, p5: Int, p6: Int, p7: Int, p8:
    Int) {
373                     fabPosting?.removeOnLayoutChangeListener(this)
374                     revealPostingProgress()
375                 }
376             })
377         }
378     }
379
380     private fun ensurePostingProgressInflated() {
381         if (fabPosting != null) return
382         fabPosting = (findViewById<ViewStub>(R.id.stub_posting_progress)).inflate() as ImageButton?
383     }
384
385     private fun revealPostingProgress() {
386         val reveal = ViewAnimationUtils.createCircularReveal(fabPosting,
387                 fabPosting?.pivotX!!.toInt(),
388                 fabPosting?.pivotY!!.toInt(),
389                 0f,
390                 (fabPosting?.width!! / 2).toFloat())
391                 .setDuration(600L)
392         reveal.interpolator = AnimUtils.getFastOutLinearInInterpolator(this)
393         reveal.start()
394         val uploading: AnimatedVectorDrawable? = getDrawable(R.drawable.avd_uploading) as
    AnimatedVectorDrawable
395         if (uploading != null) {
396             fabPosting?.setImageDrawable(uploading)
397             uploading.start()
398             updateData()
399         }
```

```
400     }
401
402     private fun updateData() {
403         if (client.isLoggedIn) {
404             if (client.isConnected) {
405                 client.db.document(PhoenixUtils.ORDER_REF)
406                         .collection(client.customer.key!!)
407                         .get()
408                         .addOnCompleteListener { task ->
409                             if (task.isSuccessful) {
410                                 showSuccessProgress()
411                             } else {
412                                 Toast.makeText(this, task.exception?.localizedMessage,
413                                         Toast.LENGTH_LONG).show()
414                                 hidePostingProgress()
415                             }
416                         }
417             } else {
418                 Toast.makeText(this, "Cannot retrieve data at this time", Toast.LENGTH_LONG).show()
419                 hidePostingProgress()
420             }
421         }
422     }
423
424     private fun hidePostingProgress() {
425         // success animation
426         val failed = getDrawable(R.drawable.avd_upload_error) as AnimatedVectorDrawable?
427         if (failed != null) {
428             fabPosting?.setImageDrawable(failed)
429             failed.start()
430         }
431         fabPosting?.animate()
432                 ?.alpha(0f)
433                 ?.rotation(90f)
434                 ?.setStartDelay(2000L) // leave error on screen briefly
435                 ?.setDuration(300L)
436                 ?.setListener(object : AnimatorListenerAdapter() {
437                     override fun onAnimationEnd(animation: Animator?) {
438                         fabPosting?.visibility = View.GONE
439                         fabPosting?.alpha = 1f
440                         fabPosting?.rotation = 0f
441                     }
442                 })
443                 ?.interpolator = AnimUtils.getFastOutSlowInInterpolator(this@CartActivity)
444
445     }
446
447     private fun showSuccessProgress() {
448         // success animation
449         val complete = getDrawable(R.drawable.avd_upload_complete) as AnimatedVectorDrawable?
450         if (complete != null) {
451             fabPosting?.setImageDrawable(complete)
452             complete.start()
453             fabPosting?.postDelayed({ fabPosting?.visibility = View.GONE }, 2100)
454         }
455     }
456
457     private fun showFab() {
458         fab.alpha = 0f
459         fab.scaleX = 0f
460         fab.scaleY = 0f
461         fab.translationY = (fab.height / 2).toFloat()
462         fab.animate()
463                 .alpha(1f)
464                 .scaleX(1f)
465                 .scaleY(1f)
466                 .translationY(0f)
```

```
            .setDuration(300L)
            .setInterpolator(AnimUtils.getLinearOutSlowInInterpolator(this@CartActivity))
            .start()
    }

    //Animate the title of the toolbar
    private fun animateToolbar() {
        // this is gross but toolbar doesn't expose it's children to animate them :(
        val t = toolbar.getChildAt(0)
        if (t != null && t is TextView) {

            // fade in and space out the title.  Animating the letterSpacing performs horribly so
            // fake it by setting the desired letterSpacing then animating the scaleX ¯\_(ツ)_/¯
            t.alpha = 0f
            t.scaleX = 0.8f

            t.animate()
                    .alpha(1f)
                    .scaleX(1f)
                    .setStartDelay(300)
                    .setDuration(600).interpolator = AnimUtils.getFastOutSlowInInterpolator(this@CartActivity)
        }
    }

    //Adapter for customer's orders
    internal inner class CartItemAdapter(private val host: Activity) : RecyclerView.Adapter<CartViewHolder
>() {
        private val items: ArrayList<Order> = ArrayList(0)
        private val loading: MaterialDialog

        init {
            setHasStableIds(true)
            loading = client.getDialog()
        }

        override fun onBindViewHolder(holder: CartViewHolder, p0: Int) {
            val position = holder.adapterPosition
            val order = items[position]
            holder.name.text = order.name
            holder.price.text = NumberFormat.getCurrencyInstance(Locale.US)
                    .format(order.price?.toDouble())
            holder.quantity.text = String.format("%s units added", order.quantity)
            GlideApp.with(host)
                    .load(order.image)
                    .placeholder(R.color.content_placeholder)
                    .error(R.color.content_placeholder)
                    .diskCacheStrategy(DiskCacheStrategy.AUTOMATIC)
                    .override(Target.SIZE_ORIGINAL, Target.SIZE_ORIGINAL)
                    .transition(withCrossFade())
                    .into(holder.image)

            holder.itemView.setOnClickListener({ _ ->
                val builder = MaterialDialog.Builder(this@CartActivity)
                val v = layoutInflater.inflate(R.layout.order_item_info, null, false)
                builder.customView(v, false)  //Attach view to builder
                val dialog = builder.build()
                //Get items in layout
                val img = v.findViewById<BadgedFourThreeImageView>(R.id.order_info_image)
                val name = v.findViewById<TextView>(R.id.order_info_name)
                val details = v.findViewById<TextView>(R.id.order_info_details)
                val del = v.findViewById<Button>(R.id.order_info_delete)
                //Load image into view
                GlideApp.with(host)
                        .load(order.image)
                        .placeholder(R.color.content_placeholder)
                        .error(R.color.content_placeholder)
                        .diskCacheStrategy(DiskCacheStrategy.AUTOMATIC)
```

```
533                 .override(Target.SIZE_ORIGINAL, Target.SIZE_ORIGINAL)
534                 .transition(withCrossFade())
535                 .into(img)
536             name.text = order.name  //Order name
537             val dateInfo = DateUtils.getRelativeDateTimeString(this@CartActivity,
538                 if (order.timestamp == null) System.currentTimeMillis() else order.timestamp?.time!!,
539                 DateUtils.SECOND_IN_MILLIS, DateUtils.SECOND_IN_MILLIS, DateUtils
540                 .FORMAT_ABBREV_ALL)
541             val info = String.format("Purchased %s goods for ${NumberFormat
542                 .getCurrencyInstance().format(order.price?.toDouble())} each\n on %s",
543                 order.quantity, dateInfo)
544             details.text = info
545             //Delete action
546             del.setOnClickListener({
547                dialog.dismiss()
548                removeOrder(order)
549             })
550             dialog.show()
551          })
552       }
553 ----------------------------------------------------------------------------------------------------
554       override fun getItemCount(): Int {
555          return items.size
556       }
557 ----------------------------------------------------------------------------------------------------
558       override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CartViewHolder {
559          val v: View = LayoutInflater.from(host).inflate(R.layout.order_item, parent, false)
560          return CartViewHolder(v)
561       }
562 ----------------------------------------------------------------------------------------------------
563       override fun getItemId(position: Int): Long {
564          return items[position].id
565       }
566 ----------------------------------------------------------------------------------------------------
567       fun addOrders(newItems: ArrayList<Order>) {
568          deduplicateAndAdd(newItems)
569          notifyDataSetChanged()
570       }
571 ----------------------------------------------------------------------------------------------------
572       private fun removeOrder(order: Order) {
573          if (client.isConnected) {
574             loading.show()
575             dispatchOrderRemoved(order)
576          } else {
577             Toast.makeText(this@CartActivity, "Item cannot be removed at this time",
578                 Toast.LENGTH_SHORT).show()
579          }
580       }
581 ----------------------------------------------------------------------------------------------------
582       fun clear() {
583          items.clear()
584          notifyItemRangeChanged(0, items.size)
585       }
586 ----------------------------------------------------------------------------------------------------
587       //Remove from database
588       private fun dispatchOrderRemoved(order: Order) {
589          client.db.document(PhoenixUtils.ORDER_REF)
590                .collection(client.customer.key!!)
591                .whereEqualTo("name", order.name)
592                .get()
593                .addOnCompleteListener { task ->
594                   if (task.isSuccessful) {
595                      val documents = task.result.documents
596                      if (documents.isNotEmpty() && documents[0].exists()) {
597                         documents[0].reference.delete().addOnSuccessListener { _ ->
598                            loading.dismiss()
599                            val position = items.indexOf(order)
```

```
                            items.removeAt(position)
                            notifyItemRemoved(position)
                            getData()
                            if (BuildConfig.DEBUG) {
                                Timber.d("Item ${order.name} removed from database")
                            }
                        }
                    }
                } else {
                    loading.dismiss()
                    Toast.makeText(applicationContext, task.exception?.localizedMessage,
                            Toast.LENGTH_SHORT).show()
                }
            }.addOnFailureListener { exception ->
                loading.dismiss()
                Toast.makeText(applicationContext, exception.localizedMessage,
                        Toast.LENGTH_SHORT).show()
            }
        }


    ---------------------------------------------------------------------------------------------------------------
        @SuppressLint("LogConditional")
        private fun deduplicateAndAdd(newItems: ArrayList<Order>) {
            Timber.d(newItems.toString())
            val count = itemCount
            for (newItem in newItems) {
                val add = !(0 until count)
                        .map { getItem(it) }
                        .contains(newItem)
                if (add) {
                    add(newItem)
                }
            }
        }
    ---------------------------------------------------------------------------------------------------------------
        private fun add(newItem: Order) {
            items.add(newItem)
        }
    ---------------------------------------------------------------------------------------------------------------
        private fun getItem(position: Int): Order {
            return items[position]
        }

    }

    internal inner class CartViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView), Divided {
        var name: TextView = itemView.findViewById(R.id.order_name)
        var image: BadgedFourThreeImageView = itemView.findViewById(R.id.order_image)
        var price: TextView = itemView.findViewById(R.id.order_price)
        var quantity: TextView = itemView.findViewById(R.id.order_quantity)
    }

    companion object {
        private const val CODE_ORDER = 13
        const val RESULT_PRICE = "RESULT_PRICE"
    }
}
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.annotation.SuppressLint
8  import android.app.Activity
9  import android.app.ActivityManager
10 import android.app.ActivityOptions
11 import android.app.AlertDialog
12 import android.content.Context
13 import android.content.Intent
14 import android.graphics.BitmapFactory
15 import android.graphics.drawable.AnimatedVectorDrawable
16 import android.net.*
17 import android.os.Bundle
18 import android.support.v4.content.ContextCompat
19 import android.support.v4.view.GravityCompat
20 import android.support.v4.widget.DrawerLayout
21 import android.support.v7.widget.GridLayoutManager
22 import android.support.v7.widget.RecyclerView
23 import android.support.v7.widget.helper.ItemTouchHelper
24 import android.text.*
25 import android.text.Annotation
26 import android.text.style.ForegroundColorSpan
27 import android.text.style.ImageSpan
28 import android.transition.TransitionManager
29 import android.view.*
30 import android.widget.*
31 import com.bumptech.glide.integration.recyclerview.RecyclerViewPreloader
32 import com.bumptech.glide.util.ViewPreloadSizeProvider
33 import com.google.firebase.auth.FirebaseAuth
34 import io.pergasus.R
35 import io.pergasus.api.*
36 import io.pergasus.data.Product
37 import io.pergasus.ui.FilterAdapter.FilterAuthoriser
38 import io.pergasus.ui.recyclerview.FilterTouchHelperCallback
39 import io.pergasus.ui.recyclerview.GridItemDividerDecoration
40 import io.pergasus.ui.recyclerview.SlideInItemAnimator
41 import io.pergasus.ui.transitions.FabTransform
42 import io.pergasus.ui.transitions.MorphTransform
43 import io.pergasus.util.AnimUtils
44 import io.pergasus.util.ViewUtils
45 import io.pergasus.util.bindView
46 import timber.log.Timber
47 import java.security.InvalidParameterException
48 import java.util.*
49
50 /** Home Screen for tha application */
51 class HomeActivity : Activity() {
52
53     private val grid: RecyclerView by bindView(R.id.grid)
54     private val drawer: DrawerLayout by bindView(R.id.drawer)
55     private val toolbar: Toolbar by bindView(R.id.toolbar)
56     private val fab: ImageButton by bindView(R.id.fab)
57     private val loading: ProgressBar by bindView(android.R.id.empty)
58     private val filtersList: RecyclerView by bindView(R.id.filters)
59
60     private var noConnection: ImageView? = null
61     private var noFiltersEmptyText: TextView? = null
62     private lateinit var client: PhoenixClient
63     //Internet connectivity
64     private var connected: Boolean = true
65     private var monitoringConnectivity: Boolean = false
66     private var isLoading: Boolean = true
```

```
67    private lateinit var layoutManager: GridLayoutManager
68    private lateinit var adapter: DataAdapter
69    private lateinit var dataManager: DataManager
70    private lateinit var filtersAdapter: FilterAdapter
71
72    private lateinit var auth: FirebaseAuth
73    private lateinit var listener: FirebaseAuth.AuthStateListener
74
75    override fun onCreate(savedInstanceState: Bundle?) {
76        super.onCreate(savedInstanceState)
77        setContentView(R.layout.activity_home)
78
79        //Set action bar for the activity in order to activate the inflated menu
80        setActionBar(toolbar)
81
82        //Init client
83        client = PhoenixClient(this@HomeActivity)
84
85        //Firebase Auth
86        auth = client.auth
87
88        //Add AuthStateListener
89        listener = FirebaseAuth.AuthStateListener { firebaseAuth ->
90            val firebaseUser = firebaseAuth.currentUser
91            if (firebaseUser != null) {
92                invalidateOptionsMenu()
93            }
94        }
95
96        drawer.systemUiVisibility = (View.SYSTEM_UI_FLAG_LAYOUT_STABLE
97                or View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
98                or View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION)
99
100       //Animate the toolbar into view
101       if (savedInstanceState == null) {
102           animateToolbar()
103       }
104       setExitSharedElementCallback(DataAdapter.createSharedElementReenterCallback(this))
105
106       //Set visibility
107       fab.visibility = if (client.isConnected) View.VISIBLE else View.GONE
108
109       //set click action for FAB
110       fab.setOnClickListener {
111           if (client.isLoggedIn) {
112               val intent = Intent(this, CartActivity::class.java)
113               FabTransform.addExtras(intent,
114                       ContextCompat.getColor(this, R.color.accent), R.drawable.ic_shopping_cart_black_24dp)
115               val options = ActivityOptions.makeSceneTransitionAnimation(this, fab,
116                       getString(R.string.transition_new_designer_news_post))
117               startActivityForResult(intent, RC_CART, options.toBundle())
118           } else {
119               val intent = Intent(this, AuthActivity::class.java)
120               FabTransform.addExtras(intent,
121                       ContextCompat.getColor(this, R.color.accent), R.drawable.ic_shopping_cart_black_24dp)
122               val options = ActivityOptions.makeSceneTransitionAnimation(this, fab,
123                       getString(R.string.transition_dribbble_login))
124               startActivityForResult(intent, RC_LOGIN_BASIC, options.toBundle())
125           }
126       }
127
128
129       //Customize drawer layout to support landscape mode
130       drawer.setOnApplyWindowInsetsListener { _, p1 ->
131           // inset the toolbar down by the status bar height
132           val lpToolbar: ViewGroup.MarginLayoutParams = toolbar.layoutParams as ViewGroup.
      MarginLayoutParams
```

```
133         if (p1 != null) {
134             lpToolbar.topMargin += p1.systemWindowInsetTop
135             lpToolbar.leftMargin += p1.systemWindowInsetLeft
136             lpToolbar.rightMargin += p1.systemWindowInsetRight
137             toolbar.layoutParams = lpToolbar
138 
139             // inset the grid top by statusbar+toolbar & the bottom by the navbar (don't clip)
140             grid.setPadding(
141                     grid.paddingLeft + p1.systemWindowInsetLeft, // landscape
142                     p1.systemWindowInsetTop
143                             + ViewUtils.getActionBarSize(this@HomeActivity),
144                     grid.paddingRight + p1.systemWindowInsetRight, // landscape
145                     grid.paddingBottom + p1.systemWindowInsetBottom)
146             // inset the fab for the navbar
147             val lpFab: ViewGroup.MarginLayoutParams = fab.layoutParams as ViewGroup.
    MarginLayoutParams
148             lpFab.bottomMargin += p1.systemWindowInsetBottom // portrait
149             lpFab.rightMargin += p1.systemWindowInsetRight // landscape
150             fab.layoutParams = lpFab
151 
152             // we place a background behind the status bar to combine with it's semi-transparent
153             // color to get the desired appearance.  Set it's height to the status bar height
154             val statusBarBackground: View = findViewById(R.id.status_bar_background)
155             val lpStatus: FrameLayout.LayoutParams = statusBarBackground.layoutParams as
    FrameLayout.LayoutParams
156             lpStatus.height = p1.systemWindowInsetTop
157             statusBarBackground.layoutParams = lpStatus
158 
159             // inset the filters list for the status bar / navbar
160             // need to set the padding end for landscape case
161             val ltr: Boolean = filtersList.layoutDirection == View.LAYOUT_DIRECTION_LTR
162             filtersList.setPaddingRelative(filtersList.paddingStart,
163                     filtersList.paddingTop + p1.systemWindowInsetTop,
164                     filtersList.paddingEnd + (if (ltr) p1.systemWindowInsetRight else 0),
165                     filtersList.paddingBottom + p1.systemWindowInsetBottom)
166         }
167         // clear this listener so insets aren't re-applied
168         drawer.setOnApplyWindowInsetsListener(null)
169         p1?.consumeSystemWindowInsets()!!
170     }
171 
172     //init filters adapter
173     filtersAdapter = FilterAdapter(this@HomeActivity,
174             SourceManager.getSources(this@HomeActivity) as MutableList<Source>, object :
    FilterAuthoriser {
175         override fun requestAuthorisation(sharedElement: View, forSource: Source) {
176             val login = Intent(this@HomeActivity, AuthActivity::class.java)
177             MorphTransform.addExtras(login,
178                     ContextCompat.getColor(this@HomeActivity, R.color.background_dark),
179                     sharedElement.height.div(2))
180             val options =
181                     ActivityOptions.makeSceneTransitionAnimation(this@HomeActivity,
182                             sharedElement, getString(R.string.transition_dribbble_login))
183             startActivityForResult(login,
184                     getAuthSourceRequestCode(forSource), options.toBundle())
185         }
186     })
187 
188     //init dataManager
189     dataManager = object : DataManager(this@HomeActivity, filtersAdapter) {
190         @SuppressLint("LogConditional")
191         override fun onDataLoaded(data: List<ProductItem>) {
192             adapter.addAndResort(data)
193             emptyState()
194         }
195     }
196 
```

```
197         //RecyclerView
198         val columns: Int = resources.getInteger(R.integer.num_columns)
199         val preloadSizeProvider: ViewPreloadSizeProvider<Product> = ViewPreloadSizeProvider()
200         adapter = DataAdapter(this, dataManager, columns, preloadSizeProvider)
201         grid.adapter = adapter
202 //          layoutManager = LinearLayoutManager(this)
203         layoutManager = GridLayoutManager(this, columns)
204         layoutManager.spanSizeLookup = object : GridLayoutManager.SpanSizeLookup() {
205             override fun getSpanSize(position: Int): Int {
206                 return adapter.getItemColumnSpan(position)
207             }
208         }
209         grid.layoutManager = layoutManager
210         grid.addOnScrollListener(toolbarElevation)
211         grid.setHasFixedSize(true)
212         grid.addItemDecoration(GridItemDividerDecoration(this, R.dimen.divider_height, R.color.divider))
213         grid.itemAnimator = SlideInItemAnimator()
214
215         val shotPreloader: RecyclerViewPreloader<Product> = RecyclerViewPreloader<Product>(this,
216                 adapter, preloadSizeProvider, 4)
217         grid.addOnScrollListener(shotPreloader)
218         setupTaskDescription()
219
220         //Filters
221         filtersList.adapter = filtersAdapter
222         filtersList.itemAnimator = FilterAdapter.FilterAnimator()
223         filtersAdapter.registerFilterChangedCallback(filtersChangedCallbacks)
224         dataManager.loadAllDataSources()
225         val callback: ItemTouchHelper.Callback = FilterTouchHelperCallback(filtersAdapter, this)
226         val itemTouchHelper = ItemTouchHelper(callback)
227         itemTouchHelper.attachToRecyclerView(filtersList)
228         emptyState()
229
230     }
231
232     private val filtersChangedCallbacks: FilterAdapter.FiltersChangedCallbacks = object : FilterAdapter.
        FiltersChangedCallbacks() {
233         override fun onFiltersChanged(changedFilter: Source) {
234             if (!changedFilter.active) {
235                 adapter.removeDataSource(changedFilter.key)
236             }
237             emptyState()
238         }
239
240         override fun onFilterRemoved(removed: Source) {
241             adapter.removeDataSource(removed.key)
242             emptyState()
243         }
244     }
245
246     private fun emptyState() {
247         if (adapter.itemCount == 0) {
248             // if grid is empty check whether we're loading or if no filters are selected
249             if (filtersAdapter.enabledSourcesCount > 0) {
250                 if (connected) {
251                     isLoading = connected
252                     loading.visibility = View.VISIBLE
253                     setNoFiltersEmptyTextVisibility(View.GONE)
254                 }
255             } else {
256                 isLoading = false
257                 loading.visibility = View.GONE
258                 setNoFiltersEmptyTextVisibility(View.VISIBLE)
259             }
260             toolbar.translationZ = 0f
261         } else {
262             isLoading = false
```

```
263             noConnection?.visibility = View.GONE
264             loading.visibility = View.GONE
265             setNoFiltersEmptyTextVisibility(View.GONE)
266         }
267     }
268 
269     override fun onStart() {
270         super.onStart()
271         if (filtersAdapter.enabledSourcesCount <= 0) {
272             loading.visibility = View.GONE
273             setNoFiltersEmptyTextVisibility(View.VISIBLE)
274         }
275         auth.addAuthStateListener(listener)
276     }
277 
278     override fun onStop() {
279         auth.removeAuthStateListener(listener)
280         super.onStop()
281     }
282 
283     override fun onDestroy() {
284         dataManager.cancelLoading()
285         super.onDestroy()
286     }
287 
288     private val toolbarElevation: RecyclerView.OnScrollListener = object : RecyclerView.OnScrollListener() {
289         override fun onScrollStateChanged(recyclerView: RecyclerView?, newState: Int) {
290             // we want the grid to scroll over the top of the toolbar but for the toolbar items
291             // to be clickable when visible. To achieve this we play games with elevation. The
292             // toolbar is laid out in front of the grid but when we scroll, we lower it's elevation
293             // to allow the content to pass in front (and reset when scrolled to top of the grid)
294             if (newState == RecyclerView.SCROLL_STATE_IDLE
295                     && layoutManager.findFirstVisibleItemPosition() == 0
296                     && layoutManager.findViewByPosition(0).top == grid.paddingTop
297                     && toolbar.translationZ != 0.0f) {
298                 // at top, reset elevation
299                 toolbar.translationZ = 0.0f
300             } else if (newState == RecyclerView.SCROLL_STATE_DRAGGING
301                     && toolbar.translationZ != -1.0f
302                     && adapter.itemCount != 0) {
303                 // grid scrolled, lower toolbar to allow content to pass in front
304                 toolbar.translationZ = -1.0f
305             }
306         }
307     }
308 
309     override fun onCreateOptionsMenu(menu: Menu?): Boolean {
310         menuInflater.inflate(R.menu.home, menu)
311         return true
312     }
313 
314     override fun onPrepareOptionsMenu(menu: Menu?): Boolean {
315         val notifications: MenuItem? = menu?.findItem(R.id.menu_notifications)
316         val profile: MenuItem? = menu?.findItem(R.id.menu_profile)
317         val login: MenuItem? = menu?.findItem(R.id.menu_login)
318         val orders: MenuItem? = menu?.findItem(R.id.menu_orders)
319 
320         //Setup premium login text
321         if (login != null) {
322             if (client.isLoggedIn) {
323                 login.title = resources.getString(R.string.user_log_out_basic)
324             } else {
325                 login.title = resources.getString(R.string.user_login)
326             }
327         }
328 
329         if (profile != null) {
```

```
330             profile.isEnabled = client.isLoggedIn
331         }
332
333         if (notifications != null) {
334             notifications.isEnabled = client.isLoggedIn
335         }
336
337         if (orders != null) {
338             orders.isEnabled = client.isLoggedIn
339         }
340         return true
341     }
342 ---------------------------------------------------------------------------------------------------------
343     override fun onOptionsItemSelected(item: MenuItem?): Boolean {
344         return when (item?.itemId) {
345             R.id.menu_login -> {
346                 if (client.isLoggedIn) {
347                     val builder = AlertDialog.Builder(this@HomeActivity)
348                     builder.setMessage(getString(R.string.logout_prompt))
349                     builder.setNegativeButton("Logout", { dialogInterface, _ ->
350                         client.logout()
351                         Toast.makeText(this, resources.getString(R.string.user_logged_out), Toast
352                                 .LENGTH_SHORT).show()
353                         dialogInterface.dismiss()
354                     }).setPositiveButton("Cancel", { dialogInterface, _ ->
355                         dialogInterface.cancel()
356                     })
357                     builder.show()
358                 } else {
359                     val intent = Intent(this, AuthActivity::class.java)
360                     FabTransform.addExtras(intent,
361                             ContextCompat.getColor(this, R.color.button_accent),
362                             R.drawable.ic_shopping_cart_black_24dp)
363                     val options = ActivityOptions.makeSceneTransitionAnimation(this, fab,
364                             getString(R.string.transition_dribbble_login))
365                     startActivityForResult(intent, RC_LOGIN_BASIC, options.toBundle())
366                 }
367                 true
368             }
369             R.id.menu_profile -> {
370                 startActivity(Intent(this@HomeActivity, ProfileActivity::class.java))
371                 true
372             }
373             R.id.menu_store -> {
374                 //launch google maps to display the Accra Mall Store
375                 val mapUri = Uri.parse("geo: 5.6227348,-0.1743774")
376                 val mapIntent = Intent(Intent.ACTION_VIEW, mapUri)
377                 mapIntent.`package` = "com.google.android.apps.maps"
378                 startActivity(mapIntent)
379                 true
380             }
381             R.id.menu_filter -> {
382                 drawer.openDrawer(GravityCompat.END)
383                 true
384             }
385             R.id.menu_notifications -> {
386                 //See what's new in The Phoenix
387                 startActivity(Intent(this@HomeActivity, NotificationsActivity::class.java))
388                 true
389             }
390             R.id.menu_orders -> {
391                 //See what's new in The Phoenix
392                 startActivity(Intent(this@HomeActivity, LiveOrdersActivity::class.java))
393                 true
394             }
395             R.id.menu_search -> {
396                 val searchMenuView: View = toolbar.findViewById(R.id.menu_search)
```

```
397             val options = ActivityOptions.makeSceneTransitionAnimation(this, searchMenuView,
398                     getString(R.string.transition_search_back)).toBundle()
399             startActivityForResult(Intent(this@HomeActivity, SearchActivity::class.java), RC_SEARCH,
400                     options)
401             true
402         }
403         R.id.menu_about -> {
404             startActivity(Intent(this@HomeActivity, AboutActivity::class.java),
405                     ActivityOptions.makeSceneTransitionAnimation(this).toBundle())
406             true
407         }
408         else -> return super.onOptionsItemSelected(item)
409     }
410 }
411 
412 private fun animateToolbar() {
413     // this is gross but toolbar doesn't expose it's children to animate them :(
414     val t = toolbar.getChildAt(0)
415     if (t != null && t is TextView) {
416 
417         // fade in and space out the title.  Animating the letterSpacing performs horribly so
418         // fake it by setting the desired letterSpacing then animating the scaleX ¯\_(ツ)_/¯
419         t.alpha = 0f
420         t.scaleX = 0.8f
421 
422         t.animate()
423                 .alpha(1f)
424                 .scaleX(1f)
425                 .setStartDelay(300)
426                 .setDuration(600).interpolator = AnimUtils.getFastOutSlowInInterpolator(this@HomeActivity)
427     }
428 }
429 
430 private fun setupTaskDescription() {
431     val overviewIcon = BitmapFactory.decodeResource(resources, applicationInfo.icon)
432     setTaskDescription(ActivityManager.TaskDescription(getString(R.string.app_name),
433             overviewIcon,
434             ContextCompat.getColor(this, R.color.primary)))
435     if (overviewIcon != null) {
436         overviewIcon.recycle()
437     }
438 }
439 
440 override fun onActivityReenter(resultCode: Int, data: Intent?) {
441     if (data == null || resultCode != RESULT_OK
442             || !data.hasExtra(DetailsActivity.RESULT_EXTRA_SHOT_ID)) return
443     // When reentering, if the shared element is no longer on screen (e.g. after an
444     // orientation change) then scroll it into view.
445     val sharedShotId = data.getLongExtra(DetailsActivity.RESULT_EXTRA_SHOT_ID, -1L)
446     if (sharedShotId != -1L                                 // returning from a shot
447             && adapter.dataItemCount > 0                    // grid populated
448             && grid.findViewHolderForItemId(sharedShotId) == null) {    // view not attached
449         val position = adapter.getItemPosition(sharedShotId)
450         if (position == RecyclerView.NO_POSITION) return
451 
452         // delay the transition until our shared element is on-screen i.e. has been laid out
453         postponeEnterTransition()
454         grid.addOnLayoutChangeListener(object : View.OnLayoutChangeListener {
455             override fun onLayoutChange(p0: View?, p1: Int, p2: Int, p3: Int, p4: Int, p5: Int, p6: Int, p7: Int, p8:
    Int) {
456                 grid.removeOnLayoutChangeListener(this)
457                 startPostponedEnterTransition()
458             }
459         })
460         grid.scrollToPosition(position)
461         toolbar.translationZ = -1f
462     }
```

```
463     }
464
465     override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
466         when (requestCode) {
467             RC_LOGIN_BASIC -> {
468                 if (resultCode == RESULT_OK) {
469                     showFab()
470                 }
471             }
472             RC_AUTH_SOURCE_BUSINESS -> {
473                 if (resultCode == RESULT_OK) {
474                     filtersAdapter.enableFilterByKey(SourceManager.SOURCE_BUSINESS, this)
475                 }
476             }
477             RC_AUTH_SOURCE_FAV -> {
478                 if (resultCode == RESULT_OK) {
479                     filtersAdapter.enableFilterByKey(SourceManager.SOURCE_FAVORITE, this)
480                 }
481             }
482             RC_AUTH_SOURCE_STUDENT -> {
483                 if (resultCode == RESULT_OK) {
484                     filtersAdapter.enableFilterByKey(SourceManager.SOURCE_STUDENT, this)
485                 }
486             }
487             RC_AUTH_SOURCE_HEALTH -> {
488                 if (resultCode == RESULT_OK) {
489                     filtersAdapter.enableFilterByKey(SourceManager.SOURCE_HEALTH, this)
490                 }
491             }
492             RC_SEARCH -> {
493                 // reset the search icon which we hid
494                 val searchMenuView: View? = toolbar.findViewById(R.id.menu_search)
495                 if (searchMenuView != null) {
496                     searchMenuView.alpha = 1.0f
497                 }
498                 if (resultCode == SearchActivity.RESULT_CODE_SAVE) {
499                     val query: String? = data?.getStringExtra(SearchActivity.EXTRA_QUERY)
500                     if (query == null || TextUtils.isEmpty(query)) return
501                     Timber.d(query)
502                     var source: Source? = null
503                     var newSource = false
504                     if (data.getBooleanExtra(SearchActivity.EXTRA_SAVE, false)) {
505                         source = Source.PhoenixSearchSource(query, true)
506                         newSource = filtersAdapter.addFilter(source)
507                     }
508                     if (newSource) highlightNewSources(source)
509                 }
510             }
511             RC_CART -> {
512                 if (resultCode == RESULT_OK) {
513                     showFab()
514                 }
515             }
516         }
517     }
518
519     override fun onSaveInstanceState(savedInstanceState: Bundle) {
520         super.onSaveInstanceState(savedInstanceState)
521         savedInstanceState.putBoolean(STATE_LOADING, isLoading)
522     }
523
524     private fun showFab() {
525         fab.alpha = 0f
526         fab.scaleX = 0f
527         fab.scaleY = 0f
528         fab.translationY = (fab.height / 2).toFloat()
529         fab.animate()
```

```
            .alpha(1f)
            .scaleX(1f)
            .scaleY(1f)
            .translationY(0f)
            .setDuration(300L)
            .setInterpolator(AnimUtils.getLinearOutSlowInInterpolator(this@HomeActivity))
            .start()
    }

    private fun setNoFiltersEmptyTextVisibility(visibility: Int) {
        if (visibility == View.VISIBLE) {
            if (noFiltersEmptyText == null) {
                // create the no filters empty text
                val stub = findViewById<ViewStub>(R.id.stub_no_filters)
                noFiltersEmptyText = stub.inflate() as TextView
                val emptyText = getText(R.string.no_filters_selected) as SpannedString
                val ssb = SpannableStringBuilder(emptyText)
                val annotations = emptyText.getSpans(0, emptyText.length, Annotation::class.java)
                if (annotations != null && annotations.isNotEmpty()) {
                    for (i in annotations.indices) {
                        val annotation = annotations[i]
                        if (annotation.key == "src") {
                            // image span markup
                            val name = annotation.value
                            val id = resources.getIdentifier(name, null, packageName)
                            if (id == 0) continue
                            ssb.setSpan(ImageSpan(this, id,
                                    ImageSpan.ALIGN_BASELINE),
                                    emptyText.getSpanStart(annotation),
                                    emptyText.getSpanEnd(annotation),
                                    Spannable.SPAN_EXCLUSIVE_EXCLUSIVE)
                        } else if (annotation.key == "foregroundColor") {
                            // foreground color span markup
                            val name = annotation.value
                            val id = resources.getIdentifier(name, null, packageName)
                            if (id == 0) continue
                            ssb.setSpan(ForegroundColorSpan(ContextCompat.getColor(this, id)),
                                    emptyText.getSpanStart(annotation),
                                    emptyText.getSpanEnd(annotation),
                                    Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)
                        }
                    }
                }
                noFiltersEmptyText!!.text = ssb
                noFiltersEmptyText!!.setOnClickListener { _ -> drawer.openDrawer(GravityCompat.END) }
            }
            noFiltersEmptyText!!.visibility = visibility
        } else if (noFiltersEmptyText != null) {
            noFiltersEmptyText!!.visibility = visibility
        }

