

Introduction to Modern Application Development
Dr. Gaurav Raina
Prof. Tanmai Gopal
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – P14
Lecture - 32
Building a blog app

(Refer Slide Time: 00:04)

Recap

- Tweaked HelloWorld app (UI + Functionality)
- Toasts and Alerts.
- Logs.

Hey everyone welcome to module P14. In the previous module we tweaked a HelloWorld application UI and functionality based on our needs. We used android widgets like toasts and alerts we also used logs to help us understand debug our code better.

(Refer Slide Time: 00:18)

Contents

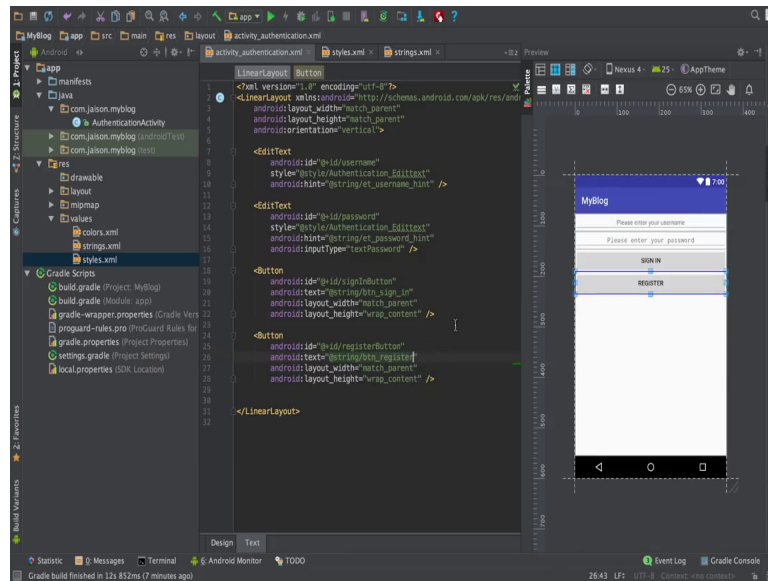
- Demo.
- Async Task
- Using gradle to integrate 3rd party libraries.
- Making Network APIs
- Intents
- RecyclerView

In this module we are going to build a very simple blog app which is going to have three different views issued to the user. The first one is going to present the user with the option to either sign in or register into the app.

The second page is going to show the user a list of articles and on clicking any those articles we will take them to a third view, which will be in the detailed blog page. We will be using the APIs that you have created in the previous modules while building your blog web app. While building this app we are going to have a look at something known as Async task, use the gradle to integrate third party libraries, make network API calls and use one of androids major app components called intents and finally, they are also going to use recycler views which is a very commonly used widget or view in a lot of android applications.

Let us start the creating a new android application, let us call it my blog clock next next empty activity, let us call this authentication activity, this will be the activity that is going to present the user the option to sign in or register let us finish and let android studio create your project ok.

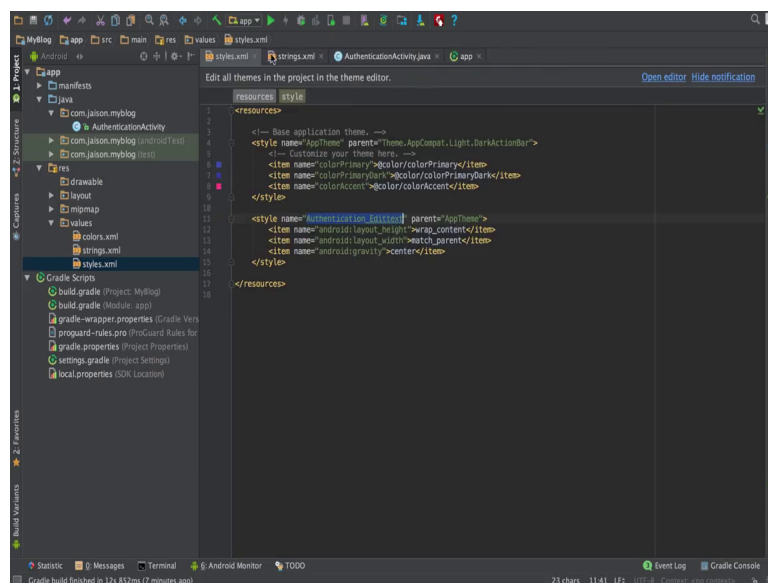
(Refer Slide Time: 01:50)



Now that our project is created, let us headover to activity under authentication. Let us remove this and user name is here set height as match parent width as match parent (Refer Time: 02:08) and let us also give it in orientation of vertical all right.