    }

    override fun onBackPressed() {
        if (drawer.isDrawerOpen(GravityCompat.END))
            drawer.closeDrawer(GravityCompat.END)
        else {
            super.onBackPressed()
            //finishAfterTransition()
        }
    }

    override fun onResume() {
        super.onResume()
        //Add login listener
        //todo: seems not to respond to auth state
        client.addLoginStatusListener(filtersAdapter)
```

```
597
598         //There seems to be a problem with the AuthStateListener so we need to manually invalidate
599         //the options menu
600         invalidateOptionsMenu()
601         checkConnectivity()
602     }
603 
604     override fun onPause() {
605         client.removeLoginStatusListener(filtersAdapter)
606         if (monitoringConnectivity) {
607             val connectivityManager: ConnectivityManager = getSystemService(Context.
    CONNECTIVITY_SERVICE) as ConnectivityManager
608             connectivityManager.unregisterNetworkCallback(connectivityCallback)
609             monitoringConnectivity = false
610         }
611         super.onPause()
612     }
613 
614     private fun checkConnectivity() {
615         val connectivityManager: ConnectivityManager = getSystemService(Context.
    CONNECTIVITY_SERVICE) as ConnectivityManager
616         val activeNetworkInfo: NetworkInfo? = connectivityManager.activeNetworkInfo
617         connected = activeNetworkInfo != null && activeNetworkInfo.isConnectedOrConnecting
618         if (!connected) {
619             loading.visibility = View.GONE
620             fab.visibility = View.GONE
621             if (noConnection == null) {
622                 val stub: ViewStub = findViewById(R.id.stub_no_connection)
623                 noConnection = stub.inflate() as ImageView
624             }
625             val avd: AnimatedVectorDrawable? = getDrawable(R.drawable.avd_no_connection) as
626                     AnimatedVectorDrawable
627             if (noConnection != null && avd != null) {
628                 noConnection!!.setImageDrawable(avd)
629                 avd.start()
630             }
631 
632             connectivityManager.registerNetworkCallback(
633                     NetworkRequest.Builder()
634                             .addCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET).build(),
635                     connectivityCallback)
636             monitoringConnectivity = true
637         }
638     }
639 
640     private val connectivityCallback: ConnectivityManager.NetworkCallback = object :
    ConnectivityManager.NetworkCallback() {
641         override fun onAvailable(network: Network?) {
642             connected = true
643             if (adapter.dataItemCount != 0) return
644             runOnUiThread({
645                 TransitionManager.beginDelayedTransition(drawer)
646                 noConnection?.visibility = View.GONE
647                 loading.visibility = View.VISIBLE
648                 fab.visibility = View.VISIBLE
649                 dataManager.loadAllDataSources()
650             })
651         }
652 
653         override fun onLost(network: Network?) {
654             connected = false
655         }
656     }
657 
658     /**
659      * get user authentication to enable access to filters from
660      * @param filter */
```

```
661     private fun getAuthSourceRequestCode(filter: Source): Int {
662         when (filter.key) {
663             SourceManager.SOURCE_HEALTH -> return RC_AUTH_SOURCE_HEALTH
664             SourceManager.SOURCE_BUSINESS -> return RC_AUTH_SOURCE_BUSINESS
665             SourceManager.SOURCE_FAVORITE -> return RC_AUTH_SOURCE_FAV
666             SourceManager.SOURCE_STUDENT -> return RC_AUTH_SOURCE_STUDENT
667         }
668         throw InvalidParameterException()
669     }
670 
671     private fun highlightNewSources(vararg sources: Source?) {
672         val closeDrawerRunnable = Runnable {
673             drawer.closeDrawer(GravityCompat.END)
674         }
675 
676         drawer.addDrawerListener(object : DrawerLayout.SimpleDrawerListener() {
677             private val filtersTouch: View.OnTouchListener = View.OnTouchListener { _, _ ->
678                 drawer.removeCallbacks(closeDrawerRunnable)
679                 false
680             }
681 
682             override fun onDrawerClosed(drawerView: View) {
683                 // reset
684                 filtersList.setOnTouchListener(null)
685                 drawer.removeDrawerListener(this)
686             }
687 
688             override fun onDrawerOpened(drawerView: View) {
689                 // scroll to the new item(s) and highlight them
690                 val filterPositions: ArrayList<Int> = ArrayList(sources.size)
691                 sources
692                         .filterNotNull()
693                         .mapTo(filterPositions) { filtersAdapter.getFilterPosition(it) }
694                 val scrollTo: Int = Collections.max(filterPositions)
695                 filtersList.smoothScrollToPosition(scrollTo)
696                 for (position in filterPositions) {
697                     filtersAdapter.highlightFilter(position)
698                 }
699                 filtersList.setOnTouchListener(filtersTouch)
700             }
701 
702             override fun onDrawerStateChanged(newState: Int) {
703                 // if the user interacts with the drawer manually then don't auto-close
704                 if (newState == DrawerLayout.STATE_DRAGGING) {
705                     drawer.removeCallbacks(closeDrawerRunnable)
706                 }
707             }
708 
709         })
710         drawer.openDrawer(GravityCompat.END)
711         drawer.postDelayed(closeDrawerRunnable, 2000L)
712     }
713 
714     companion object {
715         private const val RC_SEARCH = 1234
716         private const val STATE_LOADING = "STATE_LOADING"
717         private const val RC_LOGIN_BASIC = 450
718         private const val RC_CART = 500
719         private const val RC_AUTH_SOURCE_FAV = 19
720         private const val RC_AUTH_SOURCE_BUSINESS = 98
721         private const val RC_AUTH_SOURCE_STUDENT = 999
722         private const val RC_AUTH_SOURCE_HEALTH = 1099
723     }
724 
725 }
726 
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import `in`.uncod.android.bypass.Bypass
8  import android.annotation.SuppressLint
9  import android.app.Activity
10 import android.content.res.Resources
11 import android.net.Uri
12 import android.os.Bundle
13 import android.support.customtabs.CustomTabsIntent
14 import android.support.v4.content.ContextCompat
15 import android.support.v4.view.PagerAdapter
16 import android.support.v4.view.ViewPager
17 import android.support.v7.widget.RecyclerView
18 import android.text.Layout
19 import android.text.SpannableString
20 import android.text.Spanned
21 import android.text.TextUtils
22 import android.text.style.AlignmentSpan
23 import android.transition.TransitionInflater
24 import android.view.LayoutInflater
25 import android.view.View
26 import android.view.ViewGroup
27 import android.widget.Button
28 import android.widget.ImageView
29 import android.widget.TextView
30 import com.bumptech.glide.load.engine.DiskCacheStrategy
31 import com.bumptech.glide.load.resource.drawable.DrawableTransitionOptions.withCrossFade
32 import com.bumptech.glide.request.RequestOptions
33 import com.bumptech.glide.request.target.Target
34 import io.pergasus.R
35 import io.pergasus.api.PhoenixClient
36 import io.pergasus.ui.widget.CircularImageView
37 import io.pergasus.ui.widget.ElasticDragDismissFrameLayout
38 import io.pergasus.ui.widget.FourThreeImageView
39 import io.pergasus.ui.widget.InkPageIndicator
40 import io.pergasus.util.HtmlUtils
41 import io.pergasus.util.bindView
42 import io.pergasus.util.customtabs.CustomTabActivityHelper
43 import io.pergasus.util.glide.GlideApp
44 import org.jetbrains.annotations.Nullable
45 import java.security.InvalidParameterException
46
47
48 /** About the application */
49 class AboutActivity : Activity() {
50
51     private val draggableFrame: ElasticDragDismissFrameLayout by bindView(R.id.draggable_frame)
52     private val pager: ViewPager by bindView(R.id.pager)
53     private val pageIndicator: InkPageIndicator by bindView(R.id.indicator)
54
55     private lateinit var client: PhoenixClient
56 ____________________________________________________________________________________________
57     override fun onCreate(savedInstanceState: Bundle?) {
58         super.onCreate(savedInstanceState)
59         setContentView(R.layout.activity_about)
60
61         client = PhoenixClient(this@AboutActivity)
62
63         pager.adapter = AboutPagerAdapter()
64         pager.pageMargin = resources.getDimensionPixelSize(R.dimen.spacing_normal)
65         pageIndicator.setViewPager(pager)
66
```

```
67      draggableFrame.addListener(object : ElasticDragDismissFrameLayout.ElasticDragDismissCallback()
    {
68          override fun onDragDismissed() {
69              // if we drag dismiss downward then the default reversal of the enter
70              // transition would slide content upward which looks weird. So reverse it.
71              if (draggableFrame.translationY > 0) {
72                  window.returnTransition = TransitionInflater.from(this@AboutActivity)
73                          .inflateTransition(R.transition.about_return_downward)
74              }
75              finishAfterTransition()
76          }
77      })
78
79  }
80
81  internal inner class AboutPagerAdapter : PagerAdapter() {
82      private val PAGES: Int = 4
83      private var layoutInflater: LayoutInflater = LayoutInflater.from(this@AboutActivity)
84      private val markdown: Bypass = Bypass(this@AboutActivity, Bypass.Options())
85      private val resources: Resources = this@AboutActivity.resources
86
87      private var aboutPhoenix: View? = null
88      @Nullable
89      private var phoenixDescription: TextView? = null
90      private var aboutIcon: View? = null
91      @Nullable
92      private var iconDescription: TextView? = null
93      @Nullable
94      var icon: FourThreeImageView? = null
95      private var aboutLibs: View? = null
96      private var libsList: RecyclerView? = null
97      private var aboutDeveloper: View? = null
98      @Nullable
99      var profile: CircularImageView? = null
100     @Nullable
101     var follow: Button? = null
102
103
104     override fun isViewFromObject(view: View, `object`: Any): Boolean {
105         return view == `object`
106     }
107
108     override fun instantiateItem(container: ViewGroup, position: Int): Any {
109         val layout = getPage(position, container)
110         container.addView(layout)
111         return layout
112     }
113
114     private fun getPage(position: Int, container: ViewGroup): View {
115         return when (position) {
116             0 -> {
117                 if (aboutPhoenix == null) {
118                     aboutPhoenix = layoutInflater.inflate(R.layout.about_app, container, false)
119                     phoenixDescription = aboutPhoenix?.findViewById(R.id.about_description)
120                     // fun with spans & markdown
121                     val about0: CharSequence = markdown.markdownToSpannable(
122                             resources.getString(R.string.about_app_0), phoenixDescription, null)
123                     val about1 = SpannableString(
124                             resources.getString(R.string.about_app_1))
125                     about1.setSpan(AlignmentSpan.Standard(Layout.Alignment.ALIGN_CENTER),
126                             0, about1.length, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)
127                     val about2 = SpannableString(markdown.markdownToSpannable
128                     (resources.getString(R.string.about_app_2),
129                             phoenixDescription, null))
130                     about2.setSpan(AlignmentSpan.Standard(Layout.Alignment.ALIGN_CENTER),
131                             0, about2.length, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)
132                     val desc: CharSequence = TextUtils.concat(about0, "\n\n", about1, "\n", about2, "\n\n")
```

```
                HtmlUtils.setTextWithNiceLinks(phoenixDescription, desc)
            }
            this.aboutPhoenix!!
        }
        1 -> {
            if (aboutIcon == null) {
                aboutIcon = layoutInflater.inflate(R.layout.about_icon, container, false)
                iconDescription = aboutIcon?.findViewById(R.id.icon_description)
                icon = aboutIcon?.findViewById(R.id.icon)
                val icon0 = resources.getString(R.string.about_icon_0)
                val icon2 = resources.getString(R.string.about_accra_mall)
                val icon1 = markdown.markdownToSpannable(resources
                        .getString(R.string.about_icon_1), iconDescription, null)
                val iconDesc = TextUtils.concat(icon2, "\n\n", icon0, "\n", icon1)
                HtmlUtils.setTextWithNiceLinks(iconDescription, iconDesc)
            }
            this.aboutIcon!!
        }
        2 -> {
            if (aboutLibs == null) {
                aboutLibs = layoutInflater.inflate(R.layout.about_libs, container, false)
                libsList = aboutLibs?.findViewById(R.id.libs_list)
                libsList?.adapter = LibraryAdapter(this@AboutActivity)
            }
            this.aboutLibs!!
        }
        3 -> {
            if (aboutDeveloper == null) {
                aboutDeveloper = layoutInflater.inflate(R.layout.about_developer, container, false)
                profile = aboutDeveloper?.findViewById(R.id.developer_profile)
                follow = aboutDeveloper?.findViewById(R.id.follow)

                //Follow developer on github
                follow?.setOnClickListener({
                    CustomTabActivityHelper.openCustomTab(
                            this@AboutActivity,
                            CustomTabsIntent.Builder()
                                    .setToolbarColor(ContextCompat.getColor(this@AboutActivity,
                                            R.color.background_super_dark))
                                    .addDefaultShareMenuItem()
                                    .build(), Uri.parse("https://github.com/Quabynah"))
                })

                //Load developer profile image from resource
                GlideApp.with(profile!!.context)
                        .load(getString(R.string.quabynah_url))
                        .circleCrop()
                        .override(Target.SIZE_ORIGINAL, Target.SIZE_ORIGINAL)
                        .diskCacheStrategy(DiskCacheStrategy.AUTOMATIC)
                        .placeholder(R.drawable.avatar_placeholder)
                        .error(R.drawable.avatar_placeholder)
                        .fallback(R.drawable.avatar_placeholder)
                        .into(profile!!)
            }
            this.aboutDeveloper!!
        }
        else -> throw InvalidParameterException("View not implemented")
    }

}

override fun destroyItem(container: ViewGroup, position: Int, `object`: Any) {
    container.removeView(`object` as View)
}

override fun getCount(): Int {
    return PAGES
```

```
200         }
201     }
202
203     private class LibraryAdapter internal constructor(internal val host: Activity) : RecyclerView.Adapter<
        RecyclerView.ViewHolder>() {
204
205         override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
206             when (viewType) {
207                 VIEW_TYPE_INTRO -> return LibraryIntroHolder(LayoutInflater.from(parent.context)
208                         .inflate(R.layout.about_lib_intro, parent, false))
209                 VIEW_TYPE_LIBRARY -> return createLibraryHolder(parent)
210             }
211             throw InvalidParameterException()
212         }
213
214         private fun createLibraryHolder(parent: ViewGroup): LibraryHolder {
215             val holder = LibraryHolder(LayoutInflater.from(parent.context)
216                     .inflate(R.layout.library, parent, false))
217             val clickListener = View.OnClickListener {
218                 val position = holder.adapterPosition
219                 if (position == RecyclerView.NO_POSITION) return@OnClickListener
220                 CustomTabActivityHelper.openCustomTab(
221                         host,
222                         CustomTabsIntent.Builder()
223                                 .setToolbarColor(ContextCompat.getColor(host, R.color.background_super_dark))
224                                 .addDefaultShareMenuItem()
225                                 .build(), Uri.parse(libs[position - 1].link))
226             }
227             holder.itemView.setOnClickListener(clickListener)
228             holder.link.setOnClickListener(clickListener)
229             return holder
230         }
231
232         override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
233             if (getItemViewType(position) == VIEW_TYPE_LIBRARY) {
234                 bindLibrary(holder as LibraryHolder, libs[position - 1]) // adjust for intro
235             }
236         }
237
238         override fun getItemViewType(position: Int): Int {
239             return if (position == 0) VIEW_TYPE_INTRO else VIEW_TYPE_LIBRARY
240         }
241
242         override fun getItemCount(): Int {
243             return libs.size + 1 // + 1 for the static intro view
244         }
245
246         @SuppressLint("CheckResult")
247         private fun bindLibrary(holder: LibraryHolder, lib: Library) {
248             holder.name.text = lib.name
249             holder.description.text = lib.description
250             val request = GlideApp.with(holder.image.context)
251                     .load(lib.imageUrl)
252                     .apply(RequestOptions().placeholder(R.drawable.avatar_placeholder))
253                     .apply(RequestOptions().diskCacheStrategy(DiskCacheStrategy.ALL))
254                     .transition(withCrossFade())
255             if (lib.circleCrop) {
256                 request.apply(RequestOptions().optionalCircleCrop())
257             }
258             request.into(holder.image)
259         }
260
261         companion object {
262
263             private const val VIEW_TYPE_INTRO = 0
264             private const val VIEW_TYPE_LIBRARY = 1
265             internal val libs = arrayOf(
```

```
266                 Library("Android support libraries",
267                         "The Android support libraries offer a number of features that are not built into the
    framework.",
268                         "https://developer.android.com/topic/libraries/support-library",
269                         "https://developer.android.com/images/android_icon_125.png",
270                         false),
271                 Library("ButterKnife",
272                         "Bind Android views and callbacks to fields and methods.",
273                         "http://jakewharton.github.io/butterknife/",
274                         "https://avatars.githubusercontent.com/u/66577",
275                         true),
276                 Library("Bypass",
277                         "Skip the HTML, Bypass takes markdown and renders it directly.",
278                         "https://github.com/Uncodin/bypass",
279                         "https://avatars.githubusercontent.com/u/1072254",
280                         true),
281                 Library("Glide",
282                         "An image loading and caching library for Android focused on smooth scrolling.",
283                         "https://github.com/bumptech/glide",
284                         "https://avatars.githubusercontent.com/u/423539",
285                         false),
286                 Library("JSoup",
287                         "Java HTML Parser, with best of DOM, CSS, and jquery.",
288                         "https://github.com/jhy/jsoup/",
289                         "https://avatars.githubusercontent.com/u/76934",
290                         true),
291                 Library("Firebase",
292                         "Backend for the application",
293                         "https://github.com/jhy/jsoup/",
294                         "https://avatars.githubusercontent.com/u/76934",
295                         true),
296                 Library("Hubtel",
297                         "Mobile Money Provider for \'The Phoenix\'",
298                         "https://github.com/jhy/jsoup/",
299                         "https://avatars.githubusercontent.com/u/76934",
300                         true),
301                 Library("OkHttp",
302                         "An HTTP & HTTP/2 client for Android and Java applications.",
303                         "http://square.github.io/okhttp/",
304                         "https://avatars.githubusercontent.com/u/82592",
305                         false),
306                 Library("Retrofit",
307                         "A type-safe HTTP client for Android and Java.",
308                         "http://square.github.io/retrofit/",
309                         "https://avatars.githubusercontent.com/u/82592",
310                         false))
311         }
312     }
313
314     internal class LibraryHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
315
316         var image: ImageView = itemView.findViewById(R.id.library_image)
317         var name: TextView = itemView.findViewById(R.id.library_name)
318         var description: TextView = itemView.findViewById(R.id.library_description)
319         var link: Button = itemView.findViewById(R.id.library_link)
320     }
321
322     internal class LibraryIntroHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
323
324         var intro: TextView = itemView as TextView
325
326     }
327
328     /**
329      * Models an open source library we want to credit
330      */
331     private class Library internal constructor(internal val name: String, internal val description: String,
```

```
                                internal val link: String, internal val imageUrl: String,
                                internal val circleCrop: Boolean)

}
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui;
6
7  import android.animation.Animator;
8  import android.animation.AnimatorListenerAdapter;
9  import android.animation.ObjectAnimator;
10 import android.animation.ValueAnimator;
11 import android.animation.ValueAnimator.AnimatorUpdateListener;
12 import android.annotation.SuppressLint;
13 import android.app.Activity;
14 import android.app.ActivityOptions;
15 import android.app.SharedElementCallback;
16 import android.content.Context;
17 import android.content.Intent;
18 import android.content.res.TypedArray;
19 import android.graphics.Bitmap;
20 import android.graphics.Color;
21 import android.graphics.ColorMatrixColorFilter;
22 import android.graphics.drawable.ColorDrawable;
23 import android.graphics.drawable.Drawable;
24 import android.graphics.drawable.TransitionDrawable;
25 import android.support.annotation.ColorInt;
26 import android.support.annotation.NonNull;
27 import android.support.annotation.Nullable;
28 import android.support.v4.content.ContextCompat;
29 import android.support.v7.widget.RecyclerView;
30 import android.support.v7.widget.RecyclerView.ViewHolder;
31 import android.transition.Transition;
32 import android.transition.TransitionInflater;
33 import android.util.Log;
34 import android.util.Pair;
35 import android.view.LayoutInflater;
36 import android.view.MotionEvent;
37 import android.view.View;
38 import android.view.View.OnClickListener;
39 import android.view.ViewGroup;
40 import android.widget.Button;
41 import android.widget.ProgressBar;
42 import android.widget.TextView;
43 import android.widget.Toast;
44
45 import com.bumptech.glide.ListPreloader;
46 import com.bumptech.glide.RequestBuilder;
47 import com.bumptech.glide.load.DataSource;
48 import com.bumptech.glide.load.engine.DiskCacheStrategy;
49 import com.bumptech.glide.load.engine.GlideException;
50 import com.bumptech.glide.load.resource.gif.GifDrawable;
51 import com.bumptech.glide.request.RequestListener;
52 import com.bumptech.glide.request.RequestOptions;
53 import com.bumptech.glide.request.target.Target;
54 import com.bumptech.glide.util.ViewPreloadSizeProvider;
55 import com.google.android.gms.tasks.OnCompleteListener;
56 import com.google.android.gms.tasks.Task;
57 import com.google.firebase.firestore.DocumentSnapshot;
58 import com.google.firebase.firestore.QuerySnapshot;
59
60 import java.text.NumberFormat;
61 import java.util.ArrayList;
62 import java.util.Collections;
63 import java.util.HashMap;
64 import java.util.List;
65 import java.util.Locale;
66 import java.util.Map;
```

```
 67  import java.util.Objects;
 68
 69  import io.pergasus.BuildConfig;
 70  import io.pergasus.R;
 71  import io.pergasus.api.DataLoadingSubject;
 72  import io.pergasus.api.PhoenixClient;
 73  import io.pergasus.api.PhoenixUtils;
 74  import io.pergasus.api.ProductItem;
 75  import io.pergasus.api.ProductItemSorting.NaturalOrderWeigher;
 76  import io.pergasus.api.ProductItemSorting.ProductItemComparator;
 77  import io.pergasus.api.ProductItemSorting.ProductItemGroupWeigher;
 78  import io.pergasus.api.ProductWeigher;
 79  import io.pergasus.data.Order;
 80  import io.pergasus.data.Product;
 81  import io.pergasus.ui.widget.BadgedFourThreeImageView;
 82  import io.pergasus.util.ObservableColorMatrix;
 83  import io.pergasus.util.ShareProductTask;
 84  import io.pergasus.util.TransitionUtils;
 85  import io.pergasus.util.ViewUtils;
 86  import io.pergasus.util.glide.GlideApp;
 87  import timber.log.Timber;
 88
 89  import static com.bumptech.glide.load.resource.bitmap.BitmapTransitionOptions.withCrossFade;
 90  import static io.pergasus.util.AnimUtils.getFastOutSlowInInterpolator;
 91
 92  /**
 93   * Adapter for displaying a grid of {@link ProductItem}s.
 94   */
 95  @SuppressLint("LogConditional")
 96  public class DataAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder>
 97          implements DataLoadingSubject.DataLoadingCallbacks,
 98          ListPreloader.PreloadModelProvider<Product> {
 99
100      static final int REQUEST_CODE_VIEW_PRODUCT = 5407;
101      private static final String TAG = "DataAdapter";
102
103      private static final int TYPE_PRODUCT = 0;
104      private static final int TYPE_LOADING_MORE = -1;
105
106      // we need to hold on to an activity ref for the shared element transitions :/
107      final Activity host;
108      private final LayoutInflater layoutInflater;
109      private final ProductItemComparator comparator;
110      @Nullable
111      private final DataLoadingSubject dataLoading;
112      private final int columns;
113      private final ColorDrawable[] shotLoadingPlaceholders;
114      private final ViewPreloadSizeProvider<Product> shotPreloadSizeProvider;
115
116      @ColorInt
117      private final
118      int initialGifBadgeColor;
119      private final List<ProductItem> items;
120      private boolean showLoadingMore;
121      private NaturalOrderWeigher naturalOrderWeigher;
122      private ProductWeigher shotWeigher;
123  ------------------------------------------------------------------------------
124      //Constructor
125      public DataAdapter(Activity hostActivity,
126                         @Nullable DataLoadingSubject dataLoading,
127                         int columns,
128                         ViewPreloadSizeProvider<Product> shotPreloadSizeProvider) {
129          this.host = hostActivity;
130          this.dataLoading = dataLoading;
131          if (dataLoading != null) {
132              dataLoading.registerCallback(this);
133          }
```

```
134         this.columns = columns;
135         this.shotPreloadSizeProvider = shotPreloadSizeProvider;
136         layoutInflater = LayoutInflater.from(host);
137         comparator = new ProductItemComparator();
138         items = new ArrayList<>(0);
139         setHasStableIds(true);
140
141         // get the dribbble shot placeholder colors & badge color from the theme
142         TypedArray a = host.obtainStyledAttributes(R.styleable.DribbbleFeed);
143         int loadingColorArrayId =
144                 a.getResourceId(R.styleable.DribbbleFeed_shotLoadingPlaceholderColors, 0);
145         if (loadingColorArrayId == 0) {
146             shotLoadingPlaceholders = new ColorDrawable[]{new ColorDrawable(Color.DKGRAY)};
147         } else {
148             int[] placeholderColors = host.getResources().getIntArray(loadingColorArrayId);
149             shotLoadingPlaceholders = new ColorDrawable[placeholderColors.length];
150             for (int i = 0; i < placeholderColors.length; i++) {
151                 shotLoadingPlaceholders[i] = new ColorDrawable(placeholderColors[i]);
152             }
153         }
154         int initialGifBadgeColorId =
155                 a.getResourceId(R.styleable.DribbbleFeed_initialBadgeColor, 0);
156         initialGifBadgeColor = initialGifBadgeColorId == 0 ? 0x40ffffff : ContextCompat.getColor(host,
    initialGifBadgeColorId);
157         a.recycle();
158     }
159
160     @Override
161     public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
162         switch (viewType) {
163             case TYPE_PRODUCT:
164                 return createProductHolder(parent);
165             case TYPE_LOADING_MORE:
166                 return new LoadingMoreHolder(
167                         layoutInflater.inflate(R.layout.infinite_loading, parent, false));
168         }
169         return null;
170     }
171
172     private ProductViewHolder createProductHolder(ViewGroup parent) {
173         ProductViewHolder holder = new ProductViewHolder(layoutInflater.inflate(R.layout.
    dribbble_shot_item,
174                 parent, false));
175
176         holder.image.setBadgeColor(initialGifBadgeColor);
177         holder.image.setOnClickListener(new OnClickListener() {
178             @Override
179             public void onClick(View view) {
180                 //show details view
181                 Intent intent = new Intent();
182                 intent.setClass(host, DetailsActivity.class);
183                 intent.putExtra(DetailsActivity.EXTRA_SHOT,
184                         (Product) DataAdapter.this.getItem(holder.getAdapterPosition()));
185                 DataAdapter.this.setGridItemContentTransitions(holder.image);
186                 ActivityOptions options =
187                         ActivityOptions.makeSceneTransitionAnimation(host,
188                                 Pair.create(view, host.getString(R.string.transition_shot)),
189                                 Pair.create(view, host.getString(R.string
190                                         .transition_shot_background)));
191                 host.startActivityForResult(intent, REQUEST_CODE_VIEW_PRODUCT, options.toBundle());
192             }
193         });
194         // play animated GIFs whilst touched
195         holder.image.setOnTouchListener(new View.OnTouchListener() {
196             @Override
197             public boolean onTouch(View v, MotionEvent event) {
198                 // check if it's an event we care about, else bail fast
```

```
            int action = event.getAction();
            if (!(action == MotionEvent.ACTION_DOWN
                    || action == MotionEvent.ACTION_UP
                    || action == MotionEvent.ACTION_CANCEL)) return false;

            // get the image and check if it's an animated GIF
            Drawable drawable = holder.image.getDrawable();
            if (drawable == null) return false;
            GifDrawable gif = null;
            if (drawable instanceof GifDrawable) {
                gif = (GifDrawable) drawable;
            } else if (drawable instanceof TransitionDrawable) {
                // we fade in images on load which uses a TransitionDrawable; check its layers
                TransitionDrawable fadingIn = (TransitionDrawable) drawable;
                for (int i = 0; i < fadingIn.getNumberOfLayers(); i++) {
                    if (fadingIn.getDrawable(i) instanceof GifDrawable) {
                        gif = (GifDrawable) fadingIn.getDrawable(i);
                        break;
                    }
                }
            }
            if (gif == null) return false;
            // GIF found, start/stop it on press/lift
            switch (action) {
                case MotionEvent.ACTION_DOWN:
                    gif.start();
                    break;
                case MotionEvent.ACTION_UP:
                case MotionEvent.ACTION_CANCEL:
                    gif.stop();
                    break;
            }
            return false;
        }
    });

    PhoenixClient prefs = new PhoenixClient(host);

    if (!prefs.isLoggedIn()) {
        holder.add.setVisibility(View.GONE);
    }

    holder.add.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View view) {
            Map<String, Object> hashMap = new HashMap<>(0); //Create empty map
            Product product = (Product) items.get(holder.getAdapterPosition()); //init product

            //Populate map with data
            hashMap.put("id", product.getId());
            hashMap.put("name", product.getName());
            hashMap.put("image", product.getUrl());
            hashMap.put("price", product.getPrice());
            hashMap.put("quantity", product.getQuantity());

            //Push to database
            if (prefs.isLoggedIn() && prefs.getCustomer().getKey() != null) {
                if (prefs.isConnected()) {
                    prefs.getDb().document("phoenix/orders")  //ref: phoenix/orders
                            .collection(prefs.getCustomer().getKey())
                            .document()
                            .set(hashMap)
                            .addOnFailureListener(e -> Toast.makeText(host, e.getLocalizedMessage(),
                                    Toast.LENGTH_SHORT).show())
                            .addOnCompleteListener(new OnCompleteListener<Void>() {
                                @Override
                                public void onComplete(@NonNull Task<Void> task) {
```

```
266                             if (task.isSuccessful()) {
267                                 Toast.makeText(host,
268                                         String.format("%s has been added to your shopping cart",
269                                                 product.getName()), Toast.LENGTH_SHORT).show();
270                                 holder.add.setText(host.getString(R.string.item_added_to_cart));
271                                 holder.add.setOnClickListener(null);
272                                 holder.add.setEnabled(false);
273                             } else {
274                                 Toast.makeText(host, task.getException().getLocalizedMessage(),
275                                         Toast.LENGTH_SHORT).show();
276                                 holder.add.setText(host.getString(R.string.add_to_cart));
277                                 holder.add.setEnabled(true);
278                             }
279                         }
280                     });
281             } else {
282                 Toast.makeText(host, "Check your internet connection", Toast.LENGTH_SHORT).show();
283             }
284         }
285       }
286     });
287
288     holder.share.setOnClickListener(view -> new ShareProductTask(host,
289         (Product) items.get(holder.getAdapterPosition()))).execute());
290     return holder;
291   }
292 ---------------------------------------------------------------------------------------------
293   @Override
294   public void onBindViewHolder(ViewHolder holder, int position) {
295     switch (getItemViewType(position)) {
296       case TYPE_PRODUCT:
297         bindProductHolder((Product) getItem(position), (ProductViewHolder) holder, position);
298         break;
299       case TYPE_LOADING_MORE:
300         bindLoadingViewHolder((LoadingMoreHolder) holder, position);
301         break;
302     }
303   }
304 ---------------------------------------------------------------------------------------------
305   private void bindLoadingViewHolder(LoadingMoreHolder holder, int position) {
306     // only show the infinite load progress spinner if there are already items in the
307     // grid i.e. it's not the first item & data is being loaded
308     holder.progress.setVisibility((position > 0
309         && dataLoading != null
310         && dataLoading.isDataLoading())
311         ? View.VISIBLE : View.INVISIBLE);
312   }
313 ---------------------------------------------------------------------------------------------
314   @SuppressLint("Range")
315   private void bindProductHolder(Product shot, ProductViewHolder holder, int position) {
316     holder.price.setText(NumberFormat.getCurrencyInstance(Locale.US)
317         .format(Double.parseDouble(shot.getPrice())));
318     holder.title.setText(shot.getName());
319     holder.title.setAlpha(1.0f); // interrupted add to pocket anim can mangle
320     GlideApp.with(host)
321         .asBitmap()
322         .load(shot.getUrl())
323         .listener(new RequestListener<Bitmap>() {
324           @Override
325           public boolean onLoadFailed(@Nullable GlideException e, Object model, Target<Bitmap>
    target, boolean isFirstResource) {
326             return false;
327           }
328
329           @Override
330           public boolean onResourceReady(Bitmap resource, Object model, Target<Bitmap> target,
    DataSource dataSource, boolean isFirstResource) {
```

```
331                     if (!shot.getHasFadedIn()) {
332                         holder.image.setHasTransientState(true);
333                         ObservableColorMatrix cm = new ObservableColorMatrix();
334                         ObjectAnimator saturation = ObjectAnimator.ofFloat(
335                                 cm, ObservableColorMatrix.SATURATION, 0.0f, 1.0f);
336                         saturation.addUpdateListener(new AnimatorUpdateListener() {
337                             @Override
338                             public void onAnimationUpdate(ValueAnimator valueAnimator) {
339                                 // just animating the color matrix does not invalidate the
340                                 // drawable so need this update listener.  Also have to create a
341                                 // new CMCF as the matrix is immutable :(
342                                 holder.image.setColorFilter(new ColorMatrixColorFilter(cm));
343                             }
344                         });
345                         saturation.setDuration(2000L);
346                         saturation.setInterpolator(getFastOutSlowInInterpolator(host));
347                         saturation.addListener(new AnimatorListenerAdapter() {
348                             @Override
349                             public void onAnimationEnd(Animator animation) {
350                                 holder.image.clearColorFilter();
351                                 holder.image.setHasTransientState(false);
352                             }
353                         });
354                         saturation.start();
355                         shot.setHasFadedIn(true);
356                     }
357                     return false;
358                 }
359             })
360             .apply(RequestOptions.placeholderOf(shotLoadingPlaceholders[position %
    shotLoadingPlaceholders.length]))
361             .apply(RequestOptions.diskCacheStrategyOf(DiskCacheStrategy.DATA))
362 //              .apply(RequestOptions.overrideOf(Target.SIZE_ORIGINAL, Target.SIZE_ORIGINAL))
363             .transition(withCrossFade())
364             .into(holder.image);
365         // need both placeholder & background to prevent seeing through shot as it fades in
366         holder.image.setBackground(
367                 shotLoadingPlaceholders[position % shotLoadingPlaceholders.length]);
368         holder.image.setDrawBadge(shot.getAnimated());
369         // need a unique transition name per shot, let's use it's url
370         holder.image.setTransitionName(shot.getUrl());
371         shotPreloadSizeProvider.setView(holder.image);
372
373         PhoenixClient prefs = new PhoenixClient(host);
374         //ref: phoenix/orders/{key}/**
375         if (prefs.isLoggedIn()) {
376             prefs.getDb().document(PhoenixUtils.ORDER_REF)
377                     .collection(prefs.getCustomer().getKey())
378                     .get()
379                     .addOnCompleteListener(host, new OnCompleteListener<QuerySnapshot>() {
380                         @Override
381                         public void onComplete(@NonNull Task<QuerySnapshot> task) {
382                             if (task.isSuccessful()) {
383                                 for (DocumentSnapshot data : task.getResult().getDocuments()) {
384                                     if (data.exists()) {
385                                         Order product = data.toObject(Order.class);
386                                         if (product.getName() != null && Objects.equals(product.getName(), shot.
    getName())) {
387                                             holder.add.setText(host.getString(R.string.item_added_to_cart));
388                                             holder.add.setOnClickListener(null);
389                                             holder.add.setEnabled(false);
390                                         }
391                                     } else {
392                                         if (BuildConfig.DEBUG)
393                                             Timber.d(task.getException().getLocalizedMessage());
394                                     }
395                                 }
```