See now that (Refer Time: 02:19) let us add to edit text and width as match parent height as wrap content let us give this an id of username set a hint for this ok.

(Refer Slide Time: 02:45)

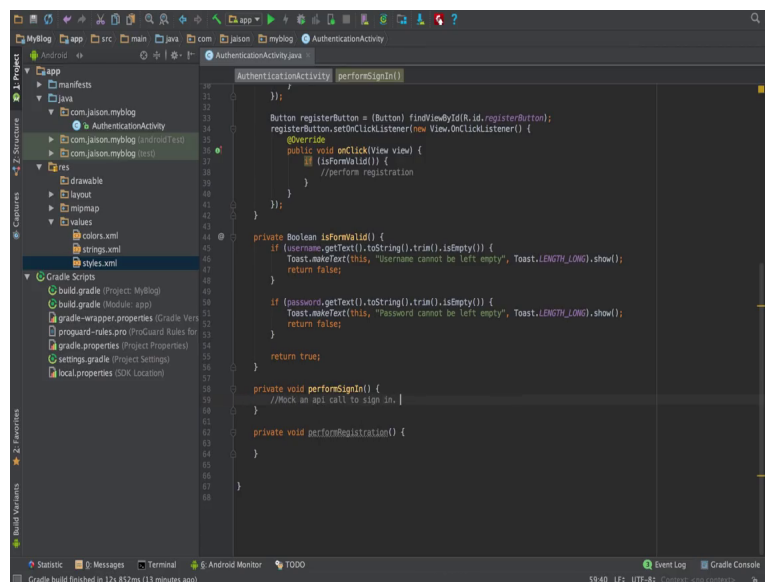


So, before we get to that we just add string e t underscore username underscore hint please enter your username similarly we will get a know password for the buttons. Let us

button we have. So, now, we have our buttons and our edit texts reference into our activity let us handle the click for both buttons ok.

So, now one thing that we want to check when the buttons are clicked is whether the username or password fields are empty, if it is empty then we should show an alert to the users saying that neither username nor password fields should be left empty. So, let us see how we can do that. Let us create another function which termed as Boolean and called is form valid what this method does is it check whether the username or password is empty. So, we can do is username dot get text dot to string dot trim is empty and return false.

(Refer Slide Time: 08:30)



```
AuthenticationActivity performSignIn() {
}

Button registerButton = (Button) findViewById(R.id.registerButton);
registerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (isFormValid()) {
            //perform registration
        }
    }
});

private Boolean isFormValid() {
    if (username.getText().toString().trim().isEmpty()) {
        Toast.makeText(this, "Username cannot be left empty", Toast.LENGTH_LONG).show();
        return false;
    }

    if (password.getText().toString().trim().isEmpty()) {
        Toast.makeText(this, "Password cannot be left empty", Toast.LENGTH_LONG).show();
        return false;
    }

    return true;
}

private void performSignIn() {
    //Mock an api call to sign in.
}

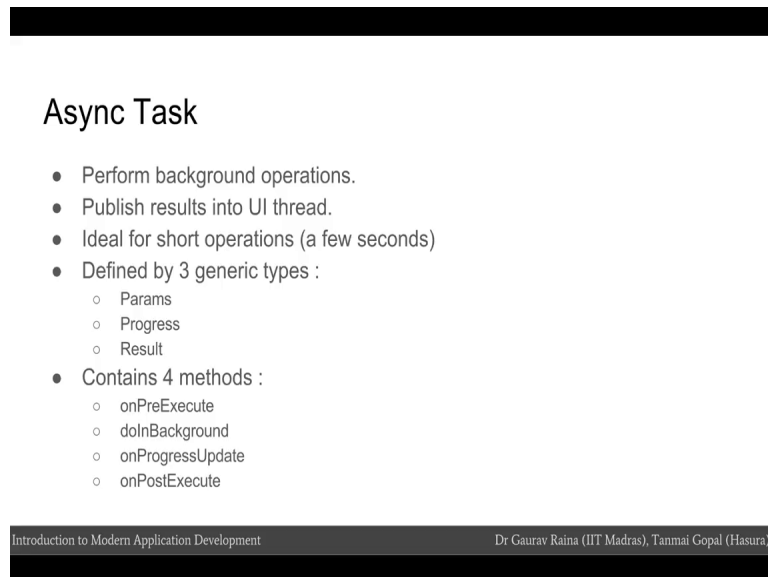
private void performRegistration() {
}
```

Similarly, if password dot get text dot to string dot trim is empty false and return true (Refer Time: 08:44). Now in case if any of these are empty let us show a toast to the user left empty it is a toast length of long. Let us show this toast and similarly for password we show toast all right. Now we do is basically check if form is valid and if it is perform sign in similarly we do same thing for registration. So, let us create two more methods called perform sign in ok.

Now, before we get to using the network API calls, let us try to mock a sign in operation by this I mean that when sometimes when in developing application, it can so happen that the API calls are not ready or have not been created. So, in scenarios like this instead of blocking your development you can mock or stimulate such a process so that you can

carry out the development until the APIs are right. So, let us see how we can do that here. So, when the user clicks on the sign in button let us perform let us call the perform sign in method, and over here let us mock an API call to sign in.

(Refer Slide Time: 10:34)



Async Task

- Perform background operations.
- Publish results into UI thread.
- Ideal for short operations (a few seconds)
- Defined by 3 generic types :
 - Params
 - Progress
 - Result
- Contains 4 methods :
 - onPreExecute
 - doInBackground
 - onProgressUpdate
 - onPostExecute

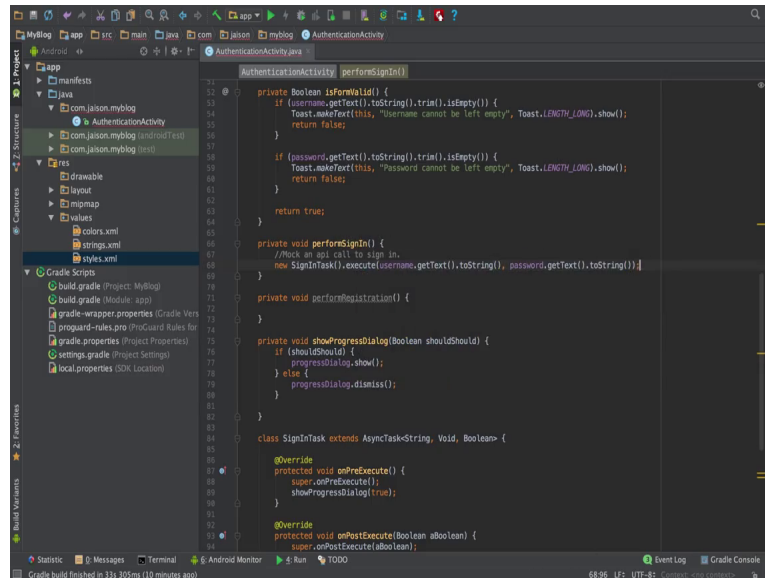
Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, for this we are going to use something known as Async task, let us see what those are. Async task help us perform background operations and then publish the result onto the UI thread. In android the UI thread is the main thread and is never supposed to perform any heavy operations. So, operations like network calls image processing etcetera are done on a background thread. Async task help in this, the ideal for short task which last a few seconds, it is defined by three generic types params, progress and result and it contains four major methods the first one being onpreexecute which is called before the task to be done begins, it is called on the UI thread and can be used to set up the task or show something like a profis bar to the user to indicate that something has been done.

Next comes do in background which is called right after onpreexecute and it is called on a background thread. The task to be performed happens here, one of the three generic types params is sent as a argument to this method. Next comes onprogressupdate which is called on the UI thread after a call to the method called publish progress which takes the type progress as is argument. Onprogressudate can be used to show progress to the user for instance print logs or animated progress path and finally, we have onpostexecute

which is invoked on the UI thread at after the background computation finishes, the result of the background computation is passed to this step as a parameter, which is a result param.