```
396                         }
397                     }
398                 }).addOnFailureListener(host, e -> Timber.d(e.getLocalizedMessage()));
399         }
400
401     }
402 ---------------------------------------------------------------------------
403     @Override
404     public int getItemViewType(int position) {
405         if (position < getDataItemCount()
406                 && getDataItemCount() > 0) {
407             ProductItem item = getItem(position);
408             if (item instanceof Product) {
409                 return TYPE_PRODUCT;
410             }
411         }
412         return TYPE_LOADING_MORE;
413     }
414 ---------------------------------------------------------------------------
415     @Override
416     public void onViewRecycled(ViewHolder holder) {
417         if (holder instanceof ProductViewHolder) {
418             ProductViewHolder viewHolder = (ProductViewHolder) holder;
419             viewHolder.image.setBadgeColor(initialGifBadgeColor);
420             viewHolder.image.setDrawBadge(false);
421             viewHolder.image.setForeground(ContextCompat.getDrawable(host, R.drawable.mid_grey_ripple)
    );
422         }
423     }
424 ---------------------------------------------------------------------------
425     @Override
426     public long getItemId(int position) {
427         if (getItemViewType(position) == TYPE_LOADING_MORE) {
428             return -1L;
429         }
430         return ((long) getItem(position).hashCode());
431     }
432 ---------------------------------------------------------------------------
433     int getItemPosition(long itemId) {
434         for (int position = 0; position < items.size(); position++) {
435             if (getItem(position).getId() == itemId) return position;
436         }
437         return RecyclerView.NO_POSITION;
438     }
439 ---------------------------------------------------------------------------
440     @Override
441     public int getItemCount() {
442         return getDataItemCount() + (showLoadingMore ? 1 : 0);
443     }
444 ---------------------------------------------------------------------------
445     @NonNull
446     @Override
447     public List<Product> getPreloadItems(int position) {
448         ProductItem item = getItem(position);
449         if (item instanceof Product) {
450             return Collections.singletonList((Product) item);
451         }
452         return Collections.emptyList();
453     }
454 ---------------------------------------------------------------------------
455     @Nullable
456     @Override
457     public RequestBuilder<?> getPreloadRequestBuilder(Product item) {
458         return GlideApp.with(host).load(item.getUrl());
459     }
460 ---------------------------------------------------------------------------
461     @Override
```

```
462     public void dataStartedLoading() {
463         if (showLoadingMore) return;
464         showLoadingMore = true;
465         notifyItemInserted(getLoadingMoreItemPosition());
466     }
467 
468     @Override
469     public void dataFinishedLoading() {
470         if (!showLoadingMore) return;
471         int loadingPos = getLoadingMoreItemPosition();
472         showLoadingMore = false;
473         notifyItemRemoved(loadingPos);
474     }
475 
476     /*Custom*/
477     @Nullable
478     private ProductItem getItem(int position) {
479         if (position < 0 || position >= items.size()) return null;
480         return items.get(position);
481     }
482 
483     int getItemColumnSpan(int position) {
484         switch (getItemViewType(position)) {
485             case TYPE_LOADING_MORE:
486                 return columns;
487             default:
488                 return getItem(position).getColspan();
489         }
490     }
491 
492     public void clear() {
493         items.clear();
494         notifyDataSetChanged();
495     }
496 
497     /**
498      * Main entry point for adding items to this adapter. Takes care of de-duplicating items and
499      * sorting them (depending on the data source). Will also expand some items to span multiple
500      * grid columns.
501      */
502     void addAndResort(List<? extends ProductItem> newItems) {
503         Log.d(TAG, "addAndResort: " + newItems);
504         weighItems(newItems);
505         deduplicateAndAdd(newItems);
506         sort();
507         expandPopularItems();
508         notifyDataSetChanged();
509     }
510 
511     /**
512      * Calculate a 'weight' [0, 1] for each data type for sorting. Each data type/source has a
513      * different metric for weighing it e.g. likes etc. but some sources should keep
514      * the order returned by the API. Weights are 'scoped' to the page they belong to and lower
515      * weights are sorted earlier in the grid (i.e. in ascending weight).
516      */
517     private void weighItems(List<? extends ProductItem> items) {
518         if (items == null || items.isEmpty()) return;
519         ProductItemGroupWeigher weigher = null;
520         if (items.get(0) instanceof Product) {
521             if (shotWeigher == null) shotWeigher = new ProductWeigher();
522             weigher = shotWeigher;
523         }
524         if (weigher != null) {
525             weigher.weigh(items);
526         }
527     }
528 
```

```
529     /**
530      * De-dupe as the same item can be returned by multiple feeds
531      */
532     private void deduplicateAndAdd(List<? extends ProductItem> newItems) {
533         int count = getDataItemCount();
534         for (ProductItem newItem : newItems) {
535             boolean add = true;
536             for (int i = 0; i < count; i++) {
537                 ProductItem existingItem = getItem(i);
538                 if (existingItem != null && existingItem.equals(newItem)) {
539                     add = false;
540                     break;
541                 }
542             }
543             if (add) {
544                 add(newItem);
545             }
546         }
547     }
548 
549     private void add(ProductItem item) {
550         items.add(item);
551     }
552 
553     private void sort() {
554         Collections.sort(items, comparator); // sort by weight
555     }
556 
557     private void expandPopularItems() {
558         // for now just expand the first image per page which should be
559         // the most popular according to our weighing & sorting
560         List<Integer> expandedPositions = new ArrayList<>(0);
561         int page = -1;
562         int count = items.size();
563         for (int i = 0; i < count; i++) {
564             ProductItem item = getItem(i);
565             if (item instanceof Product && item.getPage() > page) {
566                 item.setColspan(columns);
567                 page = item.getPage();
568                 expandedPositions.add(i);
569             } else {
570                 if (item != null) {
571                     item.setColspan(1);
572                 }
573             }
574         }
575 
576         // make sure that any expanded items are at the start of a row
577         // so that we don't leave any gaps in the grid
578         for (int expandedPos = 0; expandedPos < expandedPositions.size(); expandedPos++) {
579             int pos = expandedPositions.get(expandedPos);
580             int extraSpannedSpaces = expandedPos * (columns - 1);
581             int rowPosition = (pos + extraSpannedSpaces) % columns;
582             if (rowPosition != 0) {
583                 int swapWith = pos + (columns - rowPosition);
584                 if (swapWith < items.size()) {
585                     Collections.swap(items, pos, swapWith);
586                 }
587             }
588         }
589     }
590 
591     void removeDataSource(String dataSource) {
592         for (int i = items.size() - 1; i >= 0; i--) {
593             ProductItem item = items.get(i);
594             if (dataSource.equals(item.getDataSource())) {
595                 items.remove(i);
```

```
596             }
597         }
598         sort();
599         expandPopularItems();
600         notifyDataSetChanged();
601     }
602 
603     /**
604      * The shared element transition to dribbble shots & dn stories can intersect with the FAB.
605      * This can cause a strange layers-passing-through-each-other effect. On return hide the FAB
606      * and animate it back in after the transition.
607      */
608     private void setGridItemContentTransitions(View gridItem) {
609         View fab = host.findViewById(R.id.fab);
610         if (!ViewUtils.viewsIntersect(gridItem, fab)) return;
611 
612         Transition reenter = TransitionInflater.from(host)
613                 .inflateTransition(R.transition.grid_overlap_fab_reenter);
614         reenter.addListener(new TransitionUtils.TransitionListenerAdapter() {
615 
616             @Override
617             public void onTransitionEnd(Transition transition) {
618                 // we only want these content transitions in certain cases so clear out when done.
619                 host.getWindow().setReenterTransition(null);
620             }
621         });
622         host.getWindow().setReenterTransition(reenter);
623     }
624 
625     int getDataItemCount() {
626         return items.size();
627     }
628 
629     private int getLoadingMoreItemPosition() {
630         return showLoadingMore ? getItemCount() - 1 : RecyclerView.NO_POSITION;
631     }
632 
633     @NonNull
634     static SharedElementCallback createSharedElementReenterCallback(
635             @NonNull Context context) {
636         String shotTransitionName = context.getString(R.string.transition_shot);
637         String shotBackgroundTransitionName =
638                 context.getString(R.string.transition_shot_background);
639         return new SharedElementCallback() {
640 
641             /**
642              * We're performing a slightly unusual shared element transition i.e. from one view
643              * (image in the grid) to two views (the image & also the background of the details
644              * view, to produce the expand effect). After changing orientation, the transition
645              * system seems unable to map both shared elements (only seems to map the shot, not
646              * the background) so in this situation we manually map the background to the
647              * same view.
648              */
649             @Override
650             public void onMapSharedElements(List<String> names, Map<String, View> sharedElements) {
651                 if (sharedElements.size() != names.size()) {
652                     // couldn't map all shared elements
653                     View sharedShot = sharedElements.get(shotTransitionName);
654                     if (sharedShot != null) {
655                         // has shot so add shot background, mapped to same view
656                         sharedElements.put(shotBackgroundTransitionName, sharedShot);
657                     }
658                 }
659             }
660         };
661     }
662 
```

```
663     /*package*/static class ProductViewHolder extends RecyclerView.ViewHolder {
664         BadgedFourThreeImageView image;
665         TextView title;
666         TextView price;
667         Button add;
668         Button share;
669 
670         ProductViewHolder(View itemView) {
671             super(itemView);
672             image = itemView.findViewById(R.id.shot);
673             title = itemView.findViewById(R.id.title_product);
674             price = itemView.findViewById(R.id.price_product);
675             add = itemView.findViewById(R.id.add_product_to_cart);
676             share = itemView.findViewById(R.id.share_product);
677         }
678     }
679 
680     /*package*/static class LoadingMoreHolder extends RecyclerView.ViewHolder {
681         ProgressBar progress;
682 
683         LoadingMoreHolder(View itemView) {
684             super(itemView);
685             progress = (ProgressBar) itemView;
686         }
687     }
688 }
689 
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.animation.Animator
8  import android.animation.AnimatorListenerAdapter
9  import android.animation.ObjectAnimator
10 import android.app.Activity
11 import android.content.Context
12 import android.support.annotation.NonNull
13 import android.support.v4.content.ContextCompat
14 import android.support.v7.widget.DefaultItemAnimator
15 import android.support.v7.widget.RecyclerView
16 import android.view.LayoutInflater
17 import android.view.View
18 import android.view.ViewGroup
19 import android.view.animation.LinearInterpolator
20 import android.widget.ImageView
21 import android.widget.TextView
22 import io.pergasus.R
23 import io.pergasus.api.PhoenixClient
24 import io.pergasus.api.Source
25 import io.pergasus.api.Source.SourceComparator
26 import io.pergasus.api.SourceManager
27 import io.pergasus.ui.recyclerview.FilterSwipeDismissListener
28 import io.pergasus.util.AnimUtils
29 import io.pergasus.util.ColorUtils
30 import io.pergasus.util.ViewUtils
31 import java.util.*
32
33 class FilterAdapter(private val host: Activity, val filters: MutableList<Source>,
34                     private val authoriser: FilterAuthoriser) : RecyclerView.Adapter<FilterAdapter.FilterViewHolder>
    (),
35         FilterSwipeDismissListener, PhoenixClient.AppLoginStatusListener {
36     private val context: Context = host.applicationContext
37     private var callbacks: MutableList<FiltersChangedCallbacks>? = null
38 ____________________________________________________________________________________________________
39     init {
40         setHasStableIds(true)
41     }
42 ____________________________________________________________________________________________________
43     val enabledSourcesCount: Int
44         get() {
45             return filters.count { it.active }
46         }
47
48     interface FilterAuthoriser {
49         fun requestAuthorisation(sharedElement: View, forSource: Source)
50     }
51
52     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): FilterViewHolder {
53         val holder = FilterViewHolder(LayoutInflater.from(parent
54                 .context).inflate(R.layout.filter_item, parent, false))
55         holder.itemView.setOnClickListener(View.OnClickListener {
56             val position = holder.adapterPosition
57             if (position == RecyclerView.NO_POSITION) return@OnClickListener
58             val filter = filters[position]
59             if (isAuthorisedPhoenixSource(filter) && !PhoenixClient(holder.itemView.context).isLoggedIn) {
60                 authoriser.requestAuthorisation(holder.filterIcon, filter)
61             } else {
62                 filter.active = !filter.active
63                 holder.filterName.isEnabled = filter.active
64                 notifyItemChanged(position, if (filter.active)
65                     FILTER_ENABLED
```

```
66              else
67                  FILTER_DISABLED)
68              SourceManager.updateSource(filter, holder.itemView.context)
69              dispatchFiltersChanged(filter)
70          }
71      })
72      return holder
73  }
74
75  override fun onBindViewHolder(holder: FilterViewHolder, position: Int) {
76      val filter = filters[position]
77      if (holder != null) {
78          holder.isSwipeable = filter.isSwipeDismissable
79          holder.filterName.text = filter.name
80          holder.filterName.isEnabled = filter.active
81          if (filter.iconRes > 0) {
82              holder.filterIcon.setImageDrawable(
83                      holder.itemView.context.getDrawable(filter.iconRes))
84          }
85          holder.filterIcon.imageAlpha = if (filter.active)
86              FILTER_ICON_ENABLED_ALPHA
87          else
88              FILTER_ICON_DISABLED_ALPHA
89      }
90  }
91
92  override fun onBindViewHolder(holder: FilterViewHolder,
93                                position: Int,
94                                payloads: MutableList<Any>) {
95      if (payloads.isEmpty()) {
96          onBindViewHolder(holder, position)
97      } else {
98          // if we're doing a partial re-bind i.e. an item is enabling/disabling or being
99          // highlighted then data hasn't changed. Just set state based on the payload
100         val filterEnabled = payloads.contains(FILTER_ENABLED)
101         val filterDisabled = payloads.contains(FILTER_DISABLED)
102         if (filterEnabled || filterDisabled) {
103             holder.filterName.isEnabled = filterEnabled
104         }
105     }
106 }
107
108 override fun getItemCount(): Int {
109     return filters.size
110 }
111
112 override fun getItemId(position: Int): Long {
113     return filters[position].key.hashCode().toLong()
114 }
115
116 override fun onItemDismiss(position: Int) {
117     val removing = filters[position]
118     if (removing.isSwipeDismissable) {
119         removeFilter(removing)
120     }
121 }
122
123 override fun onUserLogin() {
124 }
125
126 override fun onUserLogout() {
127     for (i in filters.indices) {
128         val filter = filters[i]
129         if (filter.active && isAuthorisedPhoenixSource(filter)) {
130             filter.active = false
131             SourceManager.updateSource(filter, context)
132             dispatchFiltersChanged(filter)
```

```
133             notifyItemChanged(i, FILTER_DISABLED)
134         }
135     }
136 }
137 
138 /**
139  * Adds a new data source to the list of filters. If the source already exists then it is simply
140  * activated.
141  *
142  * @param toAdd the source to add
143  * @return whether the filter was added (i.e. if it did not already exist)
144  */
145 fun addFilter(toAdd: Source): Boolean {
146     // first check if it already exists
147     val count = filters.size
148     for (i in 0 until count) {
149         val existing = filters[i]
150         if (existing == toAdd && existing.key.equals(toAdd.key, ignoreCase = true)) {
151             // already exists, just ensure it's active
152             if (!existing.active) {
153                 existing.active = true
154                 dispatchFiltersChanged(existing)
155                 notifyItemChanged(i, FILTER_ENABLED)
156                 SourceManager.updateSource(existing, context)
157             }
158             return false
159         }
160     }
161     // didn't already exist, so add it
162     filters.add(toAdd)
163     Collections.sort(filters, SourceComparator())
164     dispatchFiltersChanged(toAdd)
165     notifyDataSetChanged()
166     SourceManager.addSource(toAdd, context)
167     return true
168 }
169 
170 private fun removeFilter(removing: Source) {
171     val position = filters.indexOf(removing)
172     filters.removeAt(position)
173     notifyItemRemoved(position)
174     dispatchFilterRemoved(removing)
175     SourceManager.removeSource(removing, context)
176 }
177 
178 fun getFilterPosition(filter: Source): Int {
179     return filters.indexOf(filter)
180 }
181 
182 fun enableFilterByKey(@NonNull key: String, @NonNull context: Context) {
183     val count = filters.size
184     for (i in 0 until count) {
185         val filter = filters[i]
186         if (filter.key == key) {
187             if (!filter.active) {
188                 filter.active = true
189                 notifyItemChanged(i, FILTER_ENABLED)
190                 dispatchFiltersChanged(filter)
191                 SourceManager.updateSource(filter, context)
192             }
193             return
194         }
195     }
196 }
197 
198 fun highlightFilter(adapterPosition: Int) {
199     notifyItemChanged(adapterPosition, HIGHLIGHT)
```

```
200     }
201
202     fun registerFilterChangedCallback(callback: FiltersChangedCallbacks) {
203         if (callbacks == null) {
204             callbacks = ArrayList(0)
205         }
206         callbacks!!.add(callback)
207     }
208
209     fun unregisterFilterChangedCallback(callback: FiltersChangedCallbacks) {
210         if (callbacks != null && !callbacks!!.isEmpty()) {
211             callbacks!!.remove(callback)
212         }
213     }
214
215     private fun isAuthorisedPhoenixSource(source: Source): Boolean {
216         return (source.key.equals(SourceManager.SOURCE_FAVORITE, ignoreCase = true)
217                 || source.key.equals(SourceManager.SOURCE_HEALTH, ignoreCase = true)
218                 || source.key.equals(SourceManager.SOURCE_BUSINESS, ignoreCase = true)
219                 || source.key.equals(SourceManager.SOURCE_STUDENT, ignoreCase = true))
220     }
221
222     private fun dispatchFiltersChanged(filter: Source) {
223         if (callbacks != null && !callbacks!!.isEmpty()) {
224             for (callback in callbacks!!) {
225                 callback.onFiltersChanged(filter)
226             }
227         }
228     }
229
230     private fun dispatchFilterRemoved(filter: Source) {
231         if (callbacks != null && !callbacks!!.isEmpty()) {
232             for (callback in callbacks!!) {
233                 callback.onFilterRemoved(filter)
234             }
235         }
236     }
237
238     abstract class FiltersChangedCallbacks {
239         open fun onFiltersChanged(changedFilter: Source) {}
240         open fun onFilterRemoved(removed: Source) {}
241     }
242
243     class FilterViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
244
245         var filterName: TextView = itemView.findViewById(R.id.filter_name)
246         var filterIcon: ImageView = itemView.findViewById(R.id.filter_icon)
247         var isSwipeable: Boolean = false
248
249     }
250
251     class FilterAnimator : DefaultItemAnimator() {
252
253         override fun canReuseUpdatedViewHolder(viewHolder: RecyclerView.ViewHolder): Boolean {
254             return true
255         }
256
257         override fun obtainHolderInfo(): RecyclerView.ItemAnimator.ItemHolderInfo {
258             return FilterHolderInfo()
259         }
260
261         /* package */ internal class FilterHolderInfo : RecyclerView.ItemAnimator.ItemHolderInfo() {
262             var doEnable: Boolean = false
263             var doDisable: Boolean = false
264             var doHighlight: Boolean = false
265         }
266
```

```
267     override fun recordPreLayoutInformation(state: RecyclerView.State,
268                                             viewHolder: RecyclerView.ViewHolder,
269                                             changeFlags: Int,
270                                             payloads: List<Any>): RecyclerView.ItemAnimator.ItemHolderInfo {
271         val info = super.recordPreLayoutInformation(state, viewHolder, changeFlags, payloads) as
        FilterHolderInfo
272         if (!payloads.isEmpty()) {
273             info.doEnable = payloads.contains(FILTER_ENABLED)
274             info.doDisable = payloads.contains(FILTER_DISABLED)
275             info.doHighlight = payloads.contains(HIGHLIGHT)
276         }
277         return info
278     }
279 ---------------------------------------------------------------------------------------------------------
280     override fun animateChange(oldHolder: RecyclerView.ViewHolder,
281                                newHolder: RecyclerView.ViewHolder,
282                                preInfo: RecyclerView.ItemAnimator.ItemHolderInfo,
283                                postInfo: RecyclerView.ItemAnimator.ItemHolderInfo): Boolean {
284         if (newHolder is FilterViewHolder && preInfo is FilterHolderInfo) {
285
286             if (preInfo.doEnable || preInfo.doDisable) {
287                 val iconAlpha = ObjectAnimator.ofInt(newHolder.filterIcon,
288                         ViewUtils.IMAGE_ALPHA,
289                         if (preInfo.doEnable)
290                             FILTER_ICON_ENABLED_ALPHA
291                         else
292                             FILTER_ICON_DISABLED_ALPHA)
293                 iconAlpha.duration = 300L
294                 iconAlpha.interpolator = AnimUtils.getFastOutSlowInInterpolator(newHolder
295                         .itemView.context)
296                 iconAlpha.addListener(object : AnimatorListenerAdapter() {
297                     override fun onAnimationStart(animation: Animator) {
298                         dispatchChangeStarting(newHolder, false)
299                         newHolder.itemView.setHasTransientState(true)
300                     }
301 ---------------------------------------------------------------------------------------------------------
302                     override fun onAnimationEnd(animation: Animator) {
303                         newHolder.itemView.setHasTransientState(false)
304                         dispatchChangeFinished(newHolder, false)
305                     }
306                 })
307                 iconAlpha.start()
308             } else if (preInfo.doHighlight) {
309                 val highlightColor = ContextCompat.getColor(newHolder.itemView.context, R.color.accent)
310                 val fadeFromTo = ColorUtils.modifyAlpha(highlightColor, 0)
311                 val highlightBackground = ObjectAnimator.ofArgb(
312                         newHolder.itemView,
313                         ViewUtils.BACKGROUND_COLOR,
314                         fadeFromTo,
315                         highlightColor,
316                         fadeFromTo)
317                 highlightBackground.duration = 1000L
318                 highlightBackground.interpolator = LinearInterpolator()
319                 highlightBackground.addListener(object : AnimatorListenerAdapter() {
320                     override fun onAnimationStart(animation: Animator) {
321                         dispatchChangeStarting(newHolder, false)
322                         newHolder.itemView.setHasTransientState(true)
323                     }
324 ---------------------------------------------------------------------------------------------------------
325                     override fun onAnimationEnd(animation: Animator) {
326                         newHolder.itemView.background = null
327                         newHolder.itemView.setHasTransientState(false)
328                         dispatchChangeFinished(newHolder, false)
329                     }
330                 })
331                 highlightBackground.start()
332             }
```

```
333             }
334             return super.animateChange(oldHolder, newHolder, preInfo, postInfo)
335         }
336
337     }
338
339     companion object {
340         //Filters
341         private const val FILTER_ENABLED = 2
342         private const val FILTER_DISABLED = 3
343         private const val HIGHLIGHT = 4
344         private const val FILTER_ICON_ENABLED_ALPHA = 179 // 70%
345         private const val FILTER_ICON_DISABLED_ALPHA = 51 // 20%
346     }
347
348
349 }
350
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui;
6
7  import android.animation.Animator;
8  import android.animation.AnimatorSet;
9  import android.animation.ArgbEvaluator;
10 import android.animation.ObjectAnimator;
11 import android.animation.ValueAnimator;
12 import android.graphics.Color;
13 import android.view.animation.AccelerateDecelerateInterpolator;
14 import android.view.animation.AccelerateInterpolator;
15 import android.view.animation.DecelerateInterpolator;
16
17 import com.google.android.gms.maps.GoogleMap;
18 import com.google.android.gms.maps.model.LatLng;
19 import com.google.android.gms.maps.model.Polyline;
20 import com.google.android.gms.maps.model.PolylineOptions;
21
22 import java.util.List;
23
24 public class MapAnimator {
25
26     private static MapAnimator mapAnimator;
27
28     private Polyline backgroundPolyline;
29
30     private Polyline foregroundPolyline;
31
32     private PolylineOptions optionsForeground;
33
34     private AnimatorSet firstRunAnimSet;
35
36     private AnimatorSet secondLoopRunAnimSet;
37
38     static final int GREY = Color.parseColor("#FFA7A6A6");
39
40 ____________________________________________________________________________________________
41     private MapAnimator() {
42
43     }
44 ____________________________________________________________________________________________
45     public static MapAnimator getInstance() {
46         if (mapAnimator == null) mapAnimator = new MapAnimator();
47         return mapAnimator;
48     }
49
50 ____________________________________________________________________________________________
51     public void animateRoute(GoogleMap googleMap, List<LatLng> route) {
52         if (firstRunAnimSet == null) {
53             firstRunAnimSet = new AnimatorSet();
54         } else {
55             firstRunAnimSet.removeAllListeners();
56             firstRunAnimSet.end();
57             firstRunAnimSet.cancel();
58
59             firstRunAnimSet = new AnimatorSet();
60         }
61         if (secondLoopRunAnimSet == null) {
62             secondLoopRunAnimSet = new AnimatorSet();
63         } else {
64             secondLoopRunAnimSet.removeAllListeners();
65             secondLoopRunAnimSet.end();
66             secondLoopRunAnimSet.cancel();
```

```
67
68          secondLoopRunAnimSet = new AnimatorSet();
69      }
70      //Reset the polylines
71      if (foregroundPolyline != null) foregroundPolyline.remove();
72      if (backgroundPolyline != null) backgroundPolyline.remove();
73
74
75      PolylineOptions optionsBackground = new PolylineOptions().add(route.get(0)).color(GREY).width(5);
76      backgroundPolyline = googleMap.addPolyline(optionsBackground);
77
78      optionsForeground = new PolylineOptions().add(route.get(0)).color(Color.BLACK).width(5);
79      foregroundPolyline = googleMap.addPolyline(optionsForeground);
80
81      ValueAnimator percentageCompletion = ValueAnimator.ofInt(0, 100);
82      percentageCompletion.setDuration(2000);
83      percentageCompletion.setInterpolator(new DecelerateInterpolator());
84      percentageCompletion.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
85          @Override
86          public void onAnimationUpdate(ValueAnimator animation) {
87              List<LatLng> foregroundPoints = backgroundPolyline.getPoints();
88
89              int percentageValue = (int) animation.getAnimatedValue();
90              int pointcount = foregroundPoints.size();
91              int countTobeRemoved = (int) (pointcount * (percentageValue / 100.0f));
92              List<LatLng> subListTobeRemoved = foregroundPoints.subList(0, countTobeRemoved);
93              subListTobeRemoved.clear();
94
95              foregroundPolyline.setPoints(foregroundPoints);
96          }
97      });
98      percentageCompletion.addListener(new Animator.AnimatorListener() {
99          @Override
100         public void onAnimationStart(Animator animation) {
101
102         }
103
104         @Override
105         public void onAnimationEnd(Animator animation) {
106             foregroundPolyline.setColor(GREY);
107             foregroundPolyline.setPoints(backgroundPolyline.getPoints());
108         }
109
110         @Override
111         public void onAnimationCancel(Animator animation) {
112
113         }
114
115         @Override
116         public void onAnimationRepeat(Animator animation) {
117
118         }
119     });
120
121
122     ValueAnimator colorAnimation = ValueAnimator.ofObject(new ArgbEvaluator(), GREY, Color.BLACK);
123     colorAnimation.setInterpolator(new AccelerateInterpolator());
124     colorAnimation.setDuration(1200); // milliseconds
125
126     colorAnimation.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
127         @Override
128         public void onAnimationUpdate(ValueAnimator animator) {
129             foregroundPolyline.setColor((int) animator.getAnimatedValue());
130         }
131
132     });
133
```

```
134         ObjectAnimator foregroundRouteAnimator = ObjectAnimator.ofObject(this, "routeIncreaseForward",
        new RouteEvaluator(), route.toArray());
135         foregroundRouteAnimator.setInterpolator(new AccelerateDecelerateInterpolator());
136         foregroundRouteAnimator.addListener(new Animator.AnimatorListener() {
137           @Override
138           public void onAnimationStart(Animator animation) {
139
140           }
141
142           @Override
143           public void onAnimationEnd(Animator animation) {
144             backgroundPolyline.setPoints(foregroundPolyline.getPoints());
145           }
146
147           @Override
148           public void onAnimationCancel(Animator animation) {
149
150           }
151
152           @Override
153           public void onAnimationRepeat(Animator animation) {
154
155           }
156         });
157         foregroundRouteAnimator.setDuration(1600);
158 //        foregroundRouteAnimator.start();
159
160         firstRunAnimSet.playSequentially(foregroundRouteAnimator,
161             percentageCompletion);
162         firstRunAnimSet.addListener(new Animator.AnimatorListener() {
163           @Override
164           public void onAnimationStart(Animator animation) {
165
166           }
167
168           @Override
169           public void onAnimationEnd(Animator animation) {
170             secondLoopRunAnimSet.start();
171           }
172
173           @Override
174           public void onAnimationCancel(Animator animation) {
175
176           }
177
178           @Override
179           public void onAnimationRepeat(Animator animation) {
180
181           }
182         });
183
184         secondLoopRunAnimSet.playSequentially(colorAnimation,
185             percentageCompletion);
186         secondLoopRunAnimSet.setStartDelay(200);
187
188         secondLoopRunAnimSet.addListener(new Animator.AnimatorListener() {
189           @Override
190           public void onAnimationStart(Animator animation) {
191
192           }
193
194           @Override
195           public void onAnimationEnd(Animator animation) {
196             secondLoopRunAnimSet.start();
197           }
198
199           @Override
```

```
200             public void onAnimationCancel(Animator animation) {
201
202             }
203
204             @Override
205             public void onAnimationRepeat(Animator animation) {
206
207             }
208         });
209
210         firstRunAnimSet.start();
211     }
212
213     /**
214      * This will be invoked by the ObjectAnimator multiple times. Mostly every 16ms.
215      **/
216     public void setRouteIncreaseForward(LatLng endLatLng) {
217         List<LatLng> foregroundPoints = foregroundPolyline.getPoints();
218         foregroundPoints.add(endLatLng);
219         foregroundPolyline.setPoints(foregroundPoints);
220     }
221
222 }
223
224
225
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.animation.Animator
8  import android.animation.AnimatorListenerAdapter
9  import android.annotation.SuppressLint
10 import android.app.Activity
11 import android.app.ActivityOptions
12 import android.content.Intent
13 import android.net.Uri
14 import android.os.Bundle
15 import android.support.v4.app.ActivityCompat
16 import android.support.v4.app.ShareCompat
17 import android.support.v4.content.ContextCompat
18 import android.text.TextUtils
19 import android.transition.TransitionManager
20 import android.view.View
21 import android.view.ViewGroup
22 import android.view.ViewTreeObserver
23 import android.widget.Button
24 import android.widget.ProgressBar
25 import android.widget.TextView
26 import android.widget.Toast
27 import com.afollestad.materialdialogs.MaterialDialog
28 import com.afollestad.materialdialogs.Theme
29 import com.google.android.gms.wallet.AutoResolveHelper
30 import com.google.android.gms.wallet.PaymentData
31 import com.hubtel.payments.Class.Environment
32 import com.hubtel.payments.Exception.HubtelPaymentException
33 import com.hubtel.payments.HubtelCheckout
34 import com.hubtel.payments.Interfaces.OnPaymentResponse
35 import com.hubtel.payments.SessionConfiguration
36 import com.paypal.android.sdk.payments.PayPalConfiguration
37 import com.paypal.android.sdk.payments.PayPalPayment
38 import com.paypal.android.sdk.payments.PayPalService
39 import com.paypal.android.sdk.payments.PaymentActivity
40 import io.pergasus.BuildConfig
41 import io.pergasus.R
42 import io.pergasus.api.PhoenixClient
43 import io.pergasus.api.PhoenixUtils
44 import io.pergasus.ui.transitions.FabTransform
45 import io.pergasus.ui.transitions.MorphTransform
46 import io.pergasus.ui.widget.BottomSheet
47 import io.pergasus.ui.widget.ObservableScrollView
48 import io.pergasus.util.AnimUtils
49 import io.pergasus.util.ImeUtils
50 import io.pergasus.util.bindView
51 import timber.log.Timber
52 import java.math.BigDecimal
53 import java.text.NumberFormat
54 import java.util.*
55
56 /**
57  * User orders activity
58  */
59 @SuppressLint("LogConditional")
60 class OrderActivity : Activity() {
61
62     private val bottomSheet: BottomSheet by bindView(R.id.bottom_sheet)
63     private val bottomSheetContent: ViewGroup by bindView(R.id.bottom_sheet_content)
64     private val scrollContainer: ObservableScrollView by bindView(R.id.scroll_container)
65     private val sheetTitle: TextView by bindView(R.id.title)
66     private val checkOut: Button by bindView(R.id.checkout)
```

```
67    private val loading: ProgressBar by bindView(R.id.loading)
68
69    //Transaction Details
70    private val orderTotal: TextView by bindView(R.id.order_total)
71    private val orderTax: TextView by bindView(R.id.order_tax_cost)
72    private val orderDelivery: TextView by bindView(R.id.order_delivery_cost)
73    private val orderSavings: TextView by bindView(R.id.order_savings)
74    private val orderMethod: TextView by bindView(R.id.order_payment_method)
75    private val orderLocation: TextView by bindView(R.id.order_customer_location)
76    private val orderName: TextView by bindView(R.id.order_customer_name)
77    private val updateLocation: Button by bindView(R.id.update_location)
78    private val updateMethod: Button by bindView(R.id.update_provider)
79
80
81    private var appBarElevation: Float = 0.0f
82    private lateinit var client: PhoenixClient
83    private lateinit var walletPaymentSetup: WalletPaymentSetup
84
85    override fun onCreate(savedInstanceState: Bundle?) {
86        super.onCreate(savedInstanceState)
87        setContentView(R.layout.activity_order)
88
89        if (!FabTransform.setup(this, bottomSheetContent)) {
90            MorphTransform.setup(this, bottomSheetContent,
91                    ContextCompat.getColor(this, R.color.background_light), 0)
92        }
93
94        client = PhoenixClient(this@OrderActivity)
95        appBarElevation = resources.getDimension(R.dimen.z_app_bar)
96
97        //Setup Wallet
98        walletPaymentSetup = WalletPaymentSetup(this@OrderActivity, checkOut)
99
100       //Setup PayPal
101       setupPayPal()
102
103       bottomSheet.registerCallback(object : BottomSheet.Callbacks() {
104           override fun onSheetDismissed() {
105               // After a drag dismiss, finish without the shared element return transition as
106               // it no longer makes sense.  Let the launching window know it's a drag dismiss so
107               // that it can restore any UI used as an entering shared element
108               setResult(RESULT_DRAG_DISMISSED)
109               finish()
110           }
111       })
112
113       scrollContainer.setListener { scrollY ->
114           if (scrollY != 0
115                   && sheetTitle.translationZ != appBarElevation) {
116               sheetTitle.animate()
117                       .translationZ(appBarElevation)
118                       .setStartDelay(0L)
119                       .setDuration(80L)
120                       .setInterpolator(AnimUtils.getFastOutSlowInInterpolator
121                       (this@OrderActivity))
122                       .start()
123           } else if (scrollY == 0 && sheetTitle.translationZ == appBarElevation) {
124               sheetTitle.animate()
125                       .translationZ(0f)
126                       .setStartDelay(0L)
127                       .setDuration(80L)
128                       .setInterpolator(AnimUtils.getFastOutSlowInInterpolator
129                       (this@OrderActivity))
130                       .start()
131           }
132       }
133
```