(Refer Slide Time: 12:07)



```
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

So, now let us use Async task inner application. So, let us start by creating a task called sign in task; sign in task which extends Async task as you can see it is looking for three types one is params progress in result. So, a param in this case would be string we should do string using a name and password, we do not really need a progress update for this task and finally, our result would be a Boolean indicating whether the call was successful or not. So, let us do that string void and Boolean and they are suitable in prompt if input or implement the methods click ok.

So, the method that must be implemented while extending Async task is doing background. Apart from that as I said we also have onpreexecute onpostexecute onprogressupdate. So, we will not be using onprogressupdate, but we will be using onpreexecute and onpostexecute. So, before we make the task we need to show an indicator to the user show him that that is something going on. So, let us make a class to show progress indicators. So, let us call it private void show progress dialog and this can be Boolean should show.

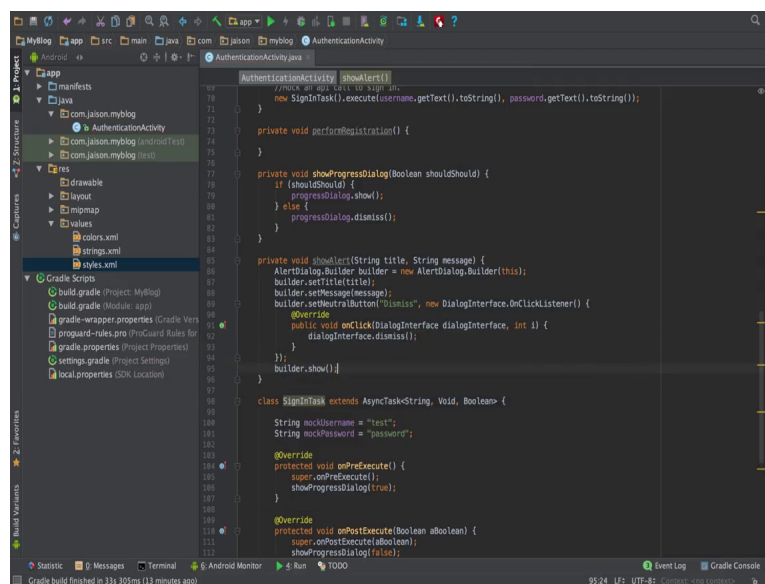
Let us also create a progress dialog, we have to import it and let us initialize it (Refer Time: 13:56). Let us give it a let us say indeterminate which should be a circular progress

dialog and give it a message of please wait. In show progress dialog you check if you are supposed to show it we are supposed to show it show and dismiss all right. So, on pre execute that is before we begin our task let us show our progress dialog and after our execution is done let us hide our progress dialog.

In the doing background method we would be receiving two strings first one would be username, it should be strings of zero and the other one can be password is strings of 1; do something with this and return true or false all right now let us see how we can do this.

At perform sign in let us do new sign in task dot execute and let us say username dot get text dot to string, password dot get text dot to string all right. So, we will be getting the username and password over here.

(Refer Slide Time: 15:53)



```
AuthenticationActivity showAlert() {
    // Mock @Mock class for sign in
    new SigninTask().execute(username.getText().toString(), password.getText().toString());
}

private void performRegistration() {
}

private void showProgressDialog(Boolean should) {
    if (should) {
        progressDialog.show();
    } else {
        progressDialog.dismiss();
    }
}

private void showAlert(String title, String message) {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle(title);
    builder.setMessage(message);
    builder.setNegativeButton("Dismiss", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface, int i) {
            dialogInterface.dismiss();
        }
    });
    builder.show();
}

class SigninTask extends AsyncTask<String, Void, Boolean> {
    String mockUsername = "test";
    String mockPassword = "password";

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        showProgressDialog(true);
    }

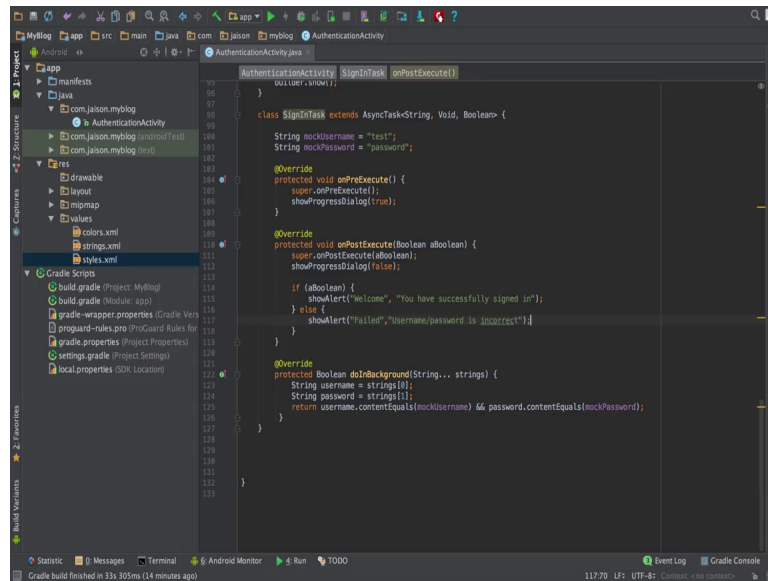
    @Override
    protected void onPostExecute(Boolean aBoolean) {
        super.onPostExecute(aBoolean);
        showProgressDialog(false);
    }
}
```

So, now since this is mock data let us make to mock value for mock username let us call this test and mock password let us call this password all right. So, what we are going to do here is we are going to check whether the username equals mock, whether username equals to the mock username and the password equals to the mock password and based on that we return true or false.

So, we say return username content equals mock username and username password content equals mock password all right, and over here is which true based on the result of

this sign in progress we should show alert to the user indicating whether this sign in has been successful or not. So, let us make a (Refer Time: 16:00) for that show alert which takes into here which takes the two parameters title and message let us set this up now title will be title, message will be message been sent, let us put one neutral button called dismiss which would dismiss the dialog an and sign in show the dialog all right.

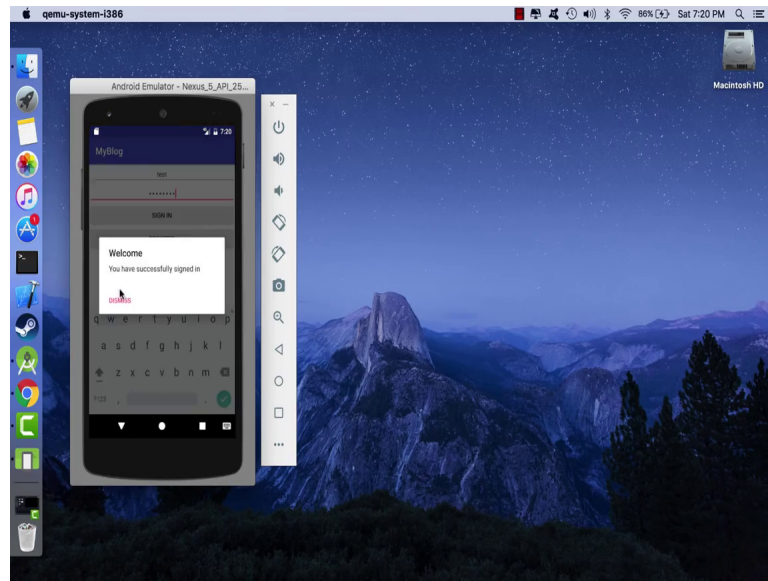
(Refer Slide Time: 17:48)