```
134         if (isShareIntent()) {
135             val intentReader = ShareCompat.IntentReader.from(this)
136             sheetTitle.text = intentReader.text
137         }
138
139         if (!hasSharedElementTransition()) {
140             // when launched from share or app shortcut there is no shared element transition so
141             // animate up the bottom sheet to establish the spatial model i.e. that it can be
142             // dismissed downward
143             overridePendingTransition(R.anim.post_story_enter, R.anim.post_story_exit)
144             bottomSheetContent.viewTreeObserver.addOnPreDrawListener(
145                     object : ViewTreeObserver.OnPreDrawListener {
146                         override fun onPreDraw(): Boolean {
147                             bottomSheetContent.viewTreeObserver.removeOnPreDrawListener(this)
148                             bottomSheetContent.translationY = bottomSheetContent.height.toFloat()
149                             bottomSheetContent.animate()
150                                     .translationY(0f)
151                                     .setStartDelay(120L)
152                                     .setDuration(240L).interpolator = AnimUtils
153                                     .getLinearOutSlowInInterpolator(this@OrderActivity)
154                             return false
155                         }
156                     })
157         }
158
159         val intent = intent
160         if (intent.hasExtra(EXTRA_CART_TITLE) && intent.hasExtra(EXTRA_CART_PRICE)) {
161             val price = intent.getDoubleExtra(EXTRA_CART_PRICE, 0.0)
162             val title = intent.getStringExtra(EXTRA_CART_TITLE)
163             //val quantity = intent.getDoubleExtra(EXTRA_CART_QUANTITY, 0.00)
164             bindDetails(price, title)
165         }
166
167         //Dismiss sheet when user clicks outside it
168         bottomSheet.setOnClickListener { dismiss() }
169     }
170 ---------------------------------------------------------------------------------------------
171     private fun setupPayPal() {
172         val intent = Intent(this, PayPalService::class.java)
173         intent.putExtra(PayPalService.EXTRA_PAYPAL_CONFIGURATION, config)
174         startService(intent)
175     }
176 ---------------------------------------------------------------------------------------------
177     //Setup details from intent data
178     private fun bindDetails(price: Double?, title: String?) {
179         if (price == null || title == null) return
180         if (BuildConfig.DEBUG) Timber.d("Price is: $price and title is: $title")
181         checkOut.isEnabled = true
182         checkOut.setOnClickListener {
183             if (client.isLoggedIn) {
184                 ImeUtils.hideIme(sheetTitle)
185                 when (orderMethod.text) {
186                     getString(R.string.slydepay) -> {
187                         doMobileMoneyPayment(price, title)
188                     }
189                     getString(R.string.pay_with_android_pay) -> {
190                         doAndroidPay(price)
191                     }
192                     getString(R.string.visa) -> {
193                         doPayPalPayment(price, title)
194                     }
195                 }
196             } else {
197                 val login = Intent(this, AuthActivity::class.java)
198                 val options = ActivityOptions.makeSceneTransitionAnimation(this, checkOut,
199                         getString(R.string.transition_dribbble_login))
200                 ActivityCompat.startActivityForResult(this@OrderActivity, login, RC_AUTH_PAYMENT,
```

```
                options.toBundle())
        }
    }

    //Add details
    orderDelivery.text = setValue(getRandDelivery())
    orderSavings.text = setValue(getRandSavings())
    orderTax.text = setValue(getRandTax())
    loadUser()

    //get amount from intent value
    var amt: Double = price

    //Update amount with additional costs
    amt += valueOf(orderDelivery)
    amt -= valueOf(orderSavings)
    amt += valueOf(orderTax)

    //Set value for the orderTotal
    TransitionManager.beginDelayedTransition(bottomSheetContent)
    orderTotal.text = setValue(amt)

    //Add actions to buttons
    updateLocation.setOnClickListener {
        startActivityForResult(Intent(this@OrderActivity, ProfileActivity::class.java)
            , RESULT_UPDATE_PROFILE)
    }

    //Array of options
    val array = arrayOf(
        getString(R.string.slydepay),
        getString(R.string.pay_with_android_pay),
        getString(R.string.visa)
    )
    updateMethod.setOnClickListener {
        MaterialDialog.Builder(this@OrderActivity)
            .theme(Theme.LIGHT)
            .title(getString(R.string.confirm_trans_hint))
            .positiveText("Ok")
            .negativeText("Cancel")
            .onNegative({ dialog, _ ->
                dialog.dismiss()
            })
            .onPositive { dialog, which ->
                when (which.ordinal) {
                    0, 1, 2 -> {
                        textForMethod(which.name)
                        dialog.dismiss()
                    }
                }
            }
            .items(*array)
            .itemsCallbackSingleChoice(0, { dialog, _, which, text ->
                when (which) {
                    0, 1, 2 -> {
                        textForMethod(text.toString())
                        dialog.dismiss()
                    }
                }
                return@itemsCallbackSingleChoice true
            })
            .build().show()
    }

}

//Generates random value for delivery cost
```

```
268     private fun getRandDelivery(): Double {
269         val random = Random(5)
270         return (1.0).plus(random.nextDouble())
271     }
272 
273     //Generates random value for purchase savings
274     private fun getRandSavings(): Double {
275         val random = Random(20)
276         return (3.0).plus(random.nextDouble())
277     }
278 
279     //Generates random value for tax charged
280     private fun getRandTax(): Double {
281         val random = Random(10)
282         return random.nextDouble()
283     }
284 
285     //Loads current user's data
286     private fun loadUser() {
287         if (client.isLoggedIn) {
288             TransitionManager.beginDelayedTransition(bottomSheetContent)
289             orderName.text = client.customer.name
290             orderLocation.text = client.getPlace()
291         }
292     }
293 
294     //Pay with [PayPal]
295     private fun doPayPalPayment(price: Double, title: String?) {
296         TransitionManager.beginDelayedTransition(bottomSheetContent)
297         checkOut.isEnabled = false
298         performPayPalPayment(price, title)
299     }
300 
301     private fun performPayPalPayment(price: Double, title: String?) {
302         val intent = Intent(this@OrderActivity, PaymentActivity::class.java)
303         // send the same configuration for restart resiliency
304         intent.putExtra(PayPalService.EXTRA_PAYPAL_CONFIGURATION, config)
305         val thingToBuy = PayPalPayment(BigDecimal(price.toString()), PhoenixUtils.DEF_CURRENCY,
306                 title, PayPalPayment.PAYMENT_INTENT_SALE)
307         intent.putExtra(PaymentActivity.EXTRA_PAYMENT, thingToBuy)
308         startActivityForResult(intent, REQUEST_CODE_PAYMENT)
309     }
310 
311     //Pay with [AndroidPay]
312     private fun doAndroidPay(price: Double) {
313         TransitionManager.beginDelayedTransition(bottomSheetContent)
314         checkOut.isEnabled = false
315         performAndroidPay(price)
316     }
317 
318     private fun performAndroidPay(price: Double) {
319         val paymentClient = walletPaymentSetup.getPaymentClient()
320         val request = walletPaymentSetup.createPaymentDataRequest(price.toString())
321         if (request != null) {
322             AutoResolveHelper.resolveTask(paymentClient.loadPaymentData(request),
323                     this@OrderActivity, LOAD_PAYMENT_DATA_REQUEST_CODE)
324         }
325     }
326 
327     //Pay with [Slydepay]
328     private fun doMobileMoneyPayment(price: Double, title: String?) {
329         TransitionManager.beginDelayedTransition(bottomSheetContent)
330         checkOut.isEnabled = false
331         performPaymentHubtel(price, title)
332     }
333 
334     private fun performPaymentHubtel(price: Double, description: String?) {
```

```
335         try {
336             val config: SessionConfiguration = SessionConfiguration().Builder()
337                     .setSecretKey(BuildConfig.HUBTEL_CLIENT_SECRET)
338                     .setClientId(BuildConfig.HUBTEL_CLIENT_ID)
339                     .setEnvironment(HUBTEL_CONFIG_ENVIRONMENT)
340                     .build()
341             val checkout = HubtelCheckout(config)
342             checkout.setPaymentDetails(price, description)
343             checkout.Pay(this)
344             checkout.setOnPaymentCallback(object : OnPaymentResponse {
345                 override fun onCancelled() {
346                     // ...
347                     Toast.makeText(applicationContext, "Process cancelled",
348                             Toast.LENGTH_LONG).show()
349                     checkOut.isEnabled = true
350                 }
351 
352                 override fun onFailed(p0: String?, p1: String?) {
353                     // ...
354                     Toast.makeText(applicationContext, "Error: $p0", Toast.LENGTH_LONG).show()
355                     checkOut.isEnabled = true
356                 }
357 
358                 override fun onSuccessful(p0: String?) {
359                     // ...
360                     setResultAndFinish(price)
361                 }
362             })
363         } catch (e: HubtelPaymentException) {
364             // ...
365             Timber.e(e)
366             Toast.makeText(applicationContext, e.message, Toast.LENGTH_LONG).show()
367         }
368     }
369 
370     private fun textForMethod(text: String) {
371         TransitionManager.beginDelayedTransition(bottomSheetContent)
372         orderMethod.text = text
373     }
374 
375     private fun setResultAndFinish(price: Double?) {
376         val intent = Intent()
377         intent.putExtra(CartActivity.RESULT_PRICE, price)
378         setResult(RESULT_PAYING, intent)
379         finishAfterTransition()
380     }
381 
382     override fun onPause() {
383         // customize window animations
384         overridePendingTransition(R.anim.post_story_enter, R.anim.post_story_exit)
385         super.onPause()
386     }
387 
388     override fun onBackPressed() {
389         dismiss()
390     }
391 
392     override fun onDestroy() {
393         // Stop service when done
394         stopService(Intent(this, PayPalService::class.java))
395         super.onDestroy()
396     }
397 
398     override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
399         super.onActivityResult(requestCode, resultCode, data)
400         when (requestCode) {
401             RC_AUTH_PAYMENT -> when (resultCode) {
```

```
            RESULT_OK -> {
                Toast.makeText(this, "You can now continue with your purchase",
                    Toast.LENGTH_SHORT).show()
            }
            RESULT_CANCELED, RESULT_FIRST_USER -> {
                //do nothing when authentication is cancelled by user
                checkOut.isEnabled = true
            }
        }
        RC_PAYMENT -> when (resultCode) {
            RESULT_OK -> {
                Toast.makeText(this, "Purchase was successful. Clearing cart data",
                    Toast.LENGTH_SHORT).show()
                if (client.isConnected) clearData()
                else {
                    val intent = Intent()
                    intent.putExtra(CartActivity.RESULT_PRICE, "dummy")
                    setResult(RESULT_PAYING, intent)
                    finishAfterTransition()
                }
            }
            RESULT_FIRST_USER, RESULT_CANCELED -> {
                Toast.makeText(this, "We were unable to complete your purchase",
                    Toast.LENGTH_LONG).show()
                checkOut.isEnabled = true
            }
        }
        RESULT_UPDATE_PROFILE -> when (resultCode) {
            RESULT_OK -> {
                loadUser()
            }
        }
        LOAD_PAYMENT_DATA_REQUEST_CODE -> when (resultCode) {
            RESULT_OK -> {
                if (data != null) {
                    val paymentData = PaymentData.getFromIntent(data)
                    val token = paymentData?.paymentMethodToken?.token
                    Toast.makeText(this, "Purchase was successful with token: $token",
                        Toast.LENGTH_SHORT).show()
                    if (client.isConnected) clearData()
                } else {
                    Toast.makeText(this, "We were unable to retrieve your data",
                        Toast.LENGTH_LONG).show()
                }
            }
            RESULT_CANCELED, AutoResolveHelper.RESULT_ERROR -> {
                // Log the status for debugging.
                // Generally, there is no need to show an error to
                // the user as the Google Pay API will do that.
                Toast.makeText(this, "We were unable to complete your purchase",
                    Toast.LENGTH_LONG).show()
                checkOut.isEnabled = true
            }
        }
    }
}

private fun clearData() {
    if (client.isLoggedIn) {
        TransitionManager.beginDelayedTransition(bottomSheetContent)
        checkOut.isEnabled = false
        loading.visibility = View.VISIBLE
        client.db.collection("${PhoenixUtils.ORDER_REF}/${client.customer.key!!}")
            .get()
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    var complete = false
```

```
                    for (doc in task.result.documents) {
                        doc.reference.delete()
                                .addOnCompleteListener(this@OrderActivity, { delTask ->
                                    Timber.d("${delTask.isComplete} returned")
                                    complete = true
                                })
                    }

                    if (complete) {
                        TransitionManager.beginDelayedTransition(bottomSheetContent)
                        loading.visibility = View.GONE
                        val intent = Intent()
                        intent.putExtra(CartActivity.RESULT_PRICE, "dummy")
                        setResult(RESULT_PAYING, intent)
                        finishAfterTransition()
                    }

                } else {
                    Toast.makeText(this, "Failed to empty cart with error: " +
                            "${task.exception?.localizedMessage}",
                            Toast.LENGTH_LONG).show()
                    checkOut.isEnabled = true
                }
            }
        } else {
            Toast.makeText(applicationContext, "Cannot empty your shopping cart",
                    Toast.LENGTH_SHORT).show()
            checkOut.isEnabled = true
        }
    }

    private fun dismiss() {
        if (!hasSharedElementTransition()) {
            bottomSheetContent.animate()
                    .translationY(bottomSheetContent.height.toFloat())
                    .setDuration(160L)
                    .setInterpolator(AnimUtils.getFastOutLinearInInterpolator(this@OrderActivity))
                    .setListener(object : AnimatorListenerAdapter() {
                        override fun onAnimationEnd(animation: Animator) {
                            finish()
                        }
                    })
        } else {
            finishAfterTransition()
        }
    }

    private fun isShareIntent(): Boolean {
        return intent != null && Intent.ACTION_SEND == intent.action
    }

    private fun hasSharedElementTransition(): Boolean {
        val transition = window.sharedElementEnterTransition
        return transition != null && !transition.targets.isEmpty()
    }

    /**
     * @param value converted from double to string
     */
    private fun setValue(value: Double): String {
        return NumberFormat.getCurrencyInstance(Locale.US).format(value)
    }

    /**
     * @param view converts text of TextView to double
     */
    private fun valueOf(view: TextView): Double {
```

```
536         val value = view.text.toString()
537         //Handle exceptions here
538         if (TextUtils.isEmpty(value) || !TextUtils.isDigitsOnly(value)) return 0.00
539         return value.toDouble()
540     }
541
542     companion object {
543         const val EXTRA_CART_PRICE = "EXTRA_CART_PRICE"
544         const val EXTRA_CART_TITLE = "EXTRA_CART_TITLE"
545         const val EXTRA_CART_QUANTITY = "EXTRA_CART_QUANTITY"
546         const val RESULT_DRAG_DISMISSED = 19
547         const val RESULT_PAYING = 20
548         const val RC_AUTH_PAYMENT = 21
549         const val RC_PAYMENT = 22
550         const val RESULT_UPDATE_PROFILE = 23
551         private const val LOAD_PAYMENT_DATA_REQUEST_CODE = 24
552         private const val REQUEST_CODE_PAYMENT = 25
553
554         /**
555          * - Set to PayPalConfiguration.ENVIRONMENT_PRODUCTION to move real money.
556
557          * - Set to PayPalConfiguration.ENVIRONMENT_SANDBOX to use your test credentials
558          * from https://developer.paypal.com
559
560          * - Set to PayPalConfiguration.ENVIRONMENT_NO_NETWORK to kick the tires
561          * without communicating to PayPal's servers.
562          */
563         private const val PAYPAL_CONFIG_ENVIRONMENT = PayPalConfiguration.
    ENVIRONMENT_SANDBOX
564         private val HUBTEL_CONFIG_ENVIRONMENT = Environment.LIVE_MODE
565
566         private val config = PayPalConfiguration()
567                 .environment(PAYPAL_CONFIG_ENVIRONMENT)
568                 .clientId(BuildConfig.PAYPAL_CLIENT_ID)
569                 .rememberUser(true)
570                 .merchantName("The Phoenix")
571                 .merchantPrivacyPolicyUri(Uri.parse("https://phoenix-master.firebaseapp.com/privacy.html"))
572                 .merchantUserAgreementUri(Uri.parse("https://phoenix-master.firebaseapp.com/legal.html"))
573
574     }
575
576 }
577
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.app.Activity
8  import android.app.ActivityOptions
9  import android.content.Intent
10 import android.graphics.drawable.AnimatedVectorDrawable
11 import android.os.Build
12 import android.os.Bundle
13 import android.support.v4.content.ContextCompat
14 import android.support.v7.widget.GridLayoutManager
15 import android.support.v7.widget.RecyclerView
16 import android.transition.TransitionManager
17 import android.view.View
18 import android.view.ViewGroup
19 import android.widget.*
20 import com.bumptech.glide.integration.recyclerview.RecyclerViewPreloader
21 import com.bumptech.glide.load.engine.DiskCacheStrategy
22 import com.bumptech.glide.load.resource.drawable.DrawableTransitionOptions.withCrossFade
23 import com.bumptech.glide.request.RequestOptions
24 import com.bumptech.glide.request.target.Target
25 import com.bumptech.glide.util.ViewPreloadSizeProvider
26 import com.google.android.gms.tasks.Task
27 import com.google.firebase.firestore.EventListener
28 import com.google.firebase.firestore.QuerySnapshot
29 import io.pergasus.BuildConfig
30 import io.pergasus.R
31 import io.pergasus.api.PhoenixClient
32 import io.pergasus.api.PhoenixUtils
33 import io.pergasus.api.ShopDataManager
34 import io.pergasus.data.Follow
35 import io.pergasus.data.Product
36 import io.pergasus.data.Shop
37 import io.pergasus.ui.recyclerview.SlideInItemAnimator
38 import io.pergasus.ui.transitions.MorphTransform
39 import io.pergasus.ui.widget.ElasticDragDismissFrameLayout
40 import io.pergasus.util.HtmlUtils
41 import io.pergasus.util.ViewUtils
42 import io.pergasus.util.bindView
43 import io.pergasus.util.glide.GlideApp
44 import timber.log.Timber
45 import java.text.NumberFormat
46 import java.util.*
47
48
49 class ShopsActivity : Activity() {
50
51     private val draggableFrame: ElasticDragDismissFrameLayout by bindView(R.id.draggable_frame)
52     private val container: ViewGroup by bindView(R.id.container)
53     private val name: TextView by bindView(R.id.shop_name)
54     private val about: TextView by bindView(R.id.shop_bio)
55     private val avatar: ImageView by bindView(R.id.avatar)
56     private val follow: Button by bindView(R.id.follow)
57     private val productsCount: TextView by bindView(R.id.products_count)
58     private val followersCount: TextView by bindView(R.id.followers_count)
59     private val grid: RecyclerView by bindView(R.id.shop_products)
60     private val loading: ProgressBar by bindView(R.id.loading)
61
62     private var following: Boolean = false
63     private lateinit var chromeFader: ElasticDragDismissFrameLayout.SystemChromeFader
64     private var followerCount: Int = 0
65     private var shop: Shop? = null
66     private lateinit var prefs: PhoenixClient
```

```
67
68      private var adapter: SearchDataAdapter? = null
69      private var layoutManager: GridLayoutManager? = null
70      private var dataManager: ShopDataManager? = null
71      private var columns: Int = 0
72  ----------------------------------------------------------------------
73      override fun onCreate(savedInstanceState: Bundle?) {
74          super.onCreate(savedInstanceState)
75          setContentView(R.layout.activity_shops)
76
77          chromeFader = ElasticDragDismissFrameLayout.SystemChromeFader(this)
78          prefs = PhoenixClient(this@ShopsActivity)
79
80          val intent = intent
81          if (intent.hasExtra(EXTRA_SHOP_NAME)) {
82              val extra = intent.getStringExtra(EXTRA_SHOP_NAME)
83              name.text = String.format("%s (Loading)", extra)
84              if (intent.hasExtra(EXTRA_SHOP_KEY)) {
85                  val key = intent.getStringExtra(EXTRA_SHOP_KEY)
86                  //Load shop by key, if any else load by name
87                  if (key.isNullOrEmpty()) loadShop(extra)
88                  else loadShop(key)
89              }
90          }
91
92          // setup immersive mode i.e. draw behind the system chrome & adjust insets
93          draggableFrame.systemUiVisibility = (View.SYSTEM_UI_FLAG_LAYOUT_STABLE
94                  or View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
95                  or View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION)
96
97          if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
98              draggableFrame.systemUiVisibility = (View.SYSTEM_UI_FLAG_LIGHT_NAVIGATION_BAR)
99          }
100
101         draggableFrame.setOnApplyWindowInsetsListener({ _, insets ->
102             val params = draggableFrame.layoutParams as ViewGroup.MarginLayoutParams
103             params.leftMargin += insets.systemWindowInsetLeft  //left margin
104             params.rightMargin += insets.systemWindowInsetRight  //right margin
105             (avatar.layoutParams as ViewGroup.MarginLayoutParams).topMargin += insets.
    systemWindowInsetTop
106             ViewUtils.setPaddingTop(container, insets.systemWindowInsetTop)
107             ViewUtils.setPaddingBottom(grid, insets.systemWindowInsetBottom)
108             // clear this listener so insets aren't re-applied
109             draggableFrame.setOnApplyWindowInsetsListener(null)
110             return@setOnApplyWindowInsetsListener insets
111         })
112
113         setExitSharedElementCallback(SearchDataAdapter.createSharedElementReenterCallback(this))
114     }
115 ----------------------------------------------------------------------
116     private fun loadShop(extra: String?) {
117         if (extra == null) return
118         prefs.db.collection(PhoenixUtils.DB_PREFIX + "/" + PhoenixUtils.SHOP_REF)
119                 .get()
120                 .addOnFailureListener { exception ->
121                     Toast.makeText(this@ShopsActivity, exception.localizedMessage,
122                             Toast.LENGTH_SHORT).show()
123                 }.addOnCompleteListener { task ->
124                     if (task.isSuccessful) {
125                         for (shops in task.result.documents) {
126                             if (shops.exists()) {
127                                 val obj = shops.toObject(Shop::class.java)
128                                 if (obj.name == extra || obj.key == extra) {
129                                     shop = obj  //Assign shop to object
130                                     bindShop()
131                                 }
132                             }
```

```
                }
            } else {
                Toast.makeText(this@ShopsActivity, task.exception?.localizedMessage,
                    Toast.LENGTH_SHORT).show()
            }
        }
    }

    private fun bindShop() {
        Timber.d("bindShop called successfully")
        val res = resources
        val nf: NumberFormat = NumberFormat.getInstance()

        //Load image from url
        GlideApp.with(this)
            .load(shop?.logo)
            .apply(RequestOptions().circleCrop())
            .apply(RequestOptions().diskCacheStrategy(DiskCacheStrategy.AUTOMATIC))
            .apply(RequestOptions().error(R.drawable.ic_player))
            .apply(RequestOptions().override(Target.SIZE_ORIGINAL, Target.SIZE_ORIGINAL))
            .apply(RequestOptions().placeholder(R.drawable.avatar_placeholder))
            .transition(withCrossFade())
            .into(avatar)

        //Name
        name.text = shop?.name

        //Bio
        if (shop?.motto == null) {
            about.visibility = View.GONE
        } else {
            HtmlUtils.parseAndSetText(about, shop?.motto)
        }

        setFollowersCount(shop?.followers_count)

        //Stats
        productsCount.text = res.getQuantityString(R.plurals.purchases, shop?.products_count!!
            .toLong().toInt(), nf.format(shop?.products_count!!))

        //Follow button
        follow.setOnClickListener(doFollow)

        //Add ripple and animated drawable effect to textViews
        followersCount.setOnClickListener {
            actionClick(followersCount)
        }
        productsCount.setOnClickListener {
            actionClick(productsCount)
        }

        dataManager = object : ShopDataManager(this@ShopsActivity, shop!!) {
            override fun onDataLoaded(data: List<Product>) {
                if (data.isNotEmpty()) {
                    if (adapter?.dataItemCount == 0) {
                        loading.visibility = View.GONE
                        ViewUtils.setPaddingTop(grid, productsCount.bottom)
                    }
                    adapter?.addAndResort(data)
                }
            }
        }

        columns = resources.getInteger(R.integer.num_columns)
        //RecyclerView
        val preloadSizeProvider: ViewPreloadSizeProvider<Product> = ViewPreloadSizeProvider()
        adapter = SearchDataAdapter(this, dataManager, columns, preloadSizeProvider)
```

```
200         setExitSharedElementCallback(SearchDataAdapter.createSharedElementReenterCallback(this))
201         grid.adapter = adapter
202         grid.itemAnimator = SlideInItemAnimator()
203         grid.visibility = View.VISIBLE
204         layoutManager = GridLayoutManager(this, columns)
205         layoutManager?.spanSizeLookup = object : GridLayoutManager.SpanSizeLookup() {
206             override fun getSpanSize(position: Int): Int {
207                 return adapter?.getItemColumnSpan(position)!!
208             }
209         }
210         grid.layoutManager = layoutManager
211         grid.setHasFixedSize(true)
212         val shotPreloader: RecyclerViewPreloader<Product> = RecyclerViewPreloader<Product>(this,
213                 adapter!!, preloadSizeProvider, 4)
214         grid.addOnScrollListener(shotPreloader)
215         // forward on any clicks above the first item in the grid (i.e. in the paddingTop)
216         // to 'pass through' to the view behind
217         grid.setOnTouchListener(View.OnTouchListener { _, event ->
218             val firstVisible = layoutManager?.findFirstVisibleItemPosition()!!
219             if (firstVisible > 0) return@OnTouchListener false
220
221             // if no data loaded then pass through
222             if (adapter?.dataItemCount == 0) {
223                 return@OnTouchListener container.dispatchTouchEvent(event)
224             }
225
226             val vh = grid.findViewHolderForAdapterPosition(0) ?: return@OnTouchListener false
227             val firstTop = vh.itemView.top
228             if (event?.y!! < firstTop) {
229                 return@OnTouchListener container.dispatchTouchEvent(event)
230             }
231             return@OnTouchListener false
232         })
233
234         if (dataManager?.phoenixClient!!.isLoggedIn) {
235             //Query db for the current user
236             dataManager?.phoenixClient!!
237                     .db.document(PhoenixUtils.FOLLOW_REF)
238                     .collection(shop?.key!!)
239                     .orderBy("timestamp")
240                     .addSnapshotListener(this@ShopsActivity, EventListener<QuerySnapshot?> { p0, p1 ->
241                         if (p1 != null) {
242                             Timber.d(p1.localizedMessage)
243                             return@EventListener
244                         }
245                         if (p0 != null) {
246                             for (data in p0.documents) {
247                                 if (data.exists()) {
248                                     val follow1 = data.toObject(Follow::class.java)
249                                     if (follow1.customer?.key == prefs.customer.key) {
250                                         //Set following to be true
251                                         TransitionManager.beginDelayedTransition(container)
252                                         follow.setText(R.string.following)
253                                         follow.isActivated = true
254                                     }
255                                 }
256                             }
257                         }
258                     })
259         }
260
261         if (shop?.products_count!! > 0) {
262             dataManager?.loadData()
263         } else {
264             loading.visibility = View.GONE
265         }
266     }
```

```
267
268     private fun actionClick(view: TextView) {
269         (view.compoundDrawables[1] as AnimatedVectorDrawable).start()
270         if (view.id == R.id.followers_count) {
271             UserSheet.start(this@ShopsActivity, shop!!)
272         }
273     }
274
275     private fun setFollowersCount(l: Long?) {
276         if (l == null) return
277         followerCount = l.toInt()
278         followersCount.text = resources.getQuantityString(R.plurals.followers, followerCount,
279                 NumberFormat.getInstance().format(followerCount))
280         if (followerCount == 0) {
281             followersCount.background = null
282         }
283     }
284
285     private val doFollow: View.OnClickListener = View.OnClickListener {
286         if (prefs.isLoggedIn) {
287             if (following) {
288                 prefs.db.document(PhoenixUtils.FOLLOW_REF)
289                         .collection(shop?.key!!)
290                         .document(prefs.customer.key!!)
291                         .delete()
292                         .addOnCompleteListener { task ->
293                             if (task.isSuccessful) {
294                                 updateFollowers(task, shop?.followers_count!!.minus(1))
295                             }
296                         }
297                         .addOnFailureListener { exception ->
298                             if (BuildConfig.DEBUG) {
299                                 Timber.d(exception.localizedMessage)
300                             }
301                         }
302                 following = false
303                 TransitionManager.beginDelayedTransition(container)
304                 follow.setText(R.string.follow)
305                 follow.isActivated = false
306                 setFollowersCount((followerCount - 1).toLong())
307             } else {
308                 //Get user instance and create a map from it
309                 val customer = prefs.customer
310                 //Create follow instance
311                 val hashMap = hashMapOf(
312                         Pair<String, Any?>("id", System.currentTimeMillis()),
313                         Pair<String, Any?>("customer", customer.toHashMap(customer)),
314                         Pair<String, Any?>("shop", shop?.toHashMap(shop!!)),
315                         Pair<String, Any?>("timestamp", Date(System.currentTimeMillis()))
316                 )
317
318                 //Push to database
319                 prefs.db.document(PhoenixUtils.FOLLOW_REF)
320                         .collection(shop?.key!!)
321                         .document(prefs.customer.key!!)
322                         .set(hashMap)
323                         .addOnFailureListener { exception ->
324                             if (BuildConfig.DEBUG) {
325                                 Timber.d(exception.localizedMessage)
326                             }
327                         }.addOnCompleteListener { task ->
328                             if (task.isSuccessful) {
329                                 updateFollowers(task, shop?.followers_count!!.plus(1))
330                             } else {
331                                 if (BuildConfig.DEBUG) {
332                                     Timber.d(task.exception?.localizedMessage)
333                                 }
```

```
                    }
                }
            following = true
            TransitionManager.beginDelayedTransition(container)
            follow.setText(R.string.following)
            follow.isActivated = true
            setFollowersCount((followerCount + 1).toLong())

        }
    } else {
        val login = Intent(this, AuthActivity::class.java)
        MorphTransform.addExtras(login,
                ContextCompat.getColor(this, R.color.button_accent),
                resources.getDimensionPixelSize(R.dimen.dialog_corners))
        val options = ActivityOptions.makeSceneTransitionAnimation(this, follow, getString(R.string.
transition_dribbble_login))
        startActivity(login, options.toBundle())
    }
}

private fun updateFollowers(task: Task<Void>, value: Long) {
    //Add to shop's follower's count as well
    //Ref: /phoenix/web/shops/{shop_key}
    prefs.db.document(PhoenixUtils.DB_PREFIX + "/" + PhoenixUtils.SHOP_REF + "/" + shop?.key)
            .update("followers_count", value)
            .addOnCompleteListener { updateTask ->
                if (updateTask.isSuccessful) {
                    if (BuildConfig.DEBUG) {
                        Timber.d("Followers updated")
                    }
                } else {
                    if (BuildConfig.DEBUG) {
                        Timber.d(task.exception?.localizedMessage)
                    }
                }
            }
}

override fun onResume() {
    super.onResume()
    draggableFrame.addListener(chromeFader)
}

override fun onPause() {
    draggableFrame.removeListener(chromeFader)
    super.onPause()
}

override fun onDestroy() {
    if (dataManager != null) {
        dataManager!!.cancelLoading()
    }
    super.onDestroy()
}

override fun onActivityReenter(resultCode: Int, data: Intent?) {
    if (data == null || resultCode != RESULT_OK
            || !data.hasExtra(DetailsActivity.RESULT_EXTRA_SHOT_ID)) return
    // When reentering, if the shared element is no longer on screen (e.g. after an
    // orientation change) then scroll it into view.
    val sharedShotId = data.getLongExtra(DetailsActivity.RESULT_EXTRA_SHOT_ID, -1L)
    if (sharedShotId != -1L                                 // returning from a shot
            && adapter?.dataItemCount!! > 0                 // grid populated
            && grid.findViewHolderForItemId(sharedShotId) == null) {    // view not attached
        val position = adapter?.getItemPosition(sharedShotId)
        if (position == RecyclerView.NO_POSITION) return

```

```
400             // delay the transition until our shared element is on-screen i.e. has been laid out
401             postponeEnterTransition()
402             grid.addOnLayoutChangeListener(object : View.OnLayoutChangeListener {
403                 override fun onLayoutChange(p0: View?, p1: Int, p2: Int, p3: Int, p4: Int, p5: Int, p6: Int, p7: Int, p8:
        Int) {
404                     grid.removeOnLayoutChangeListener(this)
405                     startPostponedEnterTransition()
406                 }
407             })
408             if (position != null) {
409                 grid.scrollToPosition(position)
410             }
411         }
412     }
413
414     companion object {
415         const val EXTRA_SHOP_NAME = "EXTRA_SHOP_NAME"
416         const val EXTRA_SHOP_KEY = "EXTRA_SHOP_KEY"
417
418     }
419 }
420
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.annotation.SuppressLint
8  import android.app.Activity
9  import android.app.SearchManager
10 import android.app.SharedElementCallback
11 import android.content.Intent
12 import android.graphics.Point
13 import android.graphics.Typeface
14 import android.os.Bundle
15 import android.support.annotation.TransitionRes
16 import android.support.v7.widget.GridLayoutManager
17 import android.support.v7.widget.RecyclerView
18 import android.text.InputType
19 import android.text.SpannableStringBuilder
20 import android.text.Spanned
21 import android.text.TextUtils
22 import android.text.style.StyleSpan
23 import android.transition.Transition
24 import android.transition.TransitionInflater
25 import android.transition.TransitionManager
26 import android.transition.TransitionSet
27 import android.util.Log
28 import android.util.SparseArray
29 import android.view.View
30 import android.view.ViewGroup
31 import android.view.ViewStub
32 import android.view.inputmethod.EditorInfo
33 import android.widget.*
34 import com.bumptech.glide.integration.recyclerview.RecyclerViewPreloader
35 import com.bumptech.glide.util.ViewPreloadSizeProvider
36 import io.pergasus.BuildConfig
37 import io.pergasus.R
38 import io.pergasus.api.PhoenixClient
39 import io.pergasus.api.ProductItem
40 import io.pergasus.api.SearchDataManager
41 import io.pergasus.data.Product
42 import io.pergasus.ui.recyclerview.InfiniteScrollListener
43 import io.pergasus.ui.recyclerview.SlideInItemAnimator
44 import io.pergasus.ui.transitions.CircularReveal
45 import io.pergasus.util.ImeUtils
46 import io.pergasus.util.ShortcutHelper
47 import io.pergasus.util.TransitionUtils
48 import io.pergasus.util.bindView
49
50
51 @SuppressLint("LogConditional")
52 class SearchActivity : Activity() {
53     private val searchBack: ImageButton by bindView(R.id.searchback)
54     private val searchBackContainer: ViewGroup by bindView(R.id.searchback_container)
55     private val searchView: SearchView by bindView(R.id.search_view)
56     private val searchBackground: View by bindView(R.id.search_background)
57     private val progress: ProgressBar by bindView(android.R.id.empty)
58     private val results: RecyclerView by bindView(R.id.search_results)
59     private val container: ViewGroup by bindView(R.id.container)
60     private val searchToolbar: ViewGroup by bindView(R.id.search_toolbar)
61     private val resultsContainer: ViewGroup by bindView(R.id.results_container)
62     private val fab: ImageButton by bindView(R.id.fab)
63     private val saveConfirm: Button by bindView(R.id.save_confirmed)
64     private val confirmSaveContainer: ViewGroup by bindView(R.id.confirm_save_container)
65     private val scrim: View by bindView(R.id.scrim)
66     private val resultsScrim: View by bindView(R.id.results_scrim)
```

```
67
68      private var columns: Int = 0
69      private var appBarElevation: Float = 0.0f
70      private var noResults: TextView? = null
71      private val transitions = SparseArray<Transition>()
72      private var focusQuery = true
73
74      private lateinit var adapter: SearchDataAdapter
75      private lateinit var layoutManager: GridLayoutManager
76      private lateinit var dataManager: SearchDataManager
77      private lateinit var client: PhoenixClient
78
79      override fun onCreate(savedInstanceState: Bundle?) {
80          super.onCreate(savedInstanceState)
81          setContentView(R.layout.activity_search)
82
83          client = PhoenixClient(this@SearchActivity)
84          columns = resources.getInteger(R.integer.num_columns)
85          appBarElevation = resources.getDimension(R.dimen.z_app_bar)
86          setupSearchView()
87
88          resultsScrim.setOnClickListener({ hideSaveConfirmation() })
89          fab.setOnClickListener({ save() })
90          saveConfirm.setOnClickListener({ doSave() })
91
92          searchBack.setOnClickListener({ dismiss() })
93
94          dataManager = object : SearchDataManager(this@SearchActivity) {
95              override fun onDataLoaded(data: List<ProductItem>) {
96                  if (BuildConfig.DEBUG) Log.d(TAG, "Data from search is $data")
97                  if (data.isNotEmpty()) {
98                      if (results.visibility != View.VISIBLE) {
99                          TransitionManager.beginDelayedTransition(container,
100                                 getTransition(R.transition.search_show_results))
101                         progress.visibility = View.GONE
102                         results.visibility = View.VISIBLE
103                         fab.visibility = View.VISIBLE
104                     }
105                     adapter.addAndResort(data)
106                 } else {
107                     TransitionManager.beginDelayedTransition(
108                             container, getTransition(R.transition.auto))
109                     progress.visibility = View.GONE
110                     setNoResultsVisibility(View.VISIBLE)
111                 }
112             }
113         }
114
115         //RecyclerView
116         val preloadSizeProvider: ViewPreloadSizeProvider<Product> = ViewPreloadSizeProvider()
117         adapter = SearchDataAdapter(this, dataManager, columns, preloadSizeProvider)
118         setExitSharedElementCallback(SearchDataAdapter.createSharedElementReenterCallback(this))
119         results.adapter = adapter
120         results.itemAnimator = SlideInItemAnimator()
121         layoutManager = GridLayoutManager(this, columns)
122         layoutManager.spanSizeLookup = object : GridLayoutManager.SpanSizeLookup() {
123             override fun getSpanSize(position: Int): Int {
124                 return adapter.getItemColumnSpan(position)
125             }
126         }
127         results.layoutManager = layoutManager
128         results.setHasFixedSize(true)
129         val shotPreloader: RecyclerViewPreloader<Product> = RecyclerViewPreloader<Product>(this,
130                 adapter, preloadSizeProvider, 4)
131         results.addOnScrollListener(shotPreloader)
132
133         scrim.setOnClickListener { dismiss() }
```

```
134         setupTransitions()
135         onNewIntent(intent)
136         ShortcutHelper.reportSearchUsed(this@SearchActivity)
137     }
138 
139     override fun onNewIntent(intent: Intent?) {
140         if (intent != null && intent.hasExtra(SearchManager.QUERY)) {
141             val query = intent.getStringExtra(SearchManager.QUERY)
142             if (!TextUtils.isEmpty(query)) {
143                 searchView.setQuery(query, false)
144                 searchFor(query)
145             }
146         }
147     }
148 
149     override fun onBackPressed() {
150         if (confirmSaveContainer.visibility == View.VISIBLE) {
151             hideSaveConfirmation()
152         } else {
153             dismiss()
154         }
155     }
156 
157     override fun onPause() {
158         // needed to suppress the default window animation when closing the activity
159         overridePendingTransition(0, 0)
160         super.onPause()
161     }
162 
163     override fun onDestroy() {
164         dataManager.cancelLoading()
165         super.onDestroy()
166     }
167 
168 
169     private fun save() {
170         // show the save confirmation bubble
171         TransitionManager.beginDelayedTransition(
172                 resultsContainer, getTransition(R.transition.search_show_confirm))
173         fab.visibility = View.INVISIBLE
174         confirmSaveContainer.visibility = View.VISIBLE
175         resultsScrim.visibility = View.VISIBLE
176     }
177 
178     private fun doSave() {
179         val saveData = Intent()
180         saveData.putExtra(EXTRA_QUERY, dataManager.getQuery())
181         setResult(RESULT_CODE_SAVE, saveData)
182         dismiss()
183     }
184 
185     private fun dismiss() {
186         // clear the background else the touch ripple moves with the translation which looks bad
187         searchBack.background = null
188         finishAfterTransition()
189     }
190 
191     private fun hideSaveConfirmation() {
192         if (confirmSaveContainer.visibility == View.VISIBLE) {
193             TransitionManager.beginDelayedTransition(
194                     resultsContainer, getTransition(R.transition.search_hide_confirm))
195             confirmSaveContainer.visibility = View.GONE
196             resultsScrim.visibility = View.GONE
197             fab.visibility = results.visibility
198         }
199     }
200 
```

```
201     private fun clearResults() {
202         TransitionManager.beginDelayedTransition(container, getTransition(R.transition.auto))
203         adapter.clear()
204         dataManager.clear()
205         results.visibility = View.GONE
206         progress.visibility = View.GONE
207         fab.visibility = View.GONE
208         confirmSaveContainer.visibility = View.GONE
209         resultsScrim.visibility = View.GONE
210         setNoResultsVisibility(View.GONE)
211     }
212 
213     private fun setNoResultsVisibility(visibility: Int) {
214         if (visibility == View.VISIBLE) {
215             if (noResults == null) {
216                 noResults = (findViewById<View>(R.id.stub_no_search_results) as ViewStub).inflate() as
        TextView
217                 noResults?.setOnClickListener({
218                     searchView.setQuery("", false)
219                     searchView.requestFocus()
220                     ImeUtils.showIme(searchView)
221                 })
222             }
223             val message = String.format(
224                     getString(R.string.no_search_results), searchView.query.toString())
225             val ssb = SpannableStringBuilder(message)
226             ssb.setSpan(StyleSpan(Typeface.ITALIC),
227                     message.indexOf("") + 1,
228                     message.length - 1,
229                     Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)
230             noResults?.text = ssb
231         }
232         if (noResults != null) {
233             noResults?.visibility = visibility
234         }
235     }
236 
237     private fun getTransition(@TransitionRes transitionId: Int): Transition {
238         var transition = transitions.get(transitionId)
239         if (transition == null) {
240             transition = TransitionInflater.from(this).inflateTransition(transitionId)
241             transitions.put(transitionId, transition)
242         }
243         return transition
244     }
245 
246     private fun setupSearchView() {
247         val searchManager: SearchManager = getSystemService(SEARCH_SERVICE) as SearchManager
248         searchView.setSearchableInfo(searchManager.getSearchableInfo(componentName))
249         // hint, inputType & ime options seem to be ignored from XML! Set in code
250         searchView.queryHint = getString(R.string.search_hint)
251         searchView.inputType = InputType.TYPE_TEXT_FLAG_CAP_WORDS
252         searchView.imeOptions = searchView.imeOptions or EditorInfo.IME_ACTION_SEARCH
253         EditorInfo.IME_FLAG_NO_EXTRACT_UI or EditorInfo.IME_FLAG_NO_FULLSCREEN
254 
255         searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener {
256             override fun onQueryTextSubmit(p0: String?): Boolean {
257                 if (p0 != null) {
258                     searchFor(p0)
259                 }
260                 return true
261             }
262 
263             override fun onQueryTextChange(p0: String?): Boolean {
264                 if (TextUtils.isEmpty(p0)) {
265                     clearResults()
266                 }
```

```
267             return true
268         }
269       })
270       searchView.setOnQueryTextFocusChangeListener { _, hasFocus ->
271         if (hasFocus && confirmSaveContainer.visibility == View.VISIBLE) {
272           hideSaveConfirmation()
273         }
274       }
275     }
276 
277     private fun setupTransitions() {
278       // grab the position that the search icon transitions in *from*
279       // & use it to configure the return transition
280       setEnterSharedElementCallback(object : SharedElementCallback() {
281         override fun onSharedElementStart(sharedElementNames: MutableList<String>?, sharedElements:
      MutableList<View>?, sharedElementSnapshots: MutableList<View>?) {
282           if (sharedElements != null && !sharedElements.isEmpty()) {
283             val searchIcon: View = sharedElements[0]
284             if (searchIcon.id != R.id.searchback) return
285             val centerX: Int = (searchIcon.left + searchIcon.right) / 2
286             (TransitionUtils.findTransition(
287                 window.returnTransition as TransitionSet,
288                 CircularReveal::class.java, R.id.results_container) as CircularReveal).setCenter(Point(
      centerX, 0))
289           }
290         }
291       })
292     }
293 
294     override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
295       when (requestCode) {
296         SearchDataAdapter.REQUEST_CODE_VIEW_PRODUCT -> {
297           // by default we focus the search filed when entering this screen. Don't do that
298           // when returning from viewing a search result.
299           focusQuery = false
300         }
301       }
302     }
303 
304     private fun searchFor(query: String) {
305       clearResults()
306       progress.visibility = View.VISIBLE
307       ImeUtils.hideIme(searchView)
308       searchView.clearFocus()
309       if (BuildConfig.DEBUG) Log.d(TAG, "Search started with query: $query")
310       dataManager.searchFor(query)
311     }
312 
313     companion object {
314       private val TAG = SearchActivity::class.java.simpleName
315       val RESULT_CODE_SAVE: Int = 234
316       val EXTRA_QUERY: String = "EXTRA_QUERY"
317       val EXTRA_SAVE: String = "EXTRA_SAVE"
318     }
319   }
320 
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.app.Activity
8  import android.content.Intent
9  import android.graphics.Typeface
10 import android.os.Bundle
11 import android.os.Handler
12 import android.transition.TransitionManager
13 import android.view.View
14 import android.view.ViewGroup
15 import android.widget.ProgressBar
16 import com.afollestad.materialdialogs.MaterialDialog
17 import io.pergasus.R
18 import io.pergasus.api.PhoenixApplication
19 import io.pergasus.api.PhoenixClient
20 import io.pergasus.api.PhoenixUtils
21 import io.pergasus.util.bindView
22 import timber.log.Timber
23
24 /**
25  * SPlash screen for users
26  */
27 class SplashActivity : Activity() {
28     private val container: ViewGroup by bindView(R.id.container)
29     private val loading: ProgressBar by bindView(R.id.loading)
30
31     private lateinit var prefs: PhoenixClient
32
33     override fun onCreate(savedInstanceState: Bundle?) {
34         super.onCreate(savedInstanceState)
35         setContentView(R.layout.activity_splash)
36
37         prefs = PhoenixClient(this)
38
39         val handler = Handler()
40         val clientState = PhoenixApplication.Companion.PhoenixClientState(this)
41         if (clientState.isAppRecentlyInstalled) {
42             handler.postDelayed({
43                 if (prefs.isConnected) {
44                     if (prefs.isLoggedIn) {
45                         TransitionManager.beginDelayedTransition(container)
46                         loading.visibility = View.VISIBLE
47                         loadUserData()
48                     } else {
49                         startActivity(Intent(this@SplashActivity, HomeActivity::class.java))
50                         finish()
51                     }
52                 } else {
53                     startActivity(Intent(this@SplashActivity, HomeActivity::class.java))
54                     finish()
55                 }
56
57             }, 1500)
58         } else {
59             handler.postDelayed({
60                 startActivity(Intent(this@SplashActivity, WelcomeActivity::class.java))
61                 finish()
62             }, 2500)
63         }
64     }
65
66     private fun loadUserData() {
```

```
        prefs.db.document("phoenix/mobile/${PhoenixUtils.CUSTOMER_REF}/${prefs.customer.key}")
                .get()
                .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        Timber.d("User data has been successfully retrieved from the database")
                        loading.visibility = View.GONE
                        startActivity(Intent(this@SplashActivity, HomeActivity::class.java))
                        finish()
                    } else {
                        startActivity(Intent(this@SplashActivity, HomeActivity::class.java))
                        finish()
                    }
                }.addOnFailureListener { exception ->
                    Timber.e(exception, exception.localizedMessage)
                    doLogin()
                }
    }

    private fun doLogin() {
        //Notify user of login state
        MaterialDialog.Builder(this)
                .content("You may not be logged in. Please login to continue")
                .positiveText("Login")
                .negativeText("Exit")
                .typeface(Typeface.createFromAsset(assets, "fonts/nunito_semibold.ttf"),
                        Typeface.createFromAsset(assets, "fonts/nunito_semibold.ttf"))
                .onPositive({ dialog, _ ->
                    dialog.dismiss()
                    loading.visibility = View.GONE
                    startActivity(Intent(this@SplashActivity, AuthActivity::class.java))
                })
                .onNegative({ dialog, _ ->
                    dialog.dismiss()
                    finishAfterTransition()
                }).build().show()
    }
}
```

```
1  /*
2  * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
   should be made without prior permission from the developer
3  */
4
5  package io.pergasus.ui
6
7  import android.animation.ValueAnimator
8  import android.annotation.SuppressLint
9  import android.annotation.TargetApi
10 import android.app.Activity
11 import android.app.ActivityOptions
12 import android.app.assist.AssistContent
13 import android.content.Intent
14 import android.graphics.Bitmap
15 import android.graphics.drawable.AnimatedVectorDrawable
16 import android.graphics.drawable.Drawable
17 import android.net.Uri
18 import android.os.Build
19 import android.os.Bundle
20 import android.support.annotation.NonNull
21 import android.support.customtabs.CustomTabsIntent
22 import android.support.design.widget.Snackbar
23 import android.support.v4.content.ContextCompat
24 import android.support.v7.graphics.Palette
25 import android.support.v7.widget.RecyclerView
26 import android.text.Spanned
27 import android.text.TextUtils
28 import android.text.format.DateUtils
29 import android.transition.AutoTransition
30 import android.transition.Transition
31 import android.transition.TransitionListenerAdapter
32 import android.transition.TransitionManager
33 import android.util.Log
34 import android.util.TypedValue
35 import android.view.View
36 import android.view.ViewGroup
37 import android.view.ViewTreeObserver
38 import android.widget.*
39 import com.bumptech.glide.Priority
40 import com.bumptech.glide.load.DataSource
41 import com.bumptech.glide.load.engine.DiskCacheStrategy
42 import com.bumptech.glide.load.engine.GlideException
43 import com.bumptech.glide.load.resource.drawable.DrawableTransitionOptions.withCrossFade
44 import com.bumptech.glide.request.RequestListener
45 import com.bumptech.glide.request.RequestOptions
46 import com.bumptech.glide.request.target.Target
47 import io.pergasus.BuildConfig
48 import io.pergasus.R
49 import io.pergasus.api.PhoenixClient
50 import io.pergasus.api.PhoenixUtils
51 import io.pergasus.data.Comment
52 import io.pergasus.data.Product
53 import io.pergasus.ui.recyclerview.InsetDividerDecoration
54 import io.pergasus.ui.recyclerview.SlideInItemAnimator
55 import io.pergasus.ui.transitions.FabTransform
56 import io.pergasus.ui.widget.*
57 import io.pergasus.util.*
58 import io.pergasus.util.AnimUtils.getFastOutSlowInInterpolator
59 import io.pergasus.util.customtabs.CustomTabActivityHelper
60 import io.pergasus.util.glide.GlideApp
61 import io.pergasus.util.glide.getBitmap
62 import org.jetbrains.annotations.Nullable
63 import timber.log.Timber
64 import java.text.NumberFormat
65 import java.util.*
66 import kotlin.collections.ArrayList
```

```
67
68  @SuppressLint("LogConditional")
69  /** Details of each category selected is shown here */
70  class DetailsActivity : Activity() {
71
72      //Widgets
73      private val draggableFrame: ElasticDragDismissFrameLayout by bindView(R.id.draggable_frame)
74      private val back: ImageButton by bindView(R.id.back)
75      private val more: ImageButton by bindView(R.id.more)
76      private val imageView: ParallaxScrimageView by bindView(R.id.shot)
77      private val commentsList: RecyclerView by bindView(R.id.dribbble_comments)
78      private val fab: FABToggle by bindView(R.id.fab_heart)
79
80      //Customized widgets
81      private lateinit var shotDescription: View
82      private lateinit var shotSpacer: View
83      private lateinit var likeCount: Button
84      private lateinit var viewCount: Button
85      private lateinit var share: Button
86      private lateinit var lessProduct: ImageButton
87      private lateinit var addProduct: ImageButton
88      private lateinit var productCount: TextView
89      private lateinit var playerAvatar: ImageView
90      private var enterComment: EditText? = null
91      private var postComment: ImageButton? = null
92      private lateinit var title: View
93      private lateinit var price: View
94      private lateinit var description: View
95      private lateinit var shopName: TextView
96      private lateinit var shotTimeAgo: TextView
97      private var userProductCount: Int = 1
98      private var commentFooter: View? = null
99      private lateinit var userAvatar: ImageView
100     private lateinit var chromeFader: ElasticDragDismissFrameLayout.SystemChromeFader
101     private lateinit var commentAnimator: CommentAnimator
102     private lateinit var adapter: CommentsAdapter
103
104     //Others
105     private var fabOffset: Int = 0
106     private var largeAvatarSize: Float = 0.0f
107     private var cardElevation: Float = 0.0f
108     private var product: Product? = null
109     private lateinit var prefs: PhoenixClient
110     private var performingLike: Boolean = false
111     private var allowComment: Boolean = false
112     private val data = ArrayList<Comment>(0)   //Empty arrayList
113
114     override fun onCreate(savedInstanceState: Bundle?) {
115         super.onCreate(savedInstanceState)
116         setContentView(R.layout.activity_details)
117
118         prefs = PhoenixClient(this@DetailsActivity)
119         largeAvatarSize = resources.getDimension(R.dimen.large_avatar_size)
120         cardElevation = resources.getDimension(R.dimen.z_card)
121
122         chromeFader = object : ElasticDragDismissFrameLayout.SystemChromeFader(this) {
123             override fun onDragDismissed() {
124                 setResultAndFinish()
125             }
126         }
127
128         back.setOnClickListener { setResultAndFinish() }
129
130         //SET UP VIEW FOR DETAILS
131         shotDescription = layoutInflater.inflate(R.layout.dribbble_shot_description,
132                 commentsList, false)
133         shotSpacer = shotDescription.findViewById(R.id.shot_spacer)
```

```
134         title = shotDescription.findViewById(R.id.shot_title)
135         description = shotDescription.findViewById(R.id.shot_description)
136         price = shotDescription.findViewById(R.id.shot_price)
137         likeCount = shotDescription.findViewById(R.id.shot_like_count) as Button
138         viewCount = shotDescription.findViewById(R.id.shot_view_count) as Button
139         share = shotDescription.findViewById(R.id.shot_share_action) as Button
140         lessProduct = shotDescription.findViewById(R.id.shot_less_product_count) as ImageButton
141         addProduct = shotDescription.findViewById(R.id.shot_add_product_count) as ImageButton
142         productCount = shotDescription.findViewById(R.id.shot_product_count) as TextView
143         shopName = shotDescription.findViewById(R.id.player_name) as TextView
144         playerAvatar = shotDescription.findViewById(R.id.player_avatar) as ImageView
145         shotTimeAgo = shotDescription.findViewById(R.id.shot_time_ago) as TextView
146
147         setupCommenting()
148         //RECYCLERVIEW
149         commentsList.addOnScrollListener(scrollListener)
150         commentsList.onFlingListener = flingListener
151
152         val intent = intent
153         if (intent.hasExtra(EXTRA_SHOT)) {
154             product = intent.getParcelableExtra(EXTRA_SHOT)
155             bindProduct(true)
156         } /*else if (intent.data != null) {
157             val url: HttpUrl? = HttpUrl.parse(intent.dataString)
158             if (url != null && url.pathSize() == 2 && url.pathSegments()[0] == "product") {
159                 try {
160                     //val shotPath: String = url.pathSegments()[1]
161                     //val id: Long = parseLong(shotPath.substring(0, shotPath.indexOf("-")))
162
163                 } catch (e: NumberFormatException) {
164                     reportUrlError()
165                 } catch (e: StringIndexOutOfBoundsException) {
166                     reportUrlError()
167                 }
168             }
169         }*/
170
171     }
172 ------------------------------------------------------------------------------
173     //Setup details with the provided data from intent
174     @SuppressLint("SetTextI18n")
175     private fun bindProduct(postponeTransition: Boolean) {
176         if (product == null) return
177         val res = resources
178
179         GlideApp.with(this)
180                 .load(product!!.url)
181                 .listener(shotLoadListener)
182                 .apply(RequestOptions().diskCacheStrategy(DiskCacheStrategy.AUTOMATIC))
183                 .apply(RequestOptions().priority(Priority.IMMEDIATE))
184                 .override(800, 600)
185                 .transition(withCrossFade())
186                 .into(imageView)
187         val shotClick: View.OnClickListener = View.OnClickListener {
188             if (product != null) {
189                 val intent = Intent(this@DetailsActivity, ImageDetailsActivity::class.java)
190                 intent.putExtra(ImageDetailsActivity.EXTRA_IMAGE_URL, product)
191                 val options = ActivityOptions.makeSceneTransitionAnimation(this@DetailsActivity,
192                         android.util.Pair.create(imageView, getString(R.string.transition_shot)))
193                 startActivity(intent, options.toBundle())
194             }
195         }
196         imageView.setOnClickListener(shotClick)
197         shotSpacer.setOnClickListener(shotClick)
198
199         if (postponeTransition) postponeEnterTransition()
200
```

```
201         imageView.viewTreeObserver.addOnPreDrawListener(object : ViewTreeObserver.OnPreDrawListener
        {
202             override fun onPreDraw(): Boolean {
203                 imageView.viewTreeObserver.removeOnPreDrawListener(this)
204                 calculateFabPosition()
205                 if (postponeTransition) startPostponedEnterTransition()
206                 return true
207             }
208         })
209
210         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
211             (title as FabOverlapTextView).setText(product?.name)
212         } else {
213             (title as TextView).text = product?.name
214         }
215
216         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
217             (price as FabOverlapTextView).setText(NumberFormat.getCurrencyInstance(Locale.US)
218                     .format((product?.price)?.toDouble()).toString().toLowerCase())
219         } else {
220             (price as TextView).text = NumberFormat.getCurrencyInstance(Locale.US)
221                     .format((product?.price)?.toDouble()).toString().toLowerCase()
222         }
223
224         if (!TextUtils.isEmpty(product!!.description)) {
225             val descText: Spanned = product!!.getParsedDescription(
226                     ContextCompat.getColorStateList(this, R.color.dribbble_links)!!,
227                     ContextCompat.getColor(this, R.color.dribbble_link_highlight))!!
228             if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
229                 (description as FabOverlapTextView).setText(product?.description)
230             } else {
231                 HtmlUtils.setTextWithNiceLinks(description as TextView, descText)
232             }
233         } else {
234             description.visibility = View.GONE
235         }
236
237         //Show related products
238         more.setOnClickListener({
239             val intent = Intent(this@DetailsActivity, RelatedProductsActivity::class.java)
240             val bundle = Bundle()
241             //Send whole product to new activity as bundle
242             bundle.putParcelable(RelatedProductsActivity.EXTRA_PRODUCT, product)
243             intent.putExtras(bundle)
244             startActivity(intent)
245         })
246
247         val nf = NumberFormat.getInstance()
248         //Like button
249         likeCount.text = res.getQuantityString(R.plurals.likes, 10000, nf.format(10000))
250         likeCount.setOnClickListener {
251             (likeCount.compoundDrawables[1] as AnimatedVectorDrawable).start()
252         }
253
254         //View button
255         viewCount.text = res.getQuantityString(R.plurals.views, 13000, nf.format(13000))
256         viewCount.setOnClickListener {
257             (viewCount.compoundDrawables[1] as AnimatedVectorDrawable).start()
258         }
259
260         //Share Button
261         share.setOnClickListener {
262             (share.compoundDrawables[1] as AnimatedVectorDrawable).start()
263             ShareProductTask(this, product!!).execute()
264         }
265
266         //Logic for altering quantity to be ordered
```

```
267         val quantity = product?.quantity!!.toLong()
268         productCount.text = "$userProductCount / $quantity"
269
270         //Add product quantity button
271         addProduct.setOnClickListener({
272             userProductCount++
273             productCount.text = "$userProductCount / $quantity"
274         })
275
276         //Less product quantity button
277         lessProduct.setOnClickListener({
278             //Check condition: If order count is less than # of products available...
279             userProductCount--
280             productCount.text = "$userProductCount / $quantity"
281         })
282
283         fab.setOnClickListener {
284             if (prefs.isLoggedIn) {
285                 fab.toggle()
286                 appendCart()
287             } else {
288                 val intent = Intent(this@DetailsActivity, AuthActivity::class.java)
289                 FabTransform.addExtras(intent,
290                         ContextCompat.getColor(this@DetailsActivity, R.color.button_accent),
291                         R.drawable.ic_heart_empty_56dp)
292                 val options = ActivityOptions.makeSceneTransitionAnimation(this@DetailsActivity, fab,
293                         getString(R.string.transition_dribbble_login))
294                 startActivityForResult(intent, RC_LOGIN_LIKE, options.toBundle())
295             }
296         }
297
298         if (product != null) {
299             shopName.text = "—${product?.shop?.toLowerCase()}"
300             GlideApp.with(this)
301                     .load(product?.logo)
302                     .apply(RequestOptions().diskCacheStrategy(DiskCacheStrategy.AUTOMATIC))
303                     .apply(RequestOptions().circleCrop())
304                     .apply(RequestOptions().placeholder(R.drawable.avatar_placeholder))
305                     .transition(withCrossFade())
306                     .apply(RequestOptions().override(largeAvatarSize.toInt(), largeAvatarSize.toInt()))
307                     .into(playerAvatar)
308
309             if (product?.timestamp != null) {
310                 shotTimeAgo.text = DateUtils.getRelativeTimeSpanString(product?.timestamp?.time!!,
311                         System.currentTimeMillis(),
312                         DateUtils.SECOND_IN_MILLIS).toString().toLowerCase()
313             } else {
314                 for (i in 0 until product?.brand!!.size) {
315                     //Append brand names to product's shop
316                     shotTimeAgo.text = product?.brand?.get(i)?.toLowerCase() + ", "
317                 }
318             }
319             val shopClick: View.OnClickListener = View.OnClickListener {
320                 val details = Intent(this@DetailsActivity, ShopsActivity::class.java)
321                 details.putExtra(ShopsActivity.EXTRA_SHOP_NAME, product?.shop)
322                 details.putExtra(ShopsActivity.EXTRA_SHOP_KEY, product?.shopID)
323                 val options = ActivityOptions.makeSceneTransitionAnimation(this@DetailsActivity,
324                         playerAvatar, getString(R.string.transition_player_avatar))
325                 startActivity(details, options.toBundle())
326             }
327             playerAvatar.setOnClickListener(shopClick)
328             shopName.setOnClickListener(shopClick)
329         } else {
330             shopName.visibility = View.GONE
331             playerAvatar.visibility = View.GONE
332             shotTimeAgo.visibility = View.GONE
333         }
```

```
334
335         //Bind Recyclerview's adapter to the recyclerview
336         commentAnimator = CommentAnimator()
337         commentsList.itemAnimator = commentAnimator
338         adapter = CommentsAdapter(shotDescription, commentFooter, 0L,
339                 resources.getInteger(R.integer.comment_expand_collapse_duration).toLong())
340         commentsList.adapter = adapter
341         commentsList.addItemDecoration(InsetDividerDecoration(
342                 CommentViewHolder::class.java,
343                 res.getDimensionPixelSize(R.dimen.divider_height),
344                 res.getDimensionPixelSize(R.dimen.keyline_1),
345                 ContextCompat.getColor(this, R.color.divider)))
346         loadComments()
347         if (prefs.isLoggedIn) checkLiked()
348
349     }
350 ---------------------------------------------------------------------------
351     //Load comments form database
352     private fun loadComments() {
353         if (data.isNotEmpty()) data.clear()
354         prefs.db.document(PhoenixUtils.COMMENT_REF)
355                 .collection(product?.id?.toString()!!)
356                 .whereEqualTo("id", product?.id)
357                 .get()
358                 .addOnCompleteListener({ task ->
359                     if (task.isSuccessful) {
360                         for (document in task.result.documents) {
361                             if (document.exists()) {
362                                 val comment: Comment? = document.toObject(Comment::class.java)
363                                 if (comment != null) {
364                                     data.add(comment)  //Add comment
365                                 }
366                             }
367                         }
368                         if (data.isNotEmpty()) {
369                             adapter.addComments(data)
370                         }
371                     }
372                 }).addOnFailureListener { e ->
373             Timber.d("Exception from comments: ${e.localizedMessage}")
374         }
375     }
376 ---------------------------------------------------------------------------
377     /** Post [Comment] to the database */
378     fun postComment(view: View) {
379         //Verify user login status
380         if (prefs.isLoggedIn) {
381             //Perform the following if user is logged in
382             if (TextUtils.isEmpty(enterComment?.text)) return
383             enterComment?.isEnabled = false
384             //Generate hash map for the comment itself
385             val comment = Comment(
386                     product?.id!!,
387                     enterComment?.text?.toString()!!,
388                     0,
389                     product?.url!!,
390                     prefs.customer,
391                     Date(System.currentTimeMillis())
392             )
393             //Push comment to db
394             prefs.db.document(PhoenixUtils.COMMENT_REF)
395                     .collection(product?.id?.toString()!!)
396                     .document()
397                     .set(comment.toHashMap(comment))
398                     .addOnFailureListener { e -> Timber.d("Exception on comment post: ${e.localizedMessage}") }
399                     .addOnCompleteListener { task ->
400                         if (task.isSuccessful) {
```

```
                    loadComments()
                    enterComment?.text?.clear()
                    enterComment?.isEnabled = true
                } else {
                    Timber.d("Exception on comment post: ${task.exception?.toString()}")
                }
            }

        } else {
            //Prompt user to login
            val login = Intent(this@DetailsActivity, AuthActivity::class.java)
            FabTransform.addExtras(login, ContextCompat.getColor(
                    this@DetailsActivity, R.color.background_light), R.drawable.ic_comment_add)
            val options = ActivityOptions.makeSceneTransitionAnimation(
                    this@DetailsActivity, postComment, getString(R.string.transition_dribbble_login))
            startActivityForResult(login, RC_LOGIN_COMMENT, options.toBundle())
        }
    }

    //Set status bar color from the most dominant color of the image
    //The color is picked from the top-most part of the image rather than the whole image
    private val shotLoadListener = object : RequestListener<Drawable?> {
        override fun onLoadFailed(e: GlideException?, model: Any?, target: Target<Drawable?>?,
                                  isFirstResource: Boolean): Boolean {
            return false
        }

        /** Shows loaded resource for image */
        override fun onResourceReady(resource: Drawable?, model: Any?, target: Target<Drawable?>?,
                                     dataSource: DataSource?, isFirstResource: Boolean): Boolean {
            val bitmap: Bitmap? = resource?.getBitmap()
            val twentyFourDip: Int = (TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP, 24.0f,
                    this@DetailsActivity.resources.displayMetrics)).toInt()
            if (bitmap != null) {
                Palette.from(bitmap)
                        .maximumColorCount(3)
                        .clearFilters()
                        .setRegion(0, 0, bitmap.width - 1, twentyFourDip)
                        .generate { palette ->
                            val isDark: Boolean
                            val lightness: Int = ColorUtils.isDark(palette)
                            isDark = if (lightness == ColorUtils.LIGHTNESS_UNKNOWN) {
                                ColorUtils.isDark(bitmap, bitmap.width / 2, 0)
                            } else {
                                lightness == ColorUtils.IS_DARK
                            }

                            if (!isDark) { // make back icon dark on light images
                                back.setColorFilter(ContextCompat.getColor(
                                        this@DetailsActivity, R.color.dark_icon))
                                more.setColorFilter(ContextCompat.getColor(
                                        this@DetailsActivity, R.color.dark_icon))
                            }

                            // color the status bar. Set a complementary dark color on L,
                            // light or dark color on M (with matching status bar icons)
                            var statusBarColor = window.statusBarColor
                            val topColor: Palette.Swatch? =
                                    ColorUtils.getMostPopulousSwatch(palette)
                            if (topColor != null &&
                                    (isDark || Build.VERSION.SDK_INT >= Build.VERSION_CODES.M)) {
                                statusBarColor = ColorUtils.scrimify(topColor.rgb,
                                        isDark, SCRIM_ADJUSTMENT)
                                // set a light status bar on M+
                                if (!isDark && Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                                    ViewUtils.setLightStatusBar(imageView)
                                }
```

```
                }

                if (statusBarColor != window.statusBarColor) {
                    imageView.setScrimColor(statusBarColor)
                    val statusBarColorAnim: ValueAnimator = ValueAnimator.ofArgb(
                            window.statusBarColor, statusBarColor)
                    statusBarColorAnim.addUpdateListener({ animator: ValueAnimator? ->
                        window.statusBarColor = animator!!.animatedValue as Int
                    })
                    statusBarColorAnim.duration = 1000L
                    statusBarColorAnim.interpolator = getFastOutSlowInInterpolator(this@DetailsActivity)
                    statusBarColorAnim.start()
                }
            }