```
AuthenticationActivity | SignInTask | onPostExecute()
    97 | public void show() {
    98 | }
    99 |
   100 | class SignInTask extends AsyncTask<String, Void, Boolean> {
   101 |     String mockUsername = "test";
   102 |     String mockPassword = "password";
   103 |
   104 |     @Override
   105 |     protected void onPreExecute() {
   106 |         super.onPreExecute();
   107 |         showProgressDialog(true);
   108 |     }
   109 |
   110 |     @Override
   111 |     protected void onPostExecute(Boolean aBoolean) {
   112 |         super.onPostExecute(aBoolean);
   113 |         showProgressDialog(false);
   114 |
   115 |         if (aBoolean) {
   116 |             showAlert("welcome", "You have successfully signed in!");
   117 |         } else {
   118 |             showAlert("Failed", "username/password is incorrect");
   119 |         }
   120 |     }
   121 |
   122 |     @Override
   123 |     protected Boolean doInBackground(String... strings) {
   124 |         String username = strings[0];
   125 |         String password = strings[1];
   126 |         return username.contentEquals(mockUsername) && password.contentEquals(mockPassword);
   127 |     }
   128 |
   129 | }
   130 |
   131 |
   132 |
   133 | }
```

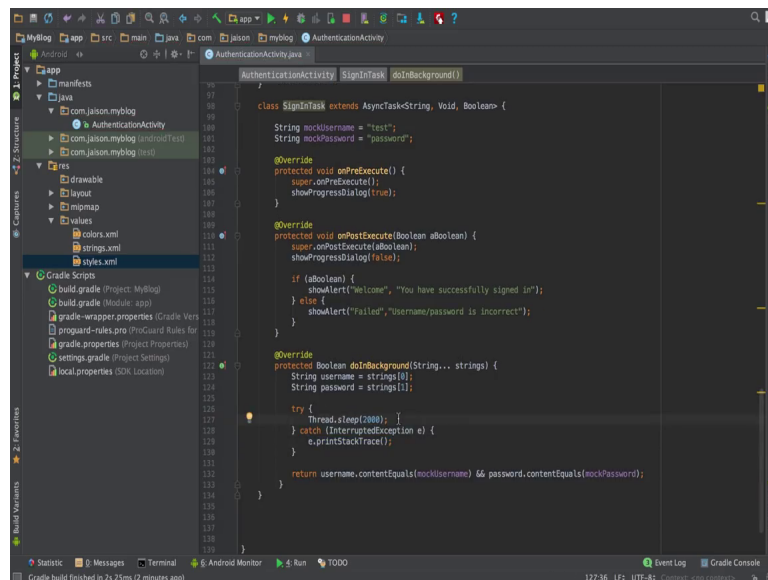
So, if it is true show alert saying welcome you have successfully signed in. If it failed username or password is incorrect all right. So, now, let us run our app and see this in action, first we try clicking an empty usernames enter something.

(Refer Slide Time: 18:48)



So, that is happening because the toast length is long. So, the next toast is shown only after the previous toast is dismissed. So, that works next let us enter a username and let us enter test and password it says successfully signed in. Although it did not show the progress part this is happening because our code is getting executed immediately.

(Refer Slide Time: 19:12)



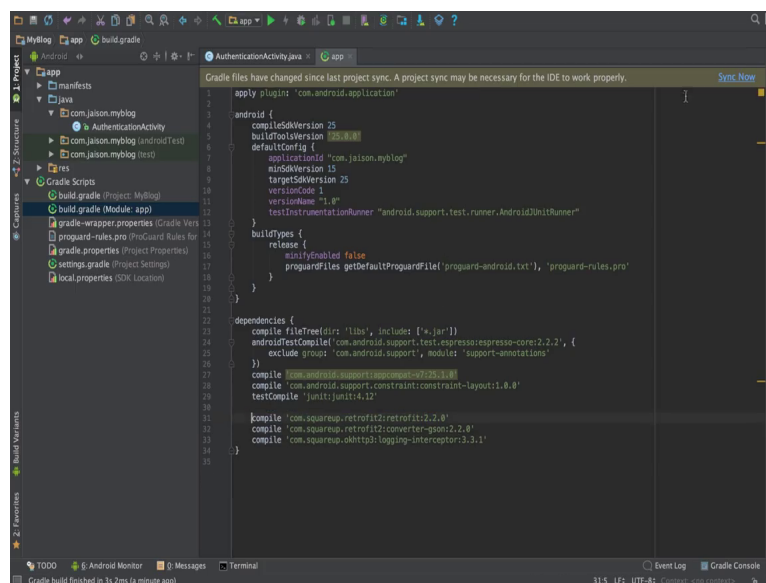
So, let us add something to make our threads sleep for about two seconds let us say thread dot sleep 2000 milliseconds which is seconds. Now as you can see android studio is prompting to surround this in a try and catch statement let us do that. So, now, what is

going to happen is that this operation is only going to be performed after two seconds. So, this return statement will only be called after two seconds and hopefully we should see our progress dialog one more thing we have to change is set indeterminate should be true. So, that the circular indication as the re (Refer Time: 19:50) ok.

Let us check this out let us say test password as you can see we have a progress indicator which is shown for two seconds as said username or password failed. I think that is because I have entered the I think there is a type one password (Refer Time: 20:11). It says you have successfully logged in and in case we enter something else it says username password is incorrect fine. So, now, we know how to mock data and carry on the develop let us now move on to making network API calls to make network API calls in our android app we are going to use a third party library called retrofit. Let us head on to square dot github dot io slash retrofit and you can read through this to understand how it works to integrate it we are going to use gradle and just copy that line go into your build dot gradle the app level build dot gradle file and place it.

So, this is to add retrofit as library to our app. Secondly, we are going to deal with json objects to make our requests and responses, to convert our requests and responses to and from json we are going to use a library called Gson, retrofit provides support for Gson and let us integrate that as well.

(Refer Slide Time: 21:16)