        Palette.from(bitmap)
                .clearFilters()
                .generate { palette ->
                    // color the ripple on the image spacer (default is grey)
                    shotSpacer.background = ViewUtils.createRipple(palette, 0.25f, 0.5f,
                            ContextCompat.getColor(this@DetailsActivity, R.color.mid_grey),
                            true)
                    // slightly more opaque ripple on the pinned image to compensate
                    // for the scrim
                    imageView.foreground = ViewUtils.createRipple(palette, 0.3f, 0.6f,
                            ContextCompat.getColor(this@DetailsActivity, R.color.mid_grey),
                            true)
                }
      }
      imageView.background = null
      return false
    }

}

//Notify user of a bad url from intent data
private fun reportUrlError() {
    Snackbar.make(draggableFrame, R.string.bad_phoenix_url, Snackbar.LENGTH_SHORT).show()
    draggableFrame.postDelayed({ finishAfterTransition() }, 3000L)
}

/**
 * We run a transition to expand/collapse comments. Scrolling the RecyclerView while this is
 * running causes issues, so we consume touch events while the transition runs.
 */
private val touchEater: View.OnTouchListener = View.OnTouchListener { _, _ -> true }

//Append cart with product
private fun appendCart() {
    performingLike = true
    val map = hashMapOf(
            Pair<String, Any?>("id", product?.id),
            Pair<String, Any?>("name", product?.name),
            Pair<String, Any?>("image", product?.url),
            Pair<String, Any?>("price", product?.price),
            Pair<String, Any?>("quantity", userProductCount.toString()),
            Pair<String, Any?>("timestamp", Date(System.currentTimeMillis()))
    )

    if (fab.isChecked) {
        //ref: phoenix/orders/{key}/**
        //Add product to customers' orders
        prefs.db.document(PhoenixUtils.ORDER_REF)  //ref: phoenix/orders
                .collection(prefs.customer.key!!)
                .document()
                .set(map)
                .addOnCompleteListener { task ->
```

```
                    if (task.isSuccessful) {
                        Toast.makeText(applicationContext,
                                "${product?.name} has been added to your shopping cart",
                                Toast.LENGTH_SHORT).show()
                    } else {
                        fab.isChecked = false
                    }
                }
        } else {
            //ref: phoenix/orders/{key}/**
            //Remove from orders
            performingLike = false
            prefs.db.document(PhoenixUtils.ORDER_REF)
                    .collection(prefs.customer.key!!)
                    .whereEqualTo("name", product?.name)
                    .get()
                    .addOnCompleteListener { task ->
                        if (task.isSuccessful) {
                            if (task.result.documents[0].exists()) {
                                task.result.documents[0].reference.delete()
                                        .addOnCompleteListener { newTask ->
                                            fab.isChecked = !newTask.isSuccessful
                                        }
                            }
                        }
                    }
        }
    }

    //Listener for entering comment
    private val enterCommentFocus: View.OnFocusChangeListener = View.OnFocusChangeListener { _,
hasFocus ->
        // kick off an anim (via animated state list) on the post button. see
        // @drawable/ic_add_comment
        postComment?.isActivated = hasFocus

        // prevent content hovering over image when not pinned.
        if (hasFocus) {
            imageView.bringToFront()
            imageView.offset = -imageView.height
            imageView.isImmediatePin = true
        }
    }

    //Open link from url using GoogleCustomTabs
    private fun openLink(url: String?) {
        if (url != null && !TextUtils.isEmpty(url)) {
            CustomTabActivityHelper.openCustomTab(
                    this@DetailsActivity,
                    CustomTabsIntent.Builder()
                            .setToolbarColor(ContextCompat.getColor(this@DetailsActivity, R.color.
background_super_dark))
                            .addDefaultShareMenuItem()
                            .build(),
                    Uri.parse(url))
        }
    }

    //Check database to see whether the user has already added item to shopping cart
    private fun checkLiked() {
        //ref: phoenix/orders/{key}/**
        prefs.db.document(PhoenixUtils.ORDER_REF)
                .collection(prefs.customer.key!!)
                .get()
                .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        for (item in task.result.documents) {
```

```
                    if (item.exists()) {
                        val product = item.toObject(Product::class.java).withId<Product>(item.id)
                        if (product.name != null && product.name!! == product.name!!) {
                            TransitionManager.beginDelayedTransition(draggableFrame)
                            fab.isChecked = true
                        }
                    }
                }
            } else {
                if (BuildConfig.DEBUG) Timber.d(task.exception?.localizedMessage)
            }
        }
    }

    //Fling listener for recyclerview
    private val flingListener: RecyclerView.OnFlingListener = object : RecyclerView.OnFlingListener() {
        override fun onFling(velocityX: Int, velocityY: Int): Boolean {
            imageView.isImmediatePin = true
            return false
        }
    }

    //Scroll Listener for the recyclerview
    private val scrollListener: RecyclerView.OnScrollListener = object : RecyclerView.OnScrollListener() {
        override fun onScrolled(recyclerView: RecyclerView?, dx: Int, dy: Int) {
            val scrollY: Int = shotDescription.top
            imageView.offset = scrollY
            fab.setOffset(fabOffset + scrollY)
        }

        override fun onScrollStateChanged(recyclerView: RecyclerView?, newState: Int) {
            // as we animate the main image's elevation change when it 'pins' at it's min height
            // a fling can cause the title to go over the image before the animation has a chance to
            // run. In this case we short circuit the animation and just jump to state.
            imageView.isImmediatePin = newState == RecyclerView.SCROLL_STATE_SETTLING
        }
    }

    //Calculate the position of the FAB when user starts to scroll the content
    private fun calculateFabPosition() {
        // calculate 'natural' position i.e. with full height image. Store it for use when scrolling
        fabOffset = imageView.height + title.height - fab.height / 2
        fab.setOffset(fabOffset)

        // calculate min position i.e. pinned to the collapsed image when scrolled
        fab.setMinOffset(imageView.minimumHeight - fab.height / 2)
    }

    //Setup commenting field
    private fun setupCommenting() {
        allowComment = prefs.isLoggedIn
        if (allowComment && commentFooter == null) {
            commentFooter = layoutInflater.inflate(R.layout.dribbble_enter_comment,
                    commentsList, false)
            userAvatar = commentFooter?.findViewById(R.id.avatar) as ForegroundImageView
            enterComment = commentFooter?.findViewById(R.id.comment) as EditText
            postComment = commentFooter?.findViewById(R.id.post_comment) as ImageButton
            enterComment?.onFocusChangeListener = enterCommentFocus

            GlideApp.with(this)
                    .load(prefs.customer.photo)
                    .override(Target.SIZE_ORIGINAL, Target.SIZE_ORIGINAL)
                    .placeholder(R.drawable.avatar_placeholder)
                    .error(R.drawable.ic_player)
                    .diskCacheStrategy(DiskCacheStrategy.AUTOMATIC)
                    .circleCrop()
                    .transition(withCrossFade())
```

```
667                 .into(userAvatar)
668
669         } else if (!allowComment && commentFooter != null) {
670             adapter.removeCommentingFooter()
671             commentFooter = null
672             Toast.makeText(applicationContext, R.string.prospects_cant_post, Toast.LENGTH_SHORT).show(
    )
673         }
674
675     }
676 ____________________________________________________________________________
677     //Set result for any calling activity
678     private fun setResultAndFinish() {
679         val resultData = Intent()
680         resultData.putExtra(RESULT_EXTRA_SHOT_ID, product?.id)
681         setResult(Activity.RESULT_OK, resultData)
682         finishAfterTransition()
683     }
684 ____________________________________________________________________________
685     //Function to check whether the user commenting is the sender of that same comment
686     private fun isOP(): Boolean {
687         return prefs.isLoggedIn
688     }
689 ____________________________________________________________________________
690     //help with return transition
691     override fun onBackPressed() {
692         setResultAndFinish()
693     }
694 ____________________________________________________________________________
695     //helps with return transition
696     override fun onNavigateUp(): Boolean {
697         setResultAndFinish()
698         return true
699     }
700 ____________________________________________________________________________
701     override fun onPause() {
702         draggableFrame.removeListener(chromeFader)
703         super.onPause()
704     }
705 ____________________________________________________________________________
706     override fun onResume() {
707         super.onResume()
708         if (!performingLike) {
709             if (prefs.isLoggedIn) checkLiked()
710         }
711         draggableFrame.addListener(chromeFader)
712     }
713 ____________________________________________________________________________
714     @TargetApi(Build.VERSION_CODES.M)
715     override fun onProvideAssistContent(outContent: AssistContent) {
716         outContent.webUri = Uri.parse(product?.url)
717     }
718 ____________________________________________________________________________
719     //Handle results from intent actions
720     override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
721         when (requestCode) {
722             RC_LOGIN_LIKE -> {
723                 if (resultCode == RESULT_OK) {
724                     fab.isChecked = true
725                     appendCart()
726                     setupCommenting()
727                 }
728             }
729
730             RC_LOGIN_COMMENT -> {
731                 if (resultCode == RESULT_OK) {
732                     setupCommenting()
```

```
733             }
734         }
735     }
736 }
737
738 //Adapter for the recyclerview
739 internal inner class CommentsAdapter(private val description: View?,
740                                      @param:Nullable private var footer: View?,
741                                      commentCount: Long,
742                                      expandDuration: Long) :
743         RecyclerView.Adapter<RecyclerView.ViewHolder>() {
744
745     private val EXPAND: Int = 0x1
746     private val COLLAPSE: Int = 0x2
747     private val COMMENT_LIKE: Int = 0x3
748     private val REPLY: Int = 0x4
749
750     private var comments = ArrayList<Comment>(0)
751     private var expandCollapse: Transition = AutoTransition()
752
753     private var loading: Boolean = false
754     private var noComments: Boolean = false
755     private var expandedCommentPosition: Int = RecyclerView.NO_POSITION
756
757     init {
758         expandCollapse.duration = expandDuration
759         //expandCollapse.interpolator = getFastOutSlowInInterpolator(this@DetailsActivity)
760         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
761             expandCollapse.addListener(object : TransitionListenerAdapter() {
762                 override fun onTransitionStart(transition: Transition?) {
763                     commentsList.setOnTouchListener(touchEater)
764                 }
765
766                 override fun onTransitionEnd(transition: Transition?) {
767                     commentAnimator.setAnimateMoves(true)
768                     commentsList.setOnTouchListener(null)
769                 }
770             })
771         }
772         noComments = commentCount == 0L
773         loading = !noComments
774
775     }
776
777     override fun getItemViewType(position: Int): Int {
778         Log.d(TAG, position.toString())
779         if (position == 0) return R.layout.dribbble_shot_description
780         if (position == 1) {
781             if (loading) return R.layout.loading
782             if (noComments) return R.layout.dribbble_no_comments
783         }
784         if (footer != null) {
785             val footerPos = if (loading || noComments) 2 else comments.size + 1
786             if (position == footerPos) return R.layout.dribbble_enter_comment
787         }
788         return R.layout.dribbble_comment
789     }
790
791     override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
792         when (getItemViewType(position)) {
793             R.layout.dribbble_comment -> {
794                 if (comments.isNotEmpty()) {
795                     bindComment(holder as CommentViewHolder, getComment(position))
796                 }
797             }
798         }
799     }
```

```
800 
801     override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int, payloads: MutableList
    <Any>) {
802         if (holder is CommentViewHolder) {
803             bindPartialCommentChange(holder, position, payloads)
804         } else {
805             onBindViewHolder(holder, position)
806         }
807     }
808 
809     private fun bindPartialCommentChange(holder: CommentViewHolder, position: Int,
    partialChangePayloads: MutableList<Any>?) {
810         // for certain changes we don't need to rebind data, just update some view state
811         if ((partialChangePayloads!!.contains(EXPAND)
812                 || partialChangePayloads.contains(COLLAPSE))
813             || partialChangePayloads.contains(REPLY)) {
814             setExpanded(holder, position == expandedCommentPosition)
815         } else if (partialChangePayloads.contains(COMMENT_LIKE)) {
816             return // nothing to do
817         } else {
818             onBindViewHolder(holder, position)
819         }
820     }
821 
822     private fun bindComment(holder: CommentViewHolder, comment: Comment?) {
823         val position = holder.adapterPosition
824         val isExpanded = position == expandedCommentPosition
825         GlideApp.with(this@DetailsActivity)
826                 .load(comment?.user?.photo)
827                 .apply(RequestOptions.circleCropTransform())
828                 .apply(RequestOptions.placeholderOf(R.drawable.avatar_placeholder))
829                 .transition(withCrossFade(300))
830                 .into(holder.avatar)
831         holder.author.text = comment?.user?.name
832         holder.author.isOriginalPoster = isOP()
833         holder.timeAgo.text = if (comment?.timestamp == null)
834             DateUtils.getRelativeTimeSpanString(Date(System.currentTimeMillis()).time + 3L,
835                     System.currentTimeMillis(),
836                     DateUtils.SECOND_IN_MILLIS)
837                     .toString().toLowerCase()
838         else
839             DateUtils.getRelativeTimeSpanString(comment.timestamp!!.time,
840                     System.currentTimeMillis(),
841                     DateUtils.SECOND_IN_MILLIS)
842                     .toString().toLowerCase()
843         HtmlUtils.setTextWithNiceLinks(holder.commentBody, comment?.getParsedBody(holder
844                 .commentBody) as CharSequence?)
845         holder.likeHeart.isChecked = comment?.liked != null && comment.liked!!
846         if (prefs.isLoggedIn) {
847             holder.likeHeart.isEnabled = comment?.user?.id != prefs.customer.id
848         }
849         holder.likesCount.text = (comment?.likes_count)?.toString()
850         setExpanded(holder, isExpanded)
851     }
852 
853     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
854         when (viewType) {
855             R.layout.dribbble_shot_description -> {
856                 return SimpleViewHolder(description!!)
857             }
858             R.layout.dribbble_comment -> {
859                 return createCommentHolder(parent, viewType)
860             }
861             R.layout.loading, R.layout.dribbble_no_comments -> {
862                 return SimpleViewHolder(layoutInflater.inflate(viewType,
863                         parent, false))
864             }
```

```
865             R.layout.dribbble_enter_comment -> {
866                 return SimpleViewHolder(footer!!)
867             }
868             else -> throw IllegalArgumentException()
869         }
870
871     }
872 ---------------------------------------------------------------------------
873     @SuppressLint("SetTextI18n")
874     private fun createCommentHolder(parent: ViewGroup?, viewType: Int): CommentViewHolder {
875         val holder = CommentViewHolder(layoutInflater.inflate
876         (viewType, parent, false))
877
878         holder.itemView.setOnClickListener {
879             val position: Int = holder.adapterPosition
880             if (position == RecyclerView.NO_POSITION) return@setOnClickListener
881             val type = getItemViewType(position)
882             if (type != R.layout.dribbble_comment) return@setOnClickListener
883             if (comments.isEmpty()) return@setOnClickListener
884
885             val comment = getComment(position)
886             TransitionManager.beginDelayedTransition(commentsList, expandCollapse)
887             commentAnimator.setAnimateMoves(false)
888
889             // collapse any currently expanded items
890             if (RecyclerView.NO_POSITION != expandedCommentPosition) {
891                 notifyItemChanged(expandedCommentPosition, COLLAPSE)
892             }
893
894             // expand this item (if it wasn't already)
895             if (expandedCommentPosition != position) {
896                 expandedCommentPosition = position
897                 notifyItemChanged(position, EXPAND)
898                 if (comment.liked == null) {
899                     //todo: some action
900                 }
901                 if (enterComment != null && enterComment!!.hasFocus()) {
902                     enterComment?.clearFocus()
903                     ImeUtils.hideIme(enterComment!!)
904                 }
905                 holder.itemView.requestFocus()
906             } else {
907                 expandedCommentPosition = RecyclerView.NO_POSITION
908             }
909
910         }
911
912         holder.reply.setOnClickListener {
913             val position = holder.adapterPosition
914             if (position == RecyclerView.NO_POSITION) return@setOnClickListener
915
916             val comment = getComment(position)
917             enterComment?.setText("@ ${comment.user?.name} ")
918             enterComment?.setSelection(enterComment?.text!!.length)
919
920             // collapse the comment and scroll the reply box (in the footer) into view
921             expandedCommentPosition = RecyclerView.NO_POSITION
922             notifyItemChanged(position, REPLY)
923             holder.reply.jumpDrawablesToCurrentState()
924             enterComment?.requestFocus()
925             commentsList.smoothScrollToPosition(itemCount - 1)
926         }
927
928         holder.likeHeart.setOnClickListener {
929             if (prefs.isLoggedIn) {
930                 val position = holder.adapterPosition
931                 if (position == RecyclerView.NO_POSITION) return@setOnClickListener
```

```
932
933                 val comment = getComment(position)
934                 if (comment.liked != null && !comment.liked!!) {
935                     comment.liked = true
936                     comment.likes_count = comment.likes_count.plus(1)
937                     holder.likesCount.text = (comment.likes_count).toString()
938                     notifyItemChanged(position, COMMENT_LIKE)
939                     //Add map
940                     val map: HashMap<String, Any> = hashMapOf(
941                             Pair("id", System.currentTimeMillis()),
942                             Pair("customer", prefs.customer),
943                             Pair("comment", comment)
944                     )
945                     prefs.db.document(PhoenixUtils.LIKES_REF)   //like ref: phoenix/likes
946                             .collection(prefs.customer.key!!)   //.../{userKey}/
947                             .document()
948                             .set(map)
949                             .addOnFailureListener { exception ->
950                                 Toast.makeText(this@DetailsActivity,
951                                         exception.localizedMessage, Toast.LENGTH_SHORT).show()
952                             }.addOnCompleteListener { task ->
953                                 if (task.isSuccessful) {
954                                     Timber.d("Like added successfully")
955                                 } else {
956                                     Toast.makeText(this@DetailsActivity,
957                                             task.exception?.localizedMessage,
958                                             Toast.LENGTH_SHORT).show()
959                                 }
960                             }
961
962                 } else {
963                     comment.liked = false
964                     comment.likes_count--
965                     holder.likesCount.text = (comment.likes_count).toString()
966                     notifyItemChanged(position, COMMENT_LIKE)
967                     prefs.db.document(PhoenixUtils.LIKES_REF)   //like ref: phoenix/likes
968                             .collection(prefs.customer.key!!)   //.../{userKey}/
969                             .get()
970                             .addOnFailureListener { exception ->
971                                 Toast.makeText(this@DetailsActivity,
972                                         exception.localizedMessage, Toast.LENGTH_SHORT).show()
973                             }
974                             .addOnCompleteListener { task ->
975                                 if (task.isSuccessful) {
976                                     task.result.documents
977                                             .filter { it.exists() && it.contains(comment.toString()) }
978                                             .forEach {
979                                                 //delete item from database
980                                                 it.reference.delete()
981                                             }
982                                 } else {
983                                     Toast.makeText(this@DetailsActivity,
984                                             task.exception?.localizedMessage,
985                                             Toast.LENGTH_SHORT).show()
986                                 }
987                             }
988
989                 }
990             } else {
991                 holder.likeHeart.isChecked = false
992                 startActivityForResult(Intent(this@DetailsActivity,
993                         AuthActivity::class.java), RC_LOGIN_LIKE)
994             }
995         }
996
997         holder.likesCount.setOnClickListener(View.OnClickListener {
998             val position = holder.adapterPosition
```

```
            if (position == RecyclerView.NO_POSITION) return@OnClickListener

            //val comment = getComment(position)
            //todo: comments like calls to know who sent a comment
        })

        return holder
    }

    @NonNull
    private fun getComment(adapterPosition: Int): Comment {
        return comments[adapterPosition - 1] // description
    }

    override fun getItemCount(): Int {
        var count = 1 // description
        if (comments.isNotEmpty()) {
            count += comments.size
        } else {
            count++ // either loading or no comments
        }
        if (footer != null) count++
        return count
    }

    private fun setExpanded(holder: CommentViewHolder, isExpanded: Boolean) {
        holder.itemView.isActivated = isExpanded
        holder.reply.visibility = if (isExpanded && allowComment) View.VISIBLE else View.GONE
        holder.likeHeart.visibility = if (isExpanded) View.VISIBLE else View.GONE
        holder.likesCount.visibility = if (isExpanded) View.VISIBLE else View.GONE
    }

    private fun hideLoadingIndicator() {
        if (!loading) return
        loading = false
        notifyItemRemoved(1)
    }

    private fun hideNoCommentsIndicator() {
        if (!noComments) return
        noComments = false
        notifyItemRemoved(1)
    }

    fun addComments(newComments: List<Comment>) {
        if (newComments.isNotEmpty()) {
            comments.clear()   //Clear comments first
            hideLoadingIndicator()
            hideNoCommentsIndicator()
            val count = comments.size
            for (item in newComments) {
                var add = true
                for (i in 0 until count) {
                    val existingItem: Comment? = comments[i]
                    if (existingItem != null && existingItem == item) {
                        add = false
                        return
                    }
                }
                if (add) {
                    comments.add(item)  //Add item
                }
            }
            notifyItemRangeInserted(1, newComments.size)
        }
        Timber.d(newComments.toString())
    }
```

```
1066 ──────────────────────────────────────────────────────────────────────────────────────────────
1067         fun removeCommentingFooter() {
1068             if (footer == null) return
1069             val footerPos = itemCount - 1
1070             footer = null
1071             notifyItemRemoved(footerPos)
1072         }
1073
1074     }
1075
1076
1077     /**
1078      * A {@link RecyclerView.ItemAnimator} which allows disabling move animations. RecyclerView
1079      * does not like animating item height changes. {@link android.transition.ChangeBounds} allows
1080      * this but in order to simultaneously collapse one item and expand another, we need to run the
1081      * Transition on the entire RecyclerView. As such it attempts to move views around. This
1082      * custom item animator allows us to stop RecyclerView from trying to handle this for us while
1083      * the transition is running.
1084      */
1085     internal class CommentAnimator : SlideInItemAnimator() {
1086         private var animateMoves = false
1087 ──────────────────────────────────────────────────────────────────────────────────────────────
1088         fun setAnimateMoves(animateMoves: Boolean) {
1089             this.animateMoves = animateMoves
1090         }
1091 ──────────────────────────────────────────────────────────────────────────────────────────────
1092         override fun animateMove(
1093                 holder: RecyclerView.ViewHolder, fromX: Int, fromY: Int, toX: Int, toY: Int): Boolean {
1094             if (!animateMoves) {
1095                 dispatchMoveFinished(holder)
1096                 return false
1097             }
1098             return super.animateMove(holder, fromX, fromY, toX, toY)
1099         }
1100
1101     }
1102
1103     //Simple viewholder
1104     internal class SimpleViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView)
1105
1106     //Comment viewholder
1107     internal class CommentViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
1108
1109         var avatar: ImageView = itemView.findViewById(R.id.player_avatar)
1110         var author: AuthorTextView = itemView.findViewById(R.id.comment_author)
1111         var timeAgo: TextView = itemView.findViewById(R.id.comment_time_ago)
1112         var commentBody: TextView = itemView.findViewById(R.id.comment_text)
1113         var reply: ImageButton = itemView.findViewById(R.id.comment_reply)
1114         var likeHeart: CheckableImageButton = itemView.findViewById(R.id.comment_like)
1115         var likesCount: TextView = itemView.findViewById(R.id.comment_likes_count)
1116
1117
1118     }
1119
1120     companion object {
1121         private val TAG: String = DetailsActivity::class.java.simpleName
1122         const val EXTRA_SHOT = "EXTRA_SHOT"
1123         const val RESULT_EXTRA_SHOT_ID = "RESULT_EXTRA_SHOT_ID"
1124         private const val RC_LOGIN_LIKE = 0
1125         private const val RC_LOGIN_COMMENT = 1
1126         private const val SCRIM_ADJUSTMENT = 0.075f
1127     }
1128 }
1129
1130
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.Manifest
8  import android.annotation.TargetApi
9  import android.app.Activity
10 import android.app.AlertDialog
11 import android.content.Intent
12 import android.content.pm.PackageManager
13 import android.net.Uri
14 import android.os.Build
15 import android.os.Bundle
16 import android.support.v4.app.ActivityCompat
17 import android.support.v4.content.ContextCompat
18 import android.transition.TransitionManager
19 import android.view.View
20 import android.view.ViewGroup
21 import android.widget.*
22 import com.bumptech.glide.load.engine.DiskCacheStrategy
23 import com.bumptech.glide.load.resource.drawable.DrawableTransitionOptions
24 import com.bumptech.glide.request.RequestOptions
25 import com.bumptech.glide.request.target.Target
26 import com.google.android.gms.common.ConnectionResult
27 import com.google.android.gms.common.GooglePlayServicesNotAvailableException
28 import com.google.android.gms.common.GooglePlayServicesRepairableException
29 import com.google.android.gms.common.api.GoogleApiClient
30 import com.google.android.gms.location.places.GeoDataClient
31 import com.google.android.gms.location.places.PlaceDetectionClient
32 import com.google.android.gms.location.places.Places
33 import com.google.android.gms.location.places.ui.PlacePicker
34 import io.pergasus.BuildConfig
35 import io.pergasus.R
36 import io.pergasus.api.PhoenixClient
37 import io.pergasus.api.PhoenixUtils
38 import io.pergasus.ui.widget.CircularImageView
39 import io.pergasus.util.bindView
40 import io.pergasus.util.glide.GlideApp
41 import timber.log.Timber
42
43 /** User's profile screen */
44 class ProfileActivity : Activity(), GoogleApiClient.OnConnectionFailedListener, ActivityCompat.
    OnRequestPermissionsResultCallback {
45
46
47     private val container: ViewGroup by bindView(R.id.container)
48     private val emailContainer: ViewGroup by bindView(R.id.container_email)
49     private val addressContainer: ViewGroup by bindView(R.id.container_address)
50     private val favContainer: ViewGroup by bindView(R.id.container_favorite)
51     private val loading: ProgressBar by bindView(R.id.loading)
52     private val name: TextView by bindView(R.id.customer_name)
53     private val about: TextView by bindView(R.id.customer_bio)
54     private val avatar: CircularImageView by bindView(R.id.avatar)
55     private val email: TextView by bindView(R.id.customer_email)
56     private val address: TextView by bindView(R.id.customer_address)
57     private val key: TextView by bindView(R.id.customer_key)
58     private val save: Button by bindView(R.id.profile_save)
59
60     private lateinit var prefs: PhoenixClient
61     private lateinit var geoDataClient: GeoDataClient
62     private lateinit var apiClient: GoogleApiClient
63     private lateinit var placeDetectionClient: PlaceDetectionClient
64     private var imageUri: Uri? = null
65     private var placeAddressLat: String? = null
```

```
66    private var placeAddressLng: String? = null
67    private var addressPlace: String? = null
68
69    override fun onCreate(savedInstanceState: Bundle?) {
70        super.onCreate(savedInstanceState)
71        setContentView(R.layout.activity_profile)
72
73        prefs = PhoenixClient(this)
74
75        // Construct a GeoDataClient.
76        geoDataClient = Places.getGeoDataClient(this, null)
77        // Construct a PlaceDetectionClient.
78        placeDetectionClient = Places.getPlaceDetectionClient(this, null)
79
80        apiClient = GoogleApiClient.Builder(this@ProfileActivity)
81                .addApi(Places.GEO_DATA_API)
82                .addApi(Places.PLACE_DETECTION_API)
83                .build()
84        //Start using the Places API.
85
86        if (prefs.isLoggedIn) {
87            loadUser()
88        }
89    }
90
91    private fun loadUser() {
92        val customer = prefs.customer
93        name.text = customer.name
94        about.text = customer.info
95        email.text = prefs.auth.currentUser?.email
96
97        //Set address here
98        address.text = if (prefs.getPlace().isNullOrEmpty())
99            getString(R.string.no_address)
100       else prefs.getPlace()
101       key.text = customer.key
102
103       GlideApp.with(this)
104               .load(customer.photo)
105               .apply(RequestOptions().circleCrop())
106               .apply(RequestOptions().diskCacheStrategy(DiskCacheStrategy.AUTOMATIC))
107               .apply(RequestOptions().fallback(R.drawable.ic_player))
108               .apply(RequestOptions().error(R.drawable.avatar_placeholder))
109               .apply(RequestOptions().override(Target.SIZE_ORIGINAL, Target.SIZE_ORIGINAL))
110               .apply(RequestOptions().placeholder(R.drawable.avatar_placeholder))
111               .transition(DrawableTransitionOptions.withCrossFade())
112               .into(avatar)
113
114       //Edit content
115       about.setOnClickListener {
116           val builder = AlertDialog.Builder(this@ProfileActivity)
117           val view: View = layoutInflater.inflate(R.layout.edit_profile, null, false)
118           val textField = view.findViewById<EditText>(R.id.edt_field)
119           textField.setText(about.text.toString())
120           textField.selectAll()
121           builder.setView(view)
122           builder.setPositiveButton("Done") { p0, _ ->
123               val s = textField.text.toString()
124               p0?.dismiss()
125               about.text = s
126               save.isEnabled = true
127           }.setNegativeButton("Cancel") { p0, _ -> p0?.cancel() }
128           builder.show()
129       }
130       name.setOnClickListener {
131           val builder = AlertDialog.Builder(this@ProfileActivity)
132           val view: View = layoutInflater.inflate(R.layout.edit_profile, null, false)
```

```
133             val textField = view.findViewById<EditText>(R.id.edt_field)
134             textField.setText(name.text.toString())
135             textField.selectAll()
136             builder.setView(view)
137             builder.setPositiveButton("Done") { p0, _ ->
138                 val s = textField.text.toString()
139                 p0?.dismiss()
140                 name.text = s
141                 save.isEnabled = true
142             }.setNegativeButton("Cancel") { p0, _ -> p0?.cancel() }
143             builder.show()
144         }
145         avatar.setOnClickListener({ pickImage() })
146         address.setOnClickListener({ pickPlace() })
147         save.setOnClickListener {
148             showLoading()  //inflate "loading" layout
149             if (imageUri != null) {
150                 uploadImage()
151             } else {
152                 if (prefs.isConnected) {
153                     updateData()
154                 } else {
155                     hideLoading()
156                     Toast.makeText(this, "You are not connected to the internet",
157                             Toast.LENGTH_LONG).show()
158                 }
159             }
160         }
161     }
162 ----------------------------------------------------------------------------------------------------
163     @TargetApi(Build.VERSION_CODES.M)
164     private fun pickPlace() {
165         if (ContextCompat.checkSelfPermission(this@ProfileActivity,
166                 Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.
    PERMISSION_GRANTED) {
167             // Construct an intent for the place picker
168             try {
169                 val intentBuilder = PlacePicker.IntentBuilder()
170                 val intent = intentBuilder.build(this@ProfileActivity)
171                 // Start the intent by requesting a result,
172                 // identified by a request code.
173                 startActivityForResult(intent, REQUEST_PLACE_PICKER)
174             } catch (e: GooglePlayServicesRepairableException) {
175                 //show exception
176                 Timber.e(e, e.localizedMessage)
177                 Toast.makeText(this, "An error occurred : ${e.localizedMessage}", Toast.LENGTH_LONG).show(
    )
178             } catch (e: GooglePlayServicesNotAvailableException) {
179                 //show exception
180                 Timber.e(e, e.localizedMessage)
181                 Toast.makeText(this, "An error occurred : ${e.localizedMessage}", Toast.LENGTH_LONG).show(
    )
182             }
183         } else {
184             requestPermissions(arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
185                     LOCATION_REQUEST_CODE)
186         }
187     }
188 ----------------------------------------------------------------------------------------------------
189     private fun showLoading() {
190         TransitionManager.beginDelayedTransition(container)
191         loading.visibility = ViewGroup.VISIBLE
192         emailContainer.visibility = ViewGroup.GONE
193         save.visibility = ViewGroup.GONE
194         addressContainer.visibility = ViewGroup.GONE
195         favContainer.visibility = ViewGroup.GONE
196     }
```

```
197
198     private fun hideLoading() {
199         TransitionManager.beginDelayedTransition(container)
200         loading.visibility = ViewGroup.GONE
201         emailContainer.visibility = ViewGroup.VISIBLE
202         save.visibility = ViewGroup.VISIBLE
203         addressContainer.visibility = ViewGroup.VISIBLE
204         favContainer.visibility = ViewGroup.VISIBLE
205     }
206
207     private fun updateData() {
208         val customer = prefs.customer
209         val hashMap = hashMapOf(Pair<String, Any?>("name", name.text.toString()),
210                 Pair<String, Any?>("about", about.text.toString()),
211                 Pair<String, Any?>("addressLat", if (placeAddressLat == null) customer.addressLat
212                 else placeAddressLat),
213                 Pair<String, Any?>("addressLng", if (placeAddressLng == null) customer.addressLng
214                 else placeAddressLng)
215         )
216         //Update data
217         prefs.db.document("phoenix/mobile")
218                 .collection(PhoenixUtils.CUSTOMER_REF)
219                 .document(prefs.customer.key!!)
220                 .update(hashMap)
221                 .addOnCompleteListener { task ->
222                     if (task.isSuccessful) {
223                         prefs.setName(name.text.toString())
224                         prefs.setAddress(addressLat = placeAddressLat!!, addressLng = placeAddressLng!!)
225                         prefs.setPlace(addressPlace)
226                         prefs.setInfo(about.text.toString())
227                         hideLoading()  //hide "loading" layout
228                         save.isEnabled = false
229                     }
230                 }
231     }
232
233     private fun uploadImage() {
234         if (imageUri == null) {
235             updateData()
236         } else {
237             prefs.storage.child(prefs.customer.key + ".jpg").putFile(imageUri!!)
238                     .addOnSuccessListener { taskSnapshot ->
239                         if (taskSnapshot.task.isComplete) {
240                             val downloadUrl = taskSnapshot.downloadUrl
241                             if (downloadUrl != null) {
242                                 prefs.setImage(downloadUrl)
243                             }
244                             if (prefs.isConnected) {
245                                 prefs.db.document("phoenix/mobile/${PhoenixUtils.CUSTOMER_REF}/${prefs.
    customer.key}")
246                                         .update("photo", downloadUrl.toString())
247                                         .addOnCompleteListener { task ->
248                                             if (task.isSuccessful) {
249                                                 updateData()
250                                             } else {
251                                                 hideLoading()
252                                                 Toast.makeText(this, task.exception?.localizedMessage,
253                                                         Toast.LENGTH_LONG).show()
254                                             }
255                                         }
256                                         .addOnFailureListener { exception ->
257                                             hideLoading()
258                                             Toast.makeText(this, exception.localizedMessage, Toast.LENGTH_LONG).
    show()
259                                         }
260
261                             } else {
```

```
                        hideLoading()
                        Toast.makeText(this,
                                "Upload will be suspended until you are connected to the internet",
                                Toast.LENGTH_LONG).show()
                    }
                } else {
                    hideLoading()
                    //Upload was not successful
                    Toast.makeText(this, taskSnapshot.task.exception?.localizedMessage,
                            Toast.LENGTH_LONG).show()
                }
            }
            .addOnFailureListener { exception ->
                hideLoading()
                Toast.makeText(this, exception.localizedMessage, Toast.LENGTH_LONG).show()
            }
    }
}

private fun pickImage() {
    val galleryIntent = Intent()
    galleryIntent.type = "image/*"
    galleryIntent.action = Intent.ACTION_GET_CONTENT
    startActivityForResult(Intent.createChooser(galleryIntent, "Select picture"), IMAGE_REQUEST_CODE)
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    when (requestCode) {
        IMAGE_REQUEST_CODE -> when (resultCode) {
            RESULT_OK -> {
                //Image picked
                val uri = data?.data
                if (uri != null) {
                    //Load image into resource
                    GlideApp.with(this)
                            .load(uri)
                            .apply(RequestOptions().circleCrop())
                            .apply(RequestOptions().error(R.drawable.avatar_placeholder))
                            .apply(RequestOptions().fallback(R.drawable.ic_player))
                            .apply(RequestOptions().override(Target.SIZE_ORIGINAL, Target.SIZE_ORIGINAL))
                            .apply(RequestOptions().placeholder(R.drawable.avatar_placeholder))
                            .transition(DrawableTransitionOptions.withCrossFade())
                            .into(avatar)

                    save.isEnabled = true  //Enable the save button
                    imageUri = uri
                }
            }
            RESULT_CANCELED, RESULT_FIRST_USER -> {/*No image selected by the user*/
            }
        }
        REQUEST_PLACE_PICKER -> when (resultCode) {
            RESULT_OK -> {
                //Place found
                // The user has selected a place. Extract the name and address.
                val place = PlacePicker.getPlace(this, data)
                val latLng = place.latLng

                //Set params
                placeAddressLat = latLng.latitude.toString()
                placeAddressLng = latLng.longitude.toString()

                //val name: CharSequence = place.name
                val address: CharSequence = place.address
                if (BuildConfig.DEBUG) {
                    Timber.d("User place found to be : $place")
                }
```

```
329                 addressPlace = address.toString()
330                 this.address.text = addressPlace
331                 save.isEnabled = true
332             }
333             RESULT_CANCELED, RESULT_FIRST_USER -> {/*No place selected by the user*/
334             }
335         }
336         else -> super.onActivityResult(requestCode, resultCode, data)
337     }
338 }
339
340 ____________________________________________________________________________________
341 override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>,
    grantResults: IntArray) {
342     if (requestCode == REQUEST_STORAGE_CODE) {
343         if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
344             pickImage()
345         } else {
346             val rationale = ActivityCompat.shouldShowRequestPermissionRationale(this@ProfileActivity,
    Manifest
347                     .permission.WRITE_EXTERNAL_STORAGE)
348             if (rationale) {
349                 Toast.makeText(this, "You need to allow this permission first", Toast.LENGTH_SHORT).show(
    )
350                 return
351             }
352         }
353     } else if (requestCode == LOCATION_REQUEST_CODE) {
354         if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
355             pickPlace()
356         } else {
357             val rationale = ActivityCompat.shouldShowRequestPermissionRationale(this@ProfileActivity,
    Manifest
358                     .permission.ACCESS_FINE_LOCATION)
359             if (rationale) {
360                 Toast.makeText(this, "You need to allow this permission first", Toast.LENGTH_SHORT).show(
    )
361                 return
362             }
363         }
364     }
365 }
366 ____________________________________________________________________________________
367 override fun onConnectionFailed(p0: ConnectionResult) {
368     if (BuildConfig.DEBUG) {
369         Timber.d(p0.errorMessage)
370     } else {
371         Toast.makeText(this, p0.errorMessage, Toast.LENGTH_SHORT).show()
372     }
373 }
374 ____________________________________________________________________________________
375 override fun onBackPressed() {
376     if (save.isEnabled) {
377         val builder = AlertDialog.Builder(this@ProfileActivity)
378         builder.setMessage(getString(R.string.save_changes_prompt))
379         builder.setCancelable(false)
380         builder.setPositiveButton("Discard", { dialogInterface, _ ->
381             dialogInterface.cancel()
382             setResult(RESULT_CANCELED)
383             super.onBackPressed()
384         })
385         builder.setNegativeButton("Save", { dialogInterface, _ ->
386             save.performClick()
387             dialogInterface.dismiss()
388             setResult(RESULT_OK)
389         })
390         builder.show()
```

```
391         } else {
392             setResult(RESULT_OK)
393             super.onBackPressed()
394         }
395     }
396 
397     override fun onNavigateUp(): Boolean {
398         setResult(RESULT_OK)
399         return true
400     }
401 
402     companion object {
403         private const val REQUEST_STORAGE_CODE = 11
404         private const val IMAGE_REQUEST_CODE = 12
405         private const val LOCATION_REQUEST_CODE = 13
406         private const val REQUEST_PLACE_PICKER = 14
407     }
408 }
409 
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.app.Activity
8  import android.content.Intent
9  import android.os.Bundle
10 import android.widget.TextView
11 import android.widget.ViewFlipper
12 import io.pergasus.R
13 import io.pergasus.util.bindView
14
15 /** Status screen for the order transaction */
16 class SuccessActivity : Activity() {
17     private val flipper: ViewFlipper by bindView(R.id.flipper_success)
18     private val shopMore: TextView by bindView(R.id.text_shop_more)
19     private val trackOrder: TextView by bindView(R.id.text_track_order)
20     private val retry: TextView by bindView(R.id.text_try_again)
21
22     override fun onCreate(savedInstanceState: Bundle?) {
23         super.onCreate(savedInstanceState)
24         setContentView(R.layout.activity_success)
25
26         //Get Intent extras
27         val page = intent.getIntExtra(EXTRA_PAGE, PAGE_SUCCESS)
28         flipper.displayedChild = page
29
30         //Navigate to the HomeScreen
31         shopMore.setOnClickListener({
32             startActivity(Intent(this@SuccessActivity, HomeActivity::class.java))
33             setResultAndFinish()
34         })
35
36         //Track current order
37         trackOrder.setOnClickListener({
38             startActivity(Intent(this@SuccessActivity, TrackingActivity::class.java))
39             setResultAndFinish()
40         })
41
42         //Get back to the OrderActivity
43         retry.setOnClickListener({
44             setResult(RESULT_CANCELED)
45             finishAfterTransition()
46         })
47     }
48
49     override fun onNavigateUp(): Boolean {
50         setResultAndFinish()
51         return true
52     }
53
54     override fun onBackPressed() {
55         setResultAndFinish()
56     }
57
58     private fun setResultAndFinish() {
59         setResult(RESULT_OK)
60         finishAfterTransition()
61     }
62
63     companion object {
64         val EXTRA_PAGE = SuccessActivity::class.java.name + ".PAGE"
65         val PAGE_SUCCESS = 0
66         val PAGE_FAIL = 1
```

```
67     }
68 }
69
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.app.Activity
8  import android.content.Intent
9  import android.graphics.Color
10 import android.os.Bundle
11 import android.widget.Toast
12 import io.pergasus.R
13 import io.pergasus.api.PhoenixApplication
14 import io.pergasus.util.onboarding.PaperOnboardingEngine
15 import io.pergasus.util.onboarding.PaperOnboardingOnChangeListener
16 import io.pergasus.util.onboarding.PaperOnboardingOnRightOutListener
17 import io.pergasus.util.onboarding.PaperOnboardingPage
18 import java.util.*
19
20
21 /**
22  * A miniature tutorial screen for new installation
23  */
24 class WelcomeActivity : Activity() {
25
26     private lateinit var clientState: PhoenixApplication.Companion.PhoenixClientState
27 ____________________________________________________________________________________
28     override fun onCreate(savedInstanceState: Bundle?) {
29         super.onCreate(savedInstanceState)
30         setContentView(R.layout.onboarding_main_layout)
31
32         clientState = PhoenixApplication.Companion.PhoenixClientState(this)
33
34         //Setup PaperOnBoardingEngine
35         val engine = PaperOnboardingEngine(findViewById(R.id.onboardingRootView), getDataForOnBoarding
    (), applicationContext)
36         engine.setOnRightOutListener(object : PaperOnboardingOnRightOutListener {
37             override fun onRightOut() {
38                 // Probably here will be your exit action
39                 Toast.makeText(applicationContext, "Press the back button to get started",
40                         Toast.LENGTH_SHORT).show()
41             }
42         })
43     }
44 ____________________________________________________________________________________
45     private fun getDataForOnBoarding(): ArrayList<PaperOnboardingPage> {
46         // prepare data
47         /*val scr1 = PaperOnboardingPage(getString(R.string.ob_header1), getString(R.string.ob_desc1),
48 ____________        Color.parseColor("#678FB4"), R.drawable.hotels, R.drawable.key)*/
49         val scr2 = PaperOnboardingPage(getString(R.string.ob_header2), getString(R.string.ob_desc2),
50                 Color.parseColor("#678FB4"), R.drawable.banks, R.drawable.wallet)
51         val scr3 = PaperOnboardingPage(getString(R.string.ob_header3), getString(R.string.ob_desc3),
52                 Color.parseColor("#9B90BC"), R.drawable.stores, R.drawable.shopping_cart)
53
54         val elements = ArrayList<PaperOnboardingPage>(0)
55 //      elements.add(scr1)
56         elements.add(scr2)
57         elements.add(scr3)
58         return elements
59     }
60 ____________________________________________________________________________________
61     override fun onBackPressed() {
62         clientState.setAppInstalledState(true)
63         startActivity(Intent(this@WelcomeActivity, HomeActivity::class.java))
64         finish()
65     }
```

```
66
67  }
68
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui;
6
7  import android.animation.TypeEvaluator;
8
9  import com.google.android.gms.maps.model.LatLng;
10
11 public class RouteEvaluator implements TypeEvaluator<LatLng> {
12     @Override
13     public LatLng evaluate(float t, LatLng startPoint, LatLng endPoint) {
14         double lat = startPoint.latitude + t * (endPoint.latitude - startPoint.latitude);
15         double lng = startPoint.longitude + t * (endPoint.longitude - startPoint.longitude);
16         return new LatLng(lat, lng);
17     }
18 }
19
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.Manifest.permission
8  import android.annotation.SuppressLint
9  import android.annotation.TargetApi
10 import android.content.pm.PackageManager
11 import android.location.Location
12 import android.os.Build.VERSION
13 import android.os.Build.VERSION_CODES
14 import android.os.Bundle
15 import android.os.Handler
16 import android.support.annotation.RequiresApi
17 import android.support.v4.content.ContextCompat
18 import android.support.v7.app.AppCompatActivity
19 import android.widget.Toast
20 import com.google.android.gms.common.ConnectionResult
21 import com.google.android.gms.common.GooglePlayServicesUtil
22 import com.google.android.gms.common.api.GoogleApiClient
23 import com.google.android.gms.common.api.GoogleApiClient.ConnectionCallbacks
24 import com.google.android.gms.common.api.GoogleApiClient.OnConnectionFailedListener
25 import com.google.android.gms.location.LocationListener
26 import com.google.android.gms.location.LocationRequest
27 import com.google.android.gms.location.LocationServices
28 import com.google.android.gms.location.places.Places
29 import com.google.android.gms.maps.CameraUpdateFactory
30 import com.google.android.gms.maps.GoogleMap
31 import com.google.android.gms.maps.OnMapReadyCallback
32 import com.google.android.gms.maps.model.LatLng
33 import com.google.android.gms.maps.model.LatLngBounds
34 import com.google.android.gms.maps.model.MapStyleOptions
35 import io.pergasus.R
36 import io.pergasus.api.PhoenixClient
37 import io.pergasus.api.PhoenixUtils
38 import io.pergasus.data.Purchase
39 import io.pergasus.util.maputil.RouteOverlayView.AnimType
40 import io.pergasus.util.maputil.TrailSupportMapFragment
41 import java.util.*
42
43
44 /**
45  * Track delivery of Goods in real-time
46  */
47 class TrackingActivity : AppCompatActivity(), OnMapReadyCallback, LocationListener,
    OnConnectionFailedListener, ConnectionCallbacks {
48
49     private var shouldPromptForPermission: Boolean = false
50
51     private var map: GoogleMap? = null
52     private var client: PhoenixClient? = null
53     private var apiClient: GoogleApiClient? = null
54     private var locationRequest: LocationRequest? = null
55     private var lastLocation: Location? = null
56     private var purchase: Purchase? = null
57     private var mapFragment: TrailSupportMapFragment? = null
58
59     override fun onCreate(savedInstanceState: Bundle?) {
60         super.onCreate(savedInstanceState)
61         setContentView(R.layout.activity_tracking)
62
63         //Init shared preferences
64         client = PhoenixClient(this)
65
```

```
66        //Setup permission primer
67        shouldPromptForPermission = hasNoPermission()
68
69        //Request permission once the activity starts
70        if (shouldPromptForPermission) {
71            if (VERSION.SDK_INT >= VERSION_CODES.M) {
72                requestLocationPermission()
73            }
74        }
75
76        //Get map fragment
77        mapFragment = supportFragmentManager.findFragmentById(R.id.map) as
      TrailSupportMapFragment
78        //Sync map
79        mapFragment!!.getMapAsync(this)
80
81        //Get intent data
82        val intent = intent
83        if (intent.hasExtra(EXTRA_PURCHASE)) {
84            purchase = intent.getParcelableExtra(EXTRA_PURCHASE)
85            showMessage("You are currently tracking Purchase: " + purchase!!.key!!)
86        }
87    }
88
89    private fun hasNoPermission(): Boolean {
90        //Checks whether user has enabled permission to access their location
91        return ContextCompat.checkSelfPermission(this@TrackingActivity,
92                permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
      ContextCompat.checkSelfPermission(this@TrackingActivity,
93                permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED
94    }
95
96    @Synchronized
97    protected fun buildGoogleApiClient() {
98        apiClient = GoogleApiClient.Builder(this)
99                .addApi(LocationServices.API)
100               .addApi(Places.GEO_DATA_API)
101               .addApi(Places.PLACE_DETECTION_API)
102               .addOnConnectionFailedListener(this)
103               .addConnectionCallbacks(this)
104               .build()
105       apiClient!!.connect()
106   }
107
108   private fun createLocationRequest() {
109       locationRequest = LocationRequest.create()
110       locationRequest!!.interval = UPDATE_INTERVAL
111       locationRequest!!.fastestInterval = FASTEST_INTERVAL
112       locationRequest!!.smallestDisplacement = DISPLACEMENT
113       locationRequest!!.priority = LocationRequest.PRIORITY_HIGH_ACCURACY
114   }
115
116   @SuppressLint("MissingPermission")
117   private fun startLocationUpdates() {
118       if (shouldPromptForPermission) return
119       if (apiClient != null && locationRequest != null) {
120           LocationServices.FusedLocationApi.requestLocationUpdates(apiClient, locationRequest, this)
121       } else {
122           showMessage("Google API client and Location request is not properly setup")
123       }
124   }
125
126   //Checks the availability of Google Play Services on the current device
127   private fun checkPlayService(): Boolean {
128       val i = GooglePlayServicesUtil.isGooglePlayServicesAvailable(applicationContext)
129       if (i != ConnectionResult.SUCCESS) {
130           if (GooglePlayServicesUtil.isUserRecoverableError(i)) {
```

```
131             GooglePlayServicesUtil.getErrorDialog(i, this, RC_PLAY_SERVICES_RESOLUTION).show()
132         } else {
133             showMessage("This device is not supported")
134             finishAfterTransition()
135         }
136         return false
137     }
138
139     return true
140 }
141 ----------
142 private fun showMessage(message: CharSequence?) {
143     Toast.makeText(applicationContext, message, Toast.LENGTH_LONG).show()
144 }
145 ----------
146 override fun onStart() {
147     super.onStart()
148     //Connect API Client
149     if (apiClient != null) apiClient!!.connect()
150 }
151 ----------
152 override fun onStop() {
153     //Disconnect API client
154     if (apiClient != null && apiClient!!.isConnected) apiClient!!.disconnect()
155     super.onStop()
156 }
157 ----------
158 override fun onMapReady(googleMap: GoogleMap) {
159     map = googleMap
160     map!!.uiSettings.isRotateGesturesEnabled = true
161     map!!.uiSettings.isTiltGesturesEnabled = false
162     map!!.setMaxZoomPreference(18f)
163
164
165     //Set custom map style
166     map!!.setMapStyle(MapStyleOptions.loadRawResourceStyle(applicationContext,
167         R.raw.zuber_map_style))
168
169     //Callback for map loaded state
170     map!!.setOnMapLoadedCallback {
171         //Get user's current location
172         if (lastLocation == null) {
173             if (shouldPromptForPermission) {
174                 if (VERSION.SDK_INT >= VERSION_CODES.M) {
175                     requestLocationPermission()
176                     shouldPromptForPermission = true
177                 }
178             }
179         } else if (checkPlayService()) {
180             buildGoogleApiClient()
181             createLocationRequest()
182             displayLocation()
183         }
184     }
185
186 }
187 ----------
188 @SuppressLint("MissingPermission")
189 private fun displayLocation() {
190     if (shouldPromptForPermission) {
191         if (VERSION.SDK_INT >= VERSION_CODES.M) {
192             requestLocationPermission()
193         }
194     } else {
195         //Check user's login state. It may have changed before this activity is created
196         if (client!!.isLoggedIn) {
197             if (apiClient != null) {
```

```
198                 if (lastLocation == null) {
199                     lastLocation = LocationServices.FusedLocationApi
200                             .getLastLocation(apiClient)
201                 }
202
203                 //Obtain user's location from last known location
204                 val userLocation = LatLng(lastLocation!!.latitude, lastLocation!!.longitude)
205                 //Get the Mall's current location
206                 val mallGeoPoint = PhoenixUtils.MALL_GEO_POINT
207
208                 //Create Bounds for location
209                 val builder = LatLngBounds.Builder()
210                 //Add user's location
211                 builder.include(userLocation)
212                 //Add mall's location
213                 builder.include(mallGeoPoint)
214                 //Build bounds
215                 val bounds = builder.build()
216                 //Create camera updater
217                 val cameraUpdate = CameraUpdateFactory.newLatLngBounds(bounds, 200)
218
219                 //Apply animation to map
220                 map!!.moveCamera(cameraUpdate)
221                 map!!.animateCamera(CameraUpdateFactory.zoomTo(15f), 2000, null)
222
223                 //Update camera movement with fragment
224                 map!!.setOnCameraMoveListener { mapFragment!!.onCameraMove(map) }
225
226                 //Add locations as list
227                 val routes = ArrayList<LatLng>(0)
228                 routes.add(userLocation)
229                 routes.add(mallGeoPoint)
230
231                 //Finally, start animation: Can be replaces with startAnimation(routes);
232                 Handler().postDelayed({ mapFragment?.setUpPath(routes, map, AnimType.ARC) }, 1000)
233
234             } else
235                 showMessage("Google API client cannot be created")
236
237         }
238     }
239 }
240 ---------------------------------------------------------------------------------------------
241 private fun startAnimation(routes: List<LatLng>) {
242     if (map == null) {
243         showMessage("Map is not ready")
244     } else {
245         MapAnimator.getInstance().animateRoute(map, routes)
246     }
247 }
248 ---------------------------------------------------------------------------------------------
249 @TargetApi(VERSION_CODES.M)
250 override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String>, grantResults:
    IntArray) {
251     if (requestCode == RC_LOCATION) {
252         if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
253             shouldPromptForPermission = false
254             if (checkPlayService()) {
255                 buildGoogleApiClient()
256                 createLocationRequest()
257                 displayLocation()
258             }
259
260         } else if (shouldShowRequestPermissionRationale(permission.ACCESS_FINE_LOCATION)) {
261             showMessage("Please accept permission to access your location")
262             requestLocationPermission()
263         }
```

```
264         }
265     }
266 
267     @RequiresApi(api = VERSION_CODES.M)
268     private fun requestLocationPermission() {
269         requestPermissions(arrayOf(permission.ACCESS_FINE_LOCATION, permission
270                 .ACCESS_COARSE_LOCATION), RC_LOCATION)
271     }
272 
273     override fun onLocationChanged(location: Location) {
274         //Set user's current location
275         lastLocation = location
276         displayLocation()
277     }
278 
279     override fun onConnectionFailed(connectionResult: ConnectionResult) {
280         //Show message to user
281         showMessage(connectionResult.errorMessage)
282     }
283 
284     override fun onConnected(bundle: Bundle?) {
285         //Update location and display routes
286         displayLocation()
287         startLocationUpdates()
288     }
289 
290     override fun onConnectionSuspended(i: Int) {
291         //Reconnect API client
292         apiClient!!.connect()
293     }
294 
295     companion object {
296         //Constants
297         const val EXTRA_PURCHASE = "EXTRA_PURCHASE"
298         const val RC_PLAY_SERVICES_RESOLUTION = 123
299         const val RC_LOCATION = 124
300         const val UPDATE_INTERVAL = 1000L   //1.0 sec update interval
301         const val FASTEST_INTERVAL = 5000L   //1.0 sec update interval
302         const val DISPLACEMENT = 10.0f   //1.0 sec update interval
303     }
304 }
305 
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.app.Activity
8  import android.content.Intent
9  import android.os.Bundle
10 import android.support.v7.widget.LinearLayoutManager
11 import android.support.v7.widget.RecyclerView
12 import android.text.format.DateUtils
13 import android.transition.TransitionManager
14 import android.view.View
15 import android.view.ViewGroup
16 import android.widget.Button
17 import android.widget.TextView
18 import android.widget.Toast
19 import com.afollestad.materialdialogs.MaterialDialog
20 import com.bumptech.glide.load.engine.DiskCacheStrategy
21 import com.bumptech.glide.load.resource.bitmap.BitmapTransitionOptions.withCrossFade
22 import io.pergasus.BuildConfig
23 import io.pergasus.R
24 import io.pergasus.api.OrderDataManager
25 import io.pergasus.api.PhoenixClient
26 import io.pergasus.api.PhoenixUtils
27 import io.pergasus.data.Purchase
28 import io.pergasus.ui.recyclerview.SlideInItemAnimator
29 import io.pergasus.ui.widget.CircularImageView
30 import io.pergasus.util.bindView
31 import io.pergasus.util.glide.GlideApp
32 import timber.log.Timber
33 import java.text.NumberFormat
34 import java.util.*
35
36 /**
37  * Find user's orders in the database
38  * */
39 class LiveOrdersActivity : Activity() {
40     private val grid: RecyclerView by bindView(R.id.grid)
41     private val container: ViewGroup by bindView(R.id.container)
42     private val noOrders: TextView by bindView(R.id.no_orders)
43
44     private lateinit var prefs: PhoenixClient
45     private lateinit var dataManager: OrderDataManager
46     private lateinit var adapter: OrdersAdapter
47
48 ____________________________________________________________________________________
49     override fun onCreate(savedInstanceState: Bundle?) {
50         super.onCreate(savedInstanceState)
51         setContentView(R.layout.activity_live_orders)
52
53         prefs = PhoenixClient(this)
54
55         dataManager = object : OrderDataManager(this@LiveOrdersActivity) {
56             override fun onDataLoaded(data: List<Purchase>) {
57                 adapter.addLiveOrder(data)
58                 checkEmptyState()
59             }
60         }
61
62         //Setup recyclerview
63         adapter = OrdersAdapter()
64         grid.adapter = adapter
65         grid.layoutManager = LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false)
66         grid.setHasFixedSize(true)
```

```
67        grid.itemAnimator = SlideInItemAnimator()
68        if (prefs.isLoggedIn) {
69            dataManager.loadData(prefs.customer.key!!)  //Start loading data
70            checkEmptyState()   //Check for data existence
71        }
72    }
73
74    private fun checkEmptyState() {
75        if (adapter.itemCount > 0) {
76            TransitionManager.beginDelayedTransition(container)
77            grid.visibility = View.VISIBLE
78            noOrders.visibility = View.GONE
79        } else {
80            TransitionManager.beginDelayedTransition(container)
81            grid.visibility = View.GONE
82            noOrders.visibility = View.VISIBLE
83        }
84    }
85
86    override fun onDestroy() {
87        dataManager.cancelLoading()
88        super.onDestroy()
89    }
90
91    internal inner class OrdersViewHolder(view: View) : RecyclerView.ViewHolder(view) {
92        var image: CircularImageView = view.findViewById(R.id.order_img)
93        var key: TextView = view.findViewById(R.id.order_number)
94        var date: TextView = view.findViewById(R.id.order_date)
95        var price: TextView = view.findViewById(R.id.order_price)
96        var track: Button = view.findViewById(R.id.track_order)
97        var revoke: Button = view.findViewById(R.id.revoke_order)
98    }
99
100   internal inner class OrdersAdapter : RecyclerView.Adapter<OrdersViewHolder>() {
101       private var purchases: ArrayList<Purchase> = ArrayList(0)
102       private val loading: MaterialDialog
103
104       init {
105           setHasStableIds(true)
106           loading = prefs.getDialog()
107       }
108
109       override fun getItemCount(): Int {
110           return purchases.size
111       }
112
113       override fun onBindViewHolder(holder: OrdersViewHolder, position: Int) {
114           val purchase = getItem(position)
115           //navigation
116           holder.itemView.setOnClickListener({
117               navTrackingView(purchase)
118           })
119           holder.track.setOnClickListener({
120               navTrackingView(purchase)
121           })
122
123           //Revoke order
124           holder.revoke.setOnClickListener({
125               MaterialDialog.Builder(this@LiveOrdersActivity)
126                       .content("Do you wish to revoke Order #${purchase.key}")
127                       .positiveText("Continue")
128                       .negativeText("Cancel")
129                       .onPositive({ dialog, _ ->
130                           dialog.dismiss()
131                           if (prefs.isConnected) {
132                               loading.show()
133                               dispatchItemRemoved(purchase)
```

```
                    } else {
                        Toast.makeText(this@LiveOrdersActivity, "Item cannot be removed at this time",
                            Toast.LENGTH_SHORT).show()
                    }
                })
                .onNegative({ dialog, _ ->
                    dialog.dismiss()
                })
                .build().show()
        })

        //Order number
        holder.key.text = String.format("Order #%s", purchase.purchaseId)
        //Order date
        if (purchase.timestamp != null) {
            holder.date.text = DateUtils.getRelativeTimeSpanString(purchase.timestamp!!.time,
                System.currentTimeMillis(), DateUtils.SECOND_IN_MILLIS)
        }
        //Order price
        holder.price.text = NumberFormat.getCurrencyInstance(Locale.US).format(purchase.price?.
    toDouble())

        //Load profile image of user
        GlideApp.with(holder.itemView.context)
            .asBitmap()
            .load(prefs.customer.photo)
            .diskCacheStrategy(DiskCacheStrategy.AUTOMATIC)
            .transition(withCrossFade())
            .placeholder(R.drawable.motor_placeholder)
            .circleCrop()
            .into(holder.image)

    }
    ------------------------------------------------------------------------------------------
    private fun navTrackingView(purchase: Purchase) {
        if (prefs.isLoggedIn) {
            val intent = Intent(this@LiveOrdersActivity, TrackingActivity::class.java)
            intent.putExtra(TrackingActivity.EXTRA_PURCHASE, purchase)
            startActivity(intent)
        } else {
            startActivity(Intent(this@LiveOrdersActivity, AuthActivity::class.java))
        }
    }
    ------------------------------------------------------------------------------------------
    private fun getItem(position: Int): Purchase {
        return purchases[position]
    }
    ------------------------------------------------------------------------------------------
    override fun getItemId(position: Int): Long {
        return purchases[position].id
    }
    ------------------------------------------------------------------------------------------
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): OrdersViewHolder {
        return OrdersViewHolder(layoutInflater.inflate(R.layout.live_order_item, parent, false))
    }
    ------------------------------------------------------------------------------------------
    /** Add live orders here from database */
    fun addLiveOrder(newItems: List<Purchase>) {
        if (newItems.isNotEmpty()) {
            for (item in newItems) {
                var add = true
                for (i in 0 until purchases.size) {
                    if (item.id == purchases[i].id) add = false
                }

                if (add) {
                    purchases.add(item)
```

```
                    notifyItemRangeChanged(0, newItems.size)
                }
            }
        }
    }

    //Remove from database
    private fun dispatchItemRemoved(purchase: Purchase) {
        prefs.db.document(PhoenixUtils.PURCHASE_REF)
                .collection(prefs.customer.key!!)
                .whereEqualTo("key", purchase.key)
                .get()
                .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        val documents = task.result.documents
                        if (documents.isNotEmpty() && documents[0].exists()) {
                            documents[0].reference.delete().addOnSuccessListener { _ ->
                                loading.dismiss()
                                val position = purchases.indexOf(purchase)
                                purchases.removeAt(position)
                                notifyItemRemoved(position)
                                if (BuildConfig.DEBUG) {
                                    Timber.d("Item ${purchase.key} removed from database")
                                }
                                dataManager.loadData(prefs.customer.key!!)
                            }
                        }
                    } else {
                        loading.dismiss()
                        Toast.makeText(applicationContext, task.exception?.localizedMessage,
                                Toast.LENGTH_SHORT).show()
                    }
                }.addOnFailureListener { exception ->
                    loading.dismiss()
                    Toast.makeText(applicationContext, exception.localizedMessage,
                            Toast.LENGTH_SHORT).show()
                }
    }