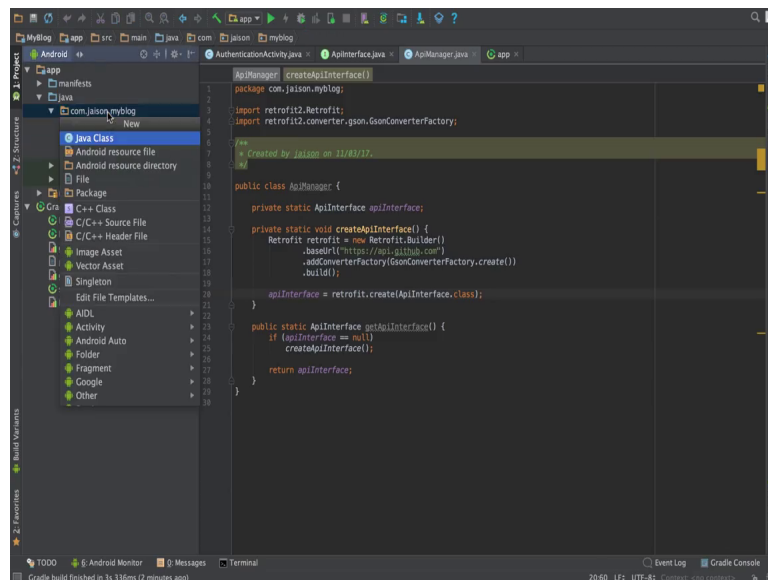
```
1  apply plugin: 'com.android.application'
2
3  android {
4      compileSdkVersion 25
5      buildToolsVersion '25.0.0'
6      defaultConfig {
7          applicationId "com.jaison.myblog"
8          minSdkVersion 13
9          targetSdkVersion 25
10         versionCode 1
11         versionName "1.0"
12         testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
13     }
14     buildTypes {
15         release {
16             minifyEnabled false
17             proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
18         }
19     }
20 }
21
22 dependencies {
23     compile fileTree(dir: 'libs', includes: ['*.jar'])
24     androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
25         exclude group: 'com.android.support', module: 'support-annotations'
26     })
27     compile 'com.android.support:appcompat-v7:23.0.0'
28     compile 'com.android.support.constraint:constraint-layout:1.0.0'
29     testCompile 'junit:junit4.12'
30
31     compile 'com.squareup.retrofit2:retrofit:2.2.0'
32     compile 'com.squareup.retrofit2:converter-gson:2.2.0'
33     compile 'com.squareup.okhttp3:logging-interceptor:3.3.1'
34 }
35
```

And finally, to log our API calls we are going to use something known as the log in interceptor. So, now, sink your project, gradle will download these libraries and integrated into our project ok.

Now, that is come let us understand what this means. A library name in gradle is separated into three parts which is group id colon artifact id colon version. So, here the group id is com dot square up dot retrofit two, the artifact id is retrofit and the version is 2.0 all right. So, now, that the libraries are been added let us get started with building a retrofit track.

First thing is retrofit provides an API interface to make our API calls. So, let us create an interface first let us call it API interface all right, this is it next let us create a manager which will handle the API calls first let us call it API manager class all right. Now here let us create a private static instant of API interface, then create a method called create API interface and finally, get a for API interface here we should check if API interface is null if it is null.

(Refer Slide Time: 23:00)



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a package named 'com.jaison.myblog'. The code editor displays the following Java code for 'ApiManager.java':

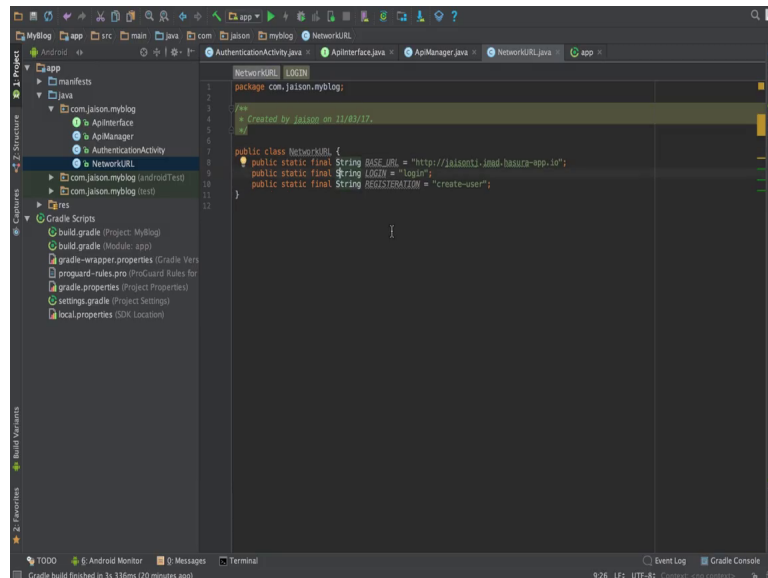
```
1 package com.jaison.myblog;
2
3 import retrofit2.Retrofit;
4 import retrofit2.converter.gson.GsonConverterFactory;
5
6 // Created by jaison on 11/03/17.
7
8 public class ApiManager {
9     private static ApiInterface apiInterface;
10
11     private static void createApiInterface() {
12         Retrofit retrofit = new Retrofit.Builder()
13             .baseUrl("https://api.github.com")
14             .addConverterFactory(GsonConverterFactory.create())
15             .build();
16
17         apiInterface = retrofit.create(ApiInterface.class);
18     }
19
20     public static ApiInterface getApiInterface() {
21         if (apiInterface == null) {
22             createApiInterface();
23         }
24         return apiInterface;
25     }
26 }
```

We should create oops static create API interface and then return API interface let us see how we can create the API interface.

So, let us head back to the documentation and this is how it work. So, input is this one converter factory for free using this is a base ULR which we will change later and here

API is service API interface is equal to API interface dot class all right. So, now, that we have set this up let us create another class to statically handle all of our ULRs.

(Refer Slide Time: 23:56)