}
}
```

File - E:\PROJECT\CSCD\phoenix-master\app\src\main\java\io\pergasus\ui\WalletPaymentSetup.kt

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.app.Activity
8  import android.widget.Button
9  import android.widget.Toast
10 import com.google.android.gms.common.api.ApiException
11 import com.google.android.gms.wallet.*
12 import io.pergasus.api.PhoenixUtils
13 import timber.log.Timber
14
15
16 /**
17  * Utility class for setting up Google Pay Wallet
18  */
19 class WalletPaymentSetup(private val activity: Activity, private val checkOut: Button) {
20     //Init payment client as test environment
21     private val paymentsClient: PaymentsClient = Wallet.getPaymentsClient(activity, Wallet.WalletOptions.
    Builder()
22             .setEnvironment(WalletConstants.ENVIRONMENT_TEST)
23             .build())
24
25     init {
26         isReadyToPay()
27     }
28
29     private fun isReadyToPay() {
30         //Setup payment request
31         val payRequest = IsReadyToPayRequest.newBuilder()
32                 //Add allowed payment methods
33                 .addAllowedPaymentMethods(mutableListOf(
34                         //Wallet payment methods
35                         WalletConstants.PAYMENT_METHOD_CARD,
36                         WalletConstants.PAYMENT_METHOD_TOKENIZED_CARD
37                 ))
38                 //Add allowed card networks
39                 .addAllowedCardNetworks(mutableListOf(
40                         //Wallet card networks
41                         WalletConstants.CARD_NETWORK_AMEX,
42                         WalletConstants.CARD_NETWORK_DISCOVER,
43                         WalletConstants.CARD_NETWORK_VISA,
44                         WalletConstants.CARD_NETWORK_MASTERCARD
45                 ))
46                 .build()
47
48         val task = paymentsClient.isReadyToPay(payRequest)
49         task.addOnCompleteListener(activity, { paymentTask ->
50             try {
51                 val result = paymentTask.getResult(ApiException::class.java)
52                 if (result) {
53                     // Show Google as payment option.
54                     Toast.makeText(activity.applicationContext,
55                             "Guess what! Google Payment option is enabled",
56                             Toast.LENGTH_SHORT).show()
57                 } else {
58                     // Hide Google as payment option.
59                     Toast.makeText(activity.applicationContext,
60                             "Oops! Google Payment option has been disabled",
61                             Toast.LENGTH_SHORT).show()
62                 }
63                 checkOut.isEnabled = true
64             } catch (e: ApiException) {
65                 Timber.e(e)
```

```
            checkOut.isEnabled = true
        }
    })
}

/**
 * Creates payment data request for token
 * @param price as string
 */
fun createPaymentDataRequest(price: String): PaymentDataRequest? {
    //Setup data for payment request
    val request = PaymentDataRequest.newBuilder()
        .setTransactionInfo(TransactionInfo.newBuilder()
            .setTotalPriceStatus(WalletConstants.TOTAL_PRICE_STATUS_FINAL)
            .setCurrencyCode(PhoenixUtils.DEF_CURRENCY)
            .setTotalPrice(price)
            .build())
        .addAllowedPaymentMethod(WalletConstants.PAYMENT_METHOD_CARD)
        .addAllowedPaymentMethod(WalletConstants.PAYMENT_METHOD_TOKENIZED_CARD)
        .setCardRequirements(CardRequirements.newBuilder()
            .addAllowedCardNetworks(mutableListOf(
                WalletConstants.CARD_NETWORK_AMEX,
                WalletConstants.CARD_NETWORK_DISCOVER,
                WalletConstants.CARD_NETWORK_VISA,
                WalletConstants.CARD_NETWORK_MASTERCARD
            ))
            .build())
    //Setup payment method tokenization parameters
    val params: PaymentMethodTokenizationParameters = PaymentMethodTokenizationParameters.
newBuilder()
        .setPaymentMethodTokenizationType(WalletConstants.
PAYMENT_METHOD_TOKENIZATION_TYPE_PAYMENT_GATEWAY)
        .addParameter(GATEWAY, "example")
        .addParameter(GATEWAY_MERCHANT_ID, "exampleGatewayMerchantId")
        .build()
    request.setPaymentMethodTokenizationParameters(params)
    return request.build()
}

/**
 * @return paymentsClient
 */
fun getPaymentClient(): PaymentsClient = paymentsClient

companion object {
    private const val GATEWAY = "gateway"
    private const val GATEWAY_MERCHANT_ID = "gatewayMerchantId"
}

}
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui;
6
7
8  import android.animation.Animator;
9  import android.animation.AnimatorListenerAdapter;
10 import android.animation.ObjectAnimator;
11 import android.animation.ValueAnimator;
12 import android.animation.ValueAnimator.AnimatorUpdateListener;
13 import android.annotation.SuppressLint;
14 import android.app.Activity;
15 import android.app.ActivityOptions;
16 import android.app.SharedElementCallback;
17 import android.content.Context;
18 import android.content.Intent;
19 import android.content.res.TypedArray;
20 import android.graphics.Color;
21 import android.graphics.ColorMatrixColorFilter;
22 import android.graphics.drawable.ColorDrawable;
23 import android.graphics.drawable.Drawable;
24 import android.graphics.drawable.TransitionDrawable;
25 import android.support.annotation.ColorInt;
26 import android.support.annotation.NonNull;
27 import android.support.annotation.Nullable;
28 import android.support.v4.content.ContextCompat;
29 import android.support.v7.widget.RecyclerView;
30 import android.support.v7.widget.RecyclerView.ViewHolder;
31 import android.transition.Transition;
32 import android.transition.TransitionInflater;
33 import android.util.Log;
34 import android.util.Pair;
35 import android.view.LayoutInflater;
36 import android.view.MotionEvent;
37 import android.view.View;
38 import android.view.View.OnClickListener;
39 import android.view.ViewGroup;
40 import android.widget.ProgressBar;
41
42 import com.bumptech.glide.ListPreloader;
43 import com.bumptech.glide.RequestBuilder;
44 import com.bumptech.glide.load.DataSource;
45 import com.bumptech.glide.load.engine.DiskCacheStrategy;
46 import com.bumptech.glide.load.engine.GlideException;
47 import com.bumptech.glide.load.resource.gif.GifDrawable;
48 import com.bumptech.glide.request.RequestListener;
49 import com.bumptech.glide.request.RequestOptions;
50 import com.bumptech.glide.request.target.Target;
51 import com.bumptech.glide.util.ViewPreloadSizeProvider;
52
53 import java.util.ArrayList;
54 import java.util.Collections;
55 import java.util.List;
56 import java.util.Map;
57
58 import io.pergasus.R;
59 import io.pergasus.api.DataLoadingSubject;
60 import io.pergasus.api.ProductItem;
61 import io.pergasus.api.ProductItemSorting.NaturalOrderWeigher;
62 import io.pergasus.api.ProductItemSorting.ProductItemComparator;
63 import io.pergasus.api.ProductItemSorting.ProductItemGroupWeigher;
64 import io.pergasus.api.ProductWeigher;
65 import io.pergasus.data.Product;
66 import io.pergasus.ui.widget.BadgedFourThreeImageView;
```

```
67  import io.pergasus.util.ObservableColorMatrix;
68  import io.pergasus.util.TransitionUtils;
69  import io.pergasus.util.ViewUtils;
70  import io.pergasus.util.glide.GlideApp;
71
72  import static com.bumptech.glide.load.resource.drawable.DrawableTransitionOptions.withCrossFade;
73  import static io.pergasus.util.AnimUtils.getFastOutSlowInInterpolator;
74
75  /**
76   * Project : phoenix-master
77   * Created by Dennis Bilson on Wed at 12:45 PM.
78   * Package name : io.pergasus.ui
79   */
80
81  public class SearchDataAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder>
82          implements DataLoadingSubject.DataLoadingCallbacks,
83          ListPreloader.PreloadModelProvider<Product> {
84      static final int REQUEST_CODE_VIEW_PRODUCT = 5407;
85      private static final String TAG = "SearchDataAdapter";
86
87      private static final int TYPE_PRODUCT = 0;
88      private static final int TYPE_LOADING_MORE = -1;
89
90      // we need to hold on to an activity ref for the shared element transitions :/
91      final Activity host;
92      private final LayoutInflater layoutInflater;
93      private final ProductItemComparator comparator;
94      @Nullable
95      private final DataLoadingSubject dataLoading;
96      private final int columns;
97      private final ColorDrawable[] shotLoadingPlaceholders;
98      private final ViewPreloadSizeProvider<Product> shotPreloadSizeProvider;
99
100     @ColorInt
101     private final
102     int initialGifBadgeColor;
103     private final List<ProductItem> items;
104     private boolean showLoadingMore;
105     private NaturalOrderWeigher naturalOrderWeigher;
106     private ProductWeigher shotWeigher;
107 ----------------------------------------------------------------------------------------------------
108     //Constructor
109     public SearchDataAdapter(Activity hostActivity,
110                              @Nullable DataLoadingSubject dataLoading,
111                              int columns,
112                              ViewPreloadSizeProvider<Product> shotPreloadSizeProvider) {
113         this.host = hostActivity;
114         this.dataLoading = dataLoading;
115         if (dataLoading != null) {
116             dataLoading.registerCallback(this);
117         }
118         this.columns = columns;
119         this.shotPreloadSizeProvider = shotPreloadSizeProvider;
120         layoutInflater = LayoutInflater.from(host);
121         comparator = new ProductItemComparator();
122         items = new ArrayList<>(0);
123         setHasStableIds(true);
124
125         // get the dribbble shot placeholder colors & badge color from the theme
126         TypedArray a = host.obtainStyledAttributes(R.styleable.DribbbleFeed);
127         int loadingColorArrayId =
128                 a.getResourceId(R.styleable.DribbbleFeed_shotLoadingPlaceholderColors, 0);
129         if (loadingColorArrayId == 0) {
130             shotLoadingPlaceholders = new ColorDrawable[]{new ColorDrawable(Color.DKGRAY)};
131         } else {
132             int[] placeholderColors = host.getResources().getIntArray(loadingColorArrayId);
133             shotLoadingPlaceholders = new ColorDrawable[placeholderColors.length];
```

```
134             for (int i = 0; i < placeholderColors.length; i++) {
135                 shotLoadingPlaceholders[i] = new ColorDrawable(placeholderColors[i]);
136             }
137         }
138         int initialGifBadgeColorId =
139                 a.getResourceId(R.styleable.DribbbleFeed_initialBadgeColor, 0);
140         initialGifBadgeColor = initialGifBadgeColorId == 0 ? 0x40ffffff : ContextCompat.getColor(host,
    initialGifBadgeColorId);
141         a.recycle();
142     }
143 
144     @Override
145     public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
146         switch (viewType) {
147             case TYPE_PRODUCT:
148                 return createProductHolder(parent);
149             case TYPE_LOADING_MORE:
150                 return new LoadingMoreHolder(
151                         layoutInflater.inflate(R.layout.infinite_loading, parent, false));
152         }
153         return null;
154     }
155 
156     private ProductViewHolder createProductHolder(ViewGroup parent) {
157         ProductViewHolder holder = new ProductViewHolder(layoutInflater.inflate(R.layout.search_item,
158                 parent, false));
159         holder.image.setBadgeColor(initialGifBadgeColor);
160         holder.image.setOnClickListener(new OnClickListener() {
161             @Override
162             public void onClick(View view) {
163                 //show details view
164                 Intent intent = new Intent();
165                 intent.setClass(host, DetailsActivity.class);
166                 intent.putExtra(DetailsActivity.EXTRA_SHOT,
167                         (Product) SearchDataAdapter.this.getItem(holder.getAdapterPosition()));
168                 SearchDataAdapter.this.setGridItemContentTransitions(holder.image);
169                 ActivityOptions options =
170                         ActivityOptions.makeSceneTransitionAnimation(host,
171                                 Pair.create(view, host.getString(R.string.transition_shot)),
172                                 Pair.create(view, host.getString(R.string
173                                         .transition_shot_background)));
174                 host.startActivityForResult(intent, REQUEST_CODE_VIEW_PRODUCT, options.toBundle());
175             }
176         });
177         // play animated GIFs whilst touched
178         holder.image.setOnTouchListener(new View.OnTouchListener() {
179             @Override
180             public boolean onTouch(View v, MotionEvent event) {
181                 // check if it's an event we care about, else bail fast
182                 int action = event.getAction();
183                 if (!(action == MotionEvent.ACTION_DOWN
184                         || action == MotionEvent.ACTION_UP
185                         || action == MotionEvent.ACTION_CANCEL)) return false;
186 
187                 // get the image and check if it's an animated GIF
188                 Drawable drawable = holder.image.getDrawable();
189                 if (drawable == null) return false;
190                 GifDrawable gif = null;
191                 if (drawable instanceof GifDrawable) {
192                     gif = (GifDrawable) drawable;
193                 } else if (drawable instanceof TransitionDrawable) {
194                     // we fade in images on load which uses a TransitionDrawable; check its layers
195                     TransitionDrawable fadingIn = (TransitionDrawable) drawable;
196                     for (int i = 0; i < fadingIn.getNumberOfLayers(); i++) {
197                         if (fadingIn.getDrawable(i) instanceof GifDrawable) {
198                             gif = (GifDrawable) fadingIn.getDrawable(i);
199                             break;
```

```
200                     }
201                 }
202             }
203             if (gif == null) return false;
204             // GIF found, start/stop it on press/lift
205             switch (action) {
206                 case MotionEvent.ACTION_DOWN:
207                     gif.start();
208                     break;
209                 case MotionEvent.ACTION_UP:
210                 case MotionEvent.ACTION_CANCEL:
211                     gif.stop();
212                     break;
213             }
214             return false;
215         }
216     });
217     return holder;
218 }
219 
220 @Override
221 public void onBindViewHolder(ViewHolder holder, int position) {
222     switch (getItemViewType(position)) {
223         case TYPE_PRODUCT:
224             bindProductHolder((Product) getItem(position), (ProductViewHolder) holder, position);
225             break;
226         case TYPE_LOADING_MORE:
227             bindLoadingViewHolder((LoadingMoreHolder) holder, position);
228             break;
229     }
230 }
231 
232 private void bindLoadingViewHolder(LoadingMoreHolder holder, int position) {
233     // only show the infinite load progress spinner if there are already items in the
234     // grid i.e. it's not the first item & data is being loaded
235     holder.progress.setVisibility((position > 0
236             && dataLoading != null
237             && dataLoading.isDataLoading())
238             ? View.VISIBLE : View.INVISIBLE);
239 }
240 
241 @SuppressLint("Range")
242 private void bindProductHolder(Product shot, ProductViewHolder holder, int position) {
243     GlideApp.with(host)
244             .load(shot.getUrl())
245             .listener(new RequestListener<Drawable>() {
246                 @Override
247                 public boolean onLoadFailed(@Nullable GlideException e, Object model, Target<Drawable>
    target, boolean isFirstResource) {
248                     return false;
249                 }
250 
251                 @Override
252                 public boolean onResourceReady(Drawable resource, Object model, Target<Drawable> target,
     DataSource dataSource, boolean isFirstResource) {
253                     if (!shot.getHasFadedIn()) {
254                         holder.image.setHasTransientState(true);
255                         ObservableColorMatrix cm = new ObservableColorMatrix();
256                         ObjectAnimator saturation = ObjectAnimator.ofFloat(
257                                 cm, ObservableColorMatrix.SATURATION, 0.0f, 1.0f);
258                         saturation.addUpdateListener(new AnimatorUpdateListener() {
259                             @Override
260                             public void onAnimationUpdate(ValueAnimator valueAnimator) {
261                                 // just animating the color matrix does not invalidate the
262                                 // drawable so need this update listener.  Also have to create a
263                                 // new CMCF as the matrix is immutable :(
264                                 holder.image.setColorFilter(new ColorMatrixColorFilter(cm));
```

```
                        }
                    });
                    saturation.setDuration(2000L);
                    saturation.setInterpolator(getFastOutSlowInInterpolator(host));
                    saturation.addListener(new AnimatorListenerAdapter() {
                        @Override
                        public void onAnimationEnd(Animator animation) {
                            holder.image.clearColorFilter();
                            holder.image.setHasTransientState(false);
                        }
                    });
                    saturation.start();
                    shot.setHasFadedIn(true);
                }
                return false;
            }
        })
        .apply(RequestOptions.placeholderOf(shotLoadingPlaceholders[position %
shotLoadingPlaceholders.length]))
        .apply(RequestOptions.diskCacheStrategyOf(DiskCacheStrategy.DATA))
        .apply(RequestOptions.fitCenterTransform())
//        .apply(RequestOptions.overrideOf(Target.SIZE_ORIGINAL, Target.SIZE_ORIGINAL))
        .transition(withCrossFade())
        .into(holder.image);
    // need both placeholder & background to prevent seeing through shot as it fades in
    holder.image.setBackground(
            shotLoadingPlaceholders[position % shotLoadingPlaceholders.length]);
    holder.image.setDrawBadge(shot.getAnimated());
    // need a unique transition name per shot, let's use it's url
    holder.image.setTransitionName(shot.getUrl());
    shotPreloadSizeProvider.setView(holder.image);
}

@Override
public int getItemViewType(int position) {
    if (position < getDataItemCount()
            && getDataItemCount() > 0) {
        ProductItem item = getItem(position);
        if (item instanceof Product) {
            return TYPE_PRODUCT;
        }
    }
    return TYPE_LOADING_MORE;
}

@Override
public void onViewRecycled(ViewHolder holder) {
    if (holder instanceof ProductViewHolder) {
        ProductViewHolder viewHolder = (ProductViewHolder) holder;
        viewHolder.image.setBadgeColor(initialGifBadgeColor);
        viewHolder.image.setDrawBadge(false);
        viewHolder.image.setForeground(ContextCompat.getDrawable(host, R.drawable.mid_grey_ripple)
);
    }
}

@Override
public long getItemId(int position) {
    if (getItemViewType(position) == TYPE_LOADING_MORE) {
        return -1L;
    }
    return ((long) getItem(position).hashCode());
}

int getItemPosition(long itemId) {
    for (int position = 0; position < items.size(); position++) {
        if (getItem(position).getId() == itemId) return position;
```

```
330         }
331         return RecyclerView.NO_POSITION;
332     }
333 
334     @Override
335     public int getItemCount() {
336         return getDataItemCount() + (showLoadingMore ? 1 : 0);
337     }
338 
339     @NonNull
340     @Override
341     public List<Product> getPreloadItems(int position) {
342         ProductItem item = getItem(position);
343         if (item instanceof Product) {
344             return Collections.singletonList((Product) item);
345         }
346         return Collections.emptyList();
347     }
348 
349     @Nullable
350     @Override
351     public RequestBuilder<?> getPreloadRequestBuilder(Product item) {
352         return GlideApp.with(host).load(item.getUrl());
353     }
354 
355     @Override
356     public void dataStartedLoading() {
357         if (showLoadingMore) return;
358         showLoadingMore = true;
359         notifyItemInserted(getLoadingMoreItemPosition());
360     }
361 
362     @Override
363     public void dataFinishedLoading() {
364         if (!showLoadingMore) return;
365         int loadingPos = getLoadingMoreItemPosition();
366         showLoadingMore = false;
367         notifyItemRemoved(loadingPos);
368     }
369 
370     /*Custom*/
371     @Nullable
372     private ProductItem getItem(int position) {
373         if (position < 0 || position >= items.size()) return null;
374         return items.get(position);
375     }
376 
377     int getItemColumnSpan(int position) {
378         switch (getItemViewType(position)) {
379             case TYPE_LOADING_MORE:
380                 return columns;
381             default:
382                 return getItem(position).getColspan();
383         }
384     }
385 
386     public void clear() {
387         items.clear();
388         notifyDataSetChanged();
389     }
390 
391     /**
392      * Main entry point for adding items to this adapter. Takes care of de-duplicating items and
393      * sorting them (depending on the data source). Will also expand some items to span multiple
394      * grid columns.
395      */
396     void addAndResort(List<? extends ProductItem> newItems) {
```

```
397         Log.d(TAG, "addAndResort: " + newItems.toString());
398         weighItems(newItems);
399         deduplicateAndAdd(newItems);
400         sort();
401         expandPopularItems();
402         notifyDataSetChanged();
403     }
404 
405     /**
406      * Calculate a 'weight' [0, 1] for each data type for sorting. Each data type/source has a
407      * different metric for weighing it e.g. likes etc. but some sources should keep
408      * the order returned by the API. Weights are 'scoped' to the page they belong to and lower
409      * weights are sorted earlier in the grid (i.e. in ascending weight).
410      */
411     private void weighItems(List<? extends ProductItem> items) {
412         if (items == null || items.isEmpty()) return;
413         ProductItemGroupWeigher weigher = null;
414         if (items.get(0) instanceof Product) {
415             if (shotWeigher == null) shotWeigher = new ProductWeigher();
416             weigher = shotWeigher;
417         }
418         if (weigher != null) {
419             weigher.weigh(items);
420         }
421     }
422 
423     /**
424      * De-dupe as the same item can be returned by multiple feeds
425      */
426     private void deduplicateAndAdd(List<? extends ProductItem> newItems) {
427         int count = getDataItemCount();
428         for (ProductItem newItem : newItems) {
429             boolean add = true;
430             for (int i = 0; i < count; i++) {
431                 ProductItem existingItem = getItem(i);
432                 if (existingItem != null && existingItem.equals(newItem)) {
433                     add = false;
434                     break;
435                 }
436             }
437             if (add) {
438                 add(newItem);
439             }
440         }
441     }
442 
443     private void add(ProductItem item) {
444         items.add(item);
445     }
446 
447     private void sort() {
448         Collections.sort(items, comparator); // sort by weight
449     }
450 
451     private void expandPopularItems() {
452         // for now just expand the first image per page which should be
453         // the most popular according to our weighing & sorting
454         List<Integer> expandedPositions = new ArrayList<>(0);
455         int page = -1;
456         int count = items.size();
457         for (int i = 0; i < count; i++) {
458             ProductItem item = getItem(i);
459             if (item instanceof Product && item.getPage() > page) {
460                 item.setColspan(columns);
461                 page = item.getPage();
462                 expandedPositions.add(i);
463             } else {
```

```
464             if (item != null) {
465                 item.setColspan(1);
466             }
467         }
468     }
469
470     // make sure that any expanded items are at the start of a row
471     // so that we don't leave any gaps in the grid
472     for (int expandedPos = 0; expandedPos < expandedPositions.size(); expandedPos++) {
473         int pos = expandedPositions.get(expandedPos);
474         int extraSpannedSpaces = expandedPos * (columns - 1);
475         int rowPosition = (pos + extraSpannedSpaces) % columns;
476         if (rowPosition != 0) {
477             int swapWith = pos + (columns - rowPosition);
478             if (swapWith < items.size()) {
479                 Collections.swap(items, pos, swapWith);
480             }
481         }
482     }
483 }
484 ---------------------------------------------------------------------------
485 void removeDataSource(String dataSource) {
486     for (int i = items.size() - 1; i >= 0; i--) {
487         ProductItem item = items.get(i);
488         if (dataSource.equals(item.getDataSource())) {
489             items.remove(i);
490         }
491     }
492     sort();
493     expandPopularItems();
494     notifyDataSetChanged();
495 }
496 ---------------------------------------------------------------------------
497 /**
498  * The shared element transition to dribbble shots & dn stories can intersect with the FAB.
499  * This can cause a strange layers-passing-through-each-other effect. On return hide the FAB
500  * and animate it back in after the transition.
501  */
502 private void setGridItemContentTransitions(View gridItem) {
503     View fab = host.findViewById(R.id.fab);
504     if (!ViewUtils.viewsIntersect(gridItem, fab)) return;
505
506     Transition reenter = TransitionInflater.from(host)
507             .inflateTransition(R.transition.grid_overlap_fab_reenter);
508     reenter.addListener(new TransitionUtils.TransitionListenerAdapter() {
509
510         @Override
511         public void onTransitionEnd(Transition transition) {
512             // we only want these content transitions in certain cases so clear out when done.
513             host.getWindow().setReenterTransition(null);
514         }
515     });
516     host.getWindow().setReenterTransition(reenter);
517 }
518 ---------------------------------------------------------------------------
519 int getDataItemCount() {
520     return items.size();
521 }
522 ---------------------------------------------------------------------------
523 private int getLoadingMoreItemPosition() {
524     return showLoadingMore ? getItemCount() - 1 : RecyclerView.NO_POSITION;
525 }
526 ---------------------------------------------------------------------------
527 @NonNull
528 static SharedElementCallback createSharedElementReenterCallback(
529         @NonNull Context context) {
530     String shotTransitionName = context.getString(R.string.transition_shot);
```

```
531         String shotBackgroundTransitionName =
532                 context.getString(R.string.transition_shot_background);
533         return new SharedElementCallback() {
534
535             /**
536              * We're performing a slightly unusual shared element transition i.e. from one view
537              * (image in the grid) to two views (the image & also the background of the details
538              * view, to produce the expand effect). After changing orientation, the transition
539              * system seems unable to map both shared elements (only seems to map the shot, not
540              * the background) so in this situation we manually map the background to the
541              * same view.
542              */
543             @Override
544             public void onMapSharedElements(List<String> names, Map<String, View> sharedElements) {
545                 if (sharedElements.size() != names.size()) {
546                     // couldn't map all shared elements
547                     View sharedShot = sharedElements.get(shotTransitionName);
548                     if (sharedShot != null) {
549                         // has shot so add shot background, mapped to same view
550                         sharedElements.put(shotBackgroundTransitionName, sharedShot);
551                     }
552                 }
553             }
554         };
555     }
556
557     /*package*/static class ProductViewHolder extends RecyclerView.ViewHolder {
558         BadgedFourThreeImageView image;
559
560         ProductViewHolder(View itemView) {
561             super(itemView);
562             image = itemView.findViewById(R.id.shot);
563         }
564     }
565
566     /*package*/static class LoadingMoreHolder extends RecyclerView.ViewHolder {
567         ProgressBar progress;
568
569         LoadingMoreHolder(View itemView) {
570             super(itemView);
571             progress = (ProgressBar) itemView;
572         }
573
574     }
575 }
576
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.annotation.TargetApi
8  import android.app.Activity
9  import android.app.assist.AssistContent
10 import android.net.Uri
11 import android.os.Build
12 import android.os.Bundle
13 import android.support.customtabs.CustomTabsIntent
14 import android.support.v4.content.ContextCompat
15 import android.view.Menu
16 import android.view.MenuItem
17 import android.view.ViewGroup
18 import android.widget.Toolbar
19 import com.bumptech.glide.Priority
20 import com.bumptech.glide.load.engine.DiskCacheStrategy
21 import com.bumptech.glide.load.resource.drawable.DrawableTransitionOptions
22 import io.pergasus.R
23 import io.pergasus.data.Product
24 import io.pergasus.ui.widget.ZoomageView
25 import io.pergasus.util.DownloadProductTask
26 import io.pergasus.util.ShareProductTask
27 import io.pergasus.util.bindView
28 import io.pergasus.util.customtabs.CustomTabActivityHelper
29 import io.pergasus.util.glide.GlideApp
30
31 /**
32  * Image details from a product
33  */
34 class ImageDetailsActivity : Activity() {
35
36     private val container: ViewGroup by bindView(R.id.container)
37     private val toolbar: Toolbar by bindView(R.id.toolbar)
38     private val imageView: ZoomageView by bindView(R.id.imageView)
39
40     private var product: Product? = null
41
42     override fun onCreate(savedInstanceState: Bundle?) {
43         super.onCreate(savedInstanceState)
44         setContentView(R.layout.activity_image_details)
45         setActionBar(toolbar)
46
47         toolbar.setNavigationOnClickListener({
48             onBackPressed()
49         })
50
51         //Get intent from parent activity
52         val intent = intent
53         if (intent.hasExtra(EXTRA_IMAGE_URL)) {
54             //Get intent data
55             product = intent.getParcelableExtra(EXTRA_IMAGE_URL)
56             loadImageFromURL()
57         }
58     }
59
60     private fun loadImageFromURL() {
61         if (product != null) {
62             //Toolbar title
63             toolbar.title = product?.name
64
65             //Load image
66             GlideApp.with(this)
```

```
67                  .load(product!!.url)
68                  .diskCacheStrategy(DiskCacheStrategy.DATA)
69                  .priority(Priority.IMMEDIATE)
70                  .override(800, 600)
71                  .transition(DrawableTransitionOptions.withCrossFade())
72                  .into(imageView)
73
74          imageView.setOnClickListener({ openLink(product!!.url!!) })
75
76          //Status bar color
77          window.statusBarColor = ContextCompat.getColor(this@ImageDetailsActivity, R.color.black)
78      }
79  }
80  ----------------------------------------------------------------------------------------------------
81  override fun onCreateOptionsMenu(menu: Menu?): Boolean {
82      menuInflater.inflate(R.menu.image, menu)
83      return true
84  }
85  ----------------------------------------------------------------------------------------------------
86  override fun onOptionsItemSelected(item: MenuItem?): Boolean {
87      return when (item?.itemId) {
88          R.id.menu_share -> {
89              if (product != null) {
90                  ShareProductTask(this, product!!).execute()
91              }
92              true
93          }
94          R.id.menu_get -> {
95              if (product != null) {
96                  DownloadProductTask(this, product!!).execute()
97              }
98              true
99          }
100         else -> super.onOptionsItemSelected(item)
101     }
102 }
103
104 ----------------------------------------------------------------------------------------------------
105 @TargetApi(Build.VERSION_CODES.M)
106 override fun onProvideAssistContent(outContent: AssistContent) {
107     outContent.webUri = Uri.parse(product!!.url)
108 }
109 ----------------------------------------------------------------------------------------------------
110 internal fun openLink(url: String) {
111     CustomTabActivityHelper.openCustomTab(
112             this@ImageDetailsActivity,
113             CustomTabsIntent.Builder()
114                     .setToolbarColor(ContextCompat.getColor(this@ImageDetailsActivity, R.color.dribbble))
115                     .addDefaultShareMenuItem()
116                     .build(),
117             Uri.parse(url))
118 }
119 ----------------------------------------------------------------------------------------------------
120 override fun onBackPressed() {
121     setResult(RESULT_OK)
122     finishAfterTransition()
123 }
124
125 companion object {
126     const val EXTRA_IMAGE_URL = "EXTRA_IMAGE_URL"
127 }
128 }
129
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.app.Activity
8  import android.os.Bundle
9  import android.support.v7.widget.LinearLayoutManager
10 import android.support.v7.widget.RecyclerView
11 import android.text.format.DateUtils
12 import android.transition.TransitionManager
13 import android.util.Log
14 import android.view.View
15 import android.view.ViewGroup
16 import android.widget.TextView
17 import android.widget.Toolbar
18 import com.bumptech.glide.ListPreloader
19 import com.bumptech.glide.RequestBuilder
20 import com.bumptech.glide.integration.recyclerview.RecyclerViewPreloader
21 import com.bumptech.glide.load.engine.DiskCacheStrategy
22 import com.bumptech.glide.request.target.Target
23 import com.bumptech.glide.util.ViewPreloadSizeProvider
24 import io.pergasus.BuildConfig
25 import io.pergasus.R
26 import io.pergasus.api.NotificationDataManager
27 import io.pergasus.api.PhoenixClient
28 import io.pergasus.data.PhoenixNotification
29 import io.pergasus.ui.widget.CircularImageView
30 import io.pergasus.util.bindView
31 import io.pergasus.util.glide.GlideApp
32 import java.util.*
33
34 /** Notification page */
35 class NotificationsActivity : Activity() {
36
37     private val container: ViewGroup by bindView(R.id.container)
38     private val grid: RecyclerView by bindView(R.id.grid)
39     private val toolbar: Toolbar by bindView(R.id.toolbar)
40     private val noNotifications: TextView by bindView(R.id.no_notifications)
41
42     private lateinit var prefs: PhoenixClient
43     private lateinit var adapter: NotificationsAdapter
44     private lateinit var layoutManager: LinearLayoutManager
45     private lateinit var dataManager: NotificationDataManager
46 ____________________________________________________________________________
47     override fun onCreate(savedInstanceState: Bundle?) {
48         super.onCreate(savedInstanceState)
49         setContentView(R.layout.activity_notifications)
50         setActionBar(toolbar)
51
52         toolbar.setNavigationOnClickListener({ finish() })  //End activity on back-arrow press
53
54         prefs = PhoenixClient(this)
55
56         //get intent data here
57         val intent = intent
58         if (intent != null) {
59             val title = intent.getStringExtra("title")
60             val body = intent.getStringExtra("body")
61             //Log results for now
62             if (BuildConfig.DEBUG) Log.d(TAG, "Message from notification is $title | $body")
63         }
64
65         val preloadSizeProvider: ViewPreloadSizeProvider<PhoenixNotification> = ViewPreloadSizeProvider()
66         adapter = NotificationsAdapter(preloadSizeProvider)
```

```
67        grid.adapter = adapter
68        grid.setHasFixedSize(true)
69        layoutManager = LinearLayoutManager(this)
70        grid.layoutManager = layoutManager
71        val shotPreloader: RecyclerViewPreloader<PhoenixNotification> =
72                RecyclerViewPreloader<PhoenixNotification>(this, adapter, preloadSizeProvider, 4)
73        grid.addOnScrollListener(shotPreloader)
74        dataManager = object : NotificationDataManager(this@NotificationsActivity) {
75            override fun onDataLoaded(data: List<PhoenixNotification>) {
76                adapter.addNotifications(data)
77                checkEmptyState()
78            }
79        }
80        dataManager.loadAllNotifications()
81        checkEmptyState()
82    }
83
84    private fun checkEmptyState() {
85        if (adapter.itemCount > 0) {
86            TransitionManager.beginDelayedTransition(container)
87            grid.visibility = View.VISIBLE
88            noNotifications.visibility = View.GONE
89        } else {
90            TransitionManager.beginDelayedTransition(container)
91            grid.visibility = View.GONE
92            noNotifications.visibility = View.VISIBLE
93        }
94    }
95
96
97    internal inner class NotificationsAdapter(private val preloadSizeProvider: ViewPreloadSizeProvider<
    PhoenixNotification>)
98        : RecyclerView.Adapter<NotificationHolder>(), ListPreloader.PreloadModelProvider<
    PhoenixNotification> {
99        private var notifications: ArrayList<PhoenixNotification> = ArrayList(0)
100
101       init {
102           setHasStableIds(true)
103       }
104
105       override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): NotificationHolder {
106           val view = layoutInflater.inflate(R.layout.notifications_item, parent, false)
107           return NotificationHolder(view)
108       }
109
110       override fun getItemCount(): Int {
111           return notifications.size
112       }
113
114       override fun onBindViewHolder(holder: NotificationHolder, position: Int) {
115           val phoenixNotifications = notifications[position]
116
117           //Load image
118           GlideApp.with(applicationContext)
119                   .load(phoenixNotifications.image)
120                   .placeholder(R.drawable.avatar_placeholder)
121                   .error(R.drawable.avatar_placeholder)
122                   .diskCacheStrategy(DiskCacheStrategy.AUTOMATIC)
123                   .override(Target.SIZE_ORIGINAL, Target.SIZE_ORIGINAL)
124                   .into(holder.image)
125
126           //Set title
127           holder.title.text = phoenixNotifications.title
128
129           //Set message
130           holder.message.text = phoenixNotifications.message
131
```

```
132             //Set time
133             if (phoenixNotifications.timestamp != null) {
134                 holder.timestamp.text = DateUtils.getRelativeTimeSpanString(phoenixNotifications.timestamp?
    .time!!,
135                         System.currentTimeMillis(), DateUtils.SECOND_IN_MILLIS,
136                         DateUtils.FORMAT_SHOW_TIME)
137
138             }
139
140             //Set item preloader
141             preloadSizeProvider.setView(holder.image)
142
143         }
144 ----------
145         override fun getPreloadItems(position: Int): MutableList<PhoenixNotification> {
146             val notification = getItem(position)
147             return Collections.singletonList(notification)
148         }
149 ----------
150         override fun getPreloadRequestBuilder(item: PhoenixNotification): RequestBuilder<*>? {
151             return GlideApp.with(applicationContext).load(item.image)
152         }
153 ----------
154         /** Main entry point for new [PhoenixNotification] */
155         fun addNotifications(newItems: List<PhoenixNotification>) {
156             val count = itemCount
157             for (data in newItems) {
158                 var add = true
159                 for (i in 0 until count) {
160                     val existingItem = getItem(i)
161                     if (existingItem != null && existingItem == data) {
162                         add = false
163                     }
164                 }
165                 if (add) {
166                     notifications.add(data)
167                 }
168             }
169             notifyDataSetChanged()
170         }
171 ----------
172         private fun getItem(position: Int): PhoenixNotification? {
173             if (position < 0 || position >= notifications.size) return null
174             return notifications[position]
175         }
176 ----------
177         override fun getItemId(position: Int): Long {
178             return notifications[position].hashCode().toLong()
179         }
180
181     }
182
183     internal inner class NotificationHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
184         var image: CircularImageView = itemView.findViewById(R.id.notification_image)
185         var title: TextView = itemView.findViewById(R.id.notification_title)
186         var message: TextView = itemView.findViewById(R.id.notification_msg)
187         var timestamp: TextView = itemView.findViewById(R.id.notification_time)
188     }
189
190     companion object {
191         private val TAG = NotificationsActivity::class.java.simpleName
192     }
193 }
194
```

```
1  /*
2   * Copyright (c) 2018. Property of Dennis Kwabena Bilson. No unauthorized duplication of this material
    should be made without prior permission from the developer
3   */
4
5  package io.pergasus.ui
6
7  import android.app.Activity
8  import android.os.Bundle
9  import android.support.v7.widget.GridLayoutManager
10 import android.support.v7.widget.RecyclerView
11 import android.view.ViewGroup
12 import android.widget.Toolbar
13 import com.afollestad.materialdialogs.MaterialDialog
14 import com.bumptech.glide.integration.recyclerview.RecyclerViewPreloader
15 import com.bumptech.glide.util.ViewPreloadSizeProvider
16 import io.pergasus.R
17 import io.pergasus.api.PhoenixClient
18 import io.pergasus.api.ProductItem
19 import io.pergasus.api.RelatedProductDataManager
20 import io.pergasus.data.Product
21 import io.pergasus.ui.recyclerview.SlideInItemAnimator
22 import io.pergasus.util.bindView
23
24 /**
25  * Related products: Shows products related to some other preset product
26  */
27 class RelatedProductsActivity : Activity() {
28     private val container: ViewGroup by bindView(R.id.container)
29     private val toolbar: Toolbar by bindView(R.id.toolbar)
30     private val results: RecyclerView by bindView(R.id.grid)
31
32     private lateinit var client: PhoenixClient
33     private lateinit var loading: MaterialDialog
34     private lateinit var dataManager: RelatedProductDataManager
35     private lateinit var layoutManager: GridLayoutManager
36     private lateinit var adapter: SearchDataAdapter
37
38 ---------------------------------------------------------------------------------------------------
39     override fun onCreate(savedInstanceState: Bundle?) {
40         super.onCreate(savedInstanceState)
41         setContentView(R.layout.activity_related_products)
42         setActionBar(toolbar)
43
44         toolbar.setNavigationOnClickListener({ finishAfterTransition() })
45
46         client = PhoenixClient(this@RelatedProductsActivity)
47         loading = client.getDialog()
48
49         dataManager = object : RelatedProductDataManager(this@RelatedProductsActivity) {
50             override fun onDataLoaded(data: List<ProductItem>) {
51                 adapter.addAndResort(data)
52                 checkEmptyState()
53             }
54         }
55
56         val columns: Int = resources.getInteger(R.integer.num_columns)
57         //RecyclerView
58         val preloadSizeProvider: ViewPreloadSizeProvider<Product> = ViewPreloadSizeProvider()
59         adapter = SearchDataAdapter(this, dataManager, columns, preloadSizeProvider)
60         setExitSharedElementCallback(SearchDataAdapter.createSharedElementReenterCallback(this))
61         results.adapter = adapter
62         results.itemAnimator = SlideInItemAnimator()
63         layoutManager = GridLayoutManager(this, columns)
64         layoutManager.spanSizeLookup = object : GridLayoutManager.SpanSizeLookup() {
65             override fun getSpanSize(position: Int): Int {
66                 return adapter.getItemColumnSpan(position)
```

```
67              }
68          }
69          results.layoutManager = layoutManager
70          results.setHasFixedSize(true)
71          val shotPreloader: RecyclerViewPreloader<Product> = RecyclerViewPreloader<Product>(this,
72                  adapter, preloadSizeProvider, 4)
73          results.addOnScrollListener(shotPreloader)
74
75          val intent = intent
76          if (intent.hasExtra(EXTRA_PRODUCT)) {
77              loading.show()
78              val product = intent.getParcelableExtra<Product>(EXTRA_PRODUCT)
79              //Compare by price, category and shop
80              dataManager.loadRelatedProductsFromSource(product)
81          }
82      }
83
84      override fun onDestroy() {
85          dataManager.cancelLoading()
86          super.onDestroy()
87      }
88
89      private fun checkEmptyState() {
90          if (loading.isShowing) loading.dismiss()
91      }
92
93      override fun onBackPressed() {
94          if (loading.isShowing) loading.dismiss()
95          else super.onBackPressed()
96      }
97
98      companion object {
99          const val EXTRA_PRODUCT = "EXTRA_PRODUCT"
100     }
101 }
102
```