```
NetworkURL LOGIN
package com.jaison.myblog;

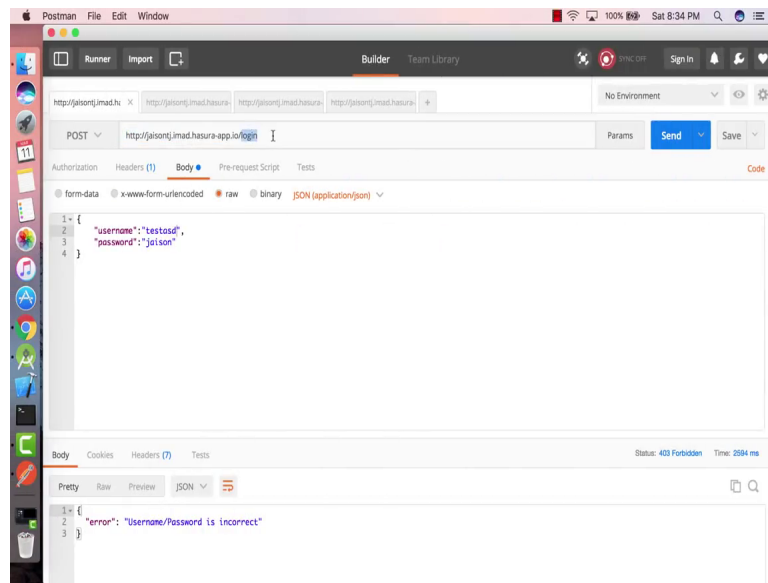
import java.util.*;

public class NetworkURL {
    public static final String BASE_URL = "http://jaisonj-imad-hasura-app-10";
    public static final String LOGIN = "login";
    public static final String REGISTRATION = "create-user";
}
```

Let us call it network URL public static string base URL which in this case would be the base URL of your Gson project. So, in my case it would be jsonj dot imad dot hasura app dot I add that you do not need this slash. So, mutually remove this slash.

Next first create a log in URL would be those login as you have create the APIs and registration was create user let us have a look at how our login and the registration API look like. So, the login API is as follows the URL that we are about to hit is this one the end point slash login.

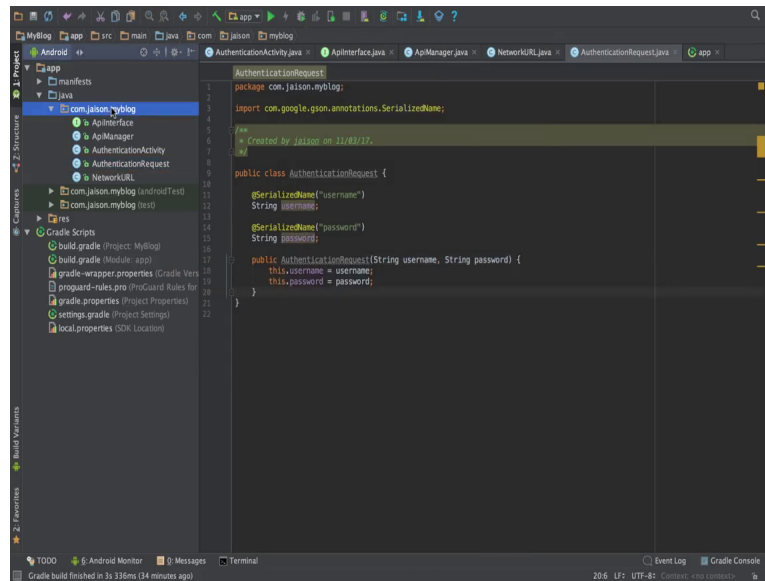
(Refer Slide Time: 24:54)



Which is a post request with a json body of type username with two keys username and password with the respective values and it gives the response to us in json which is either say error username password is incorrect or if you were to send in the right username and password it would send your response with a key message. And which has you have logged in successfully. Similarly registration is also the same same type of API which send in a request body of username and password which is of type json it is a post request, and the response is also the same it would either say error or it would say message and you have successfully logged in.

Ok. So, now, let us handle this in our let us start by creating our authentication requests and response pojoes, pojoes stands for plain java objects. So, let us call it authentication request since both are login and our create user request body is the same we can use the same authentication request body in our.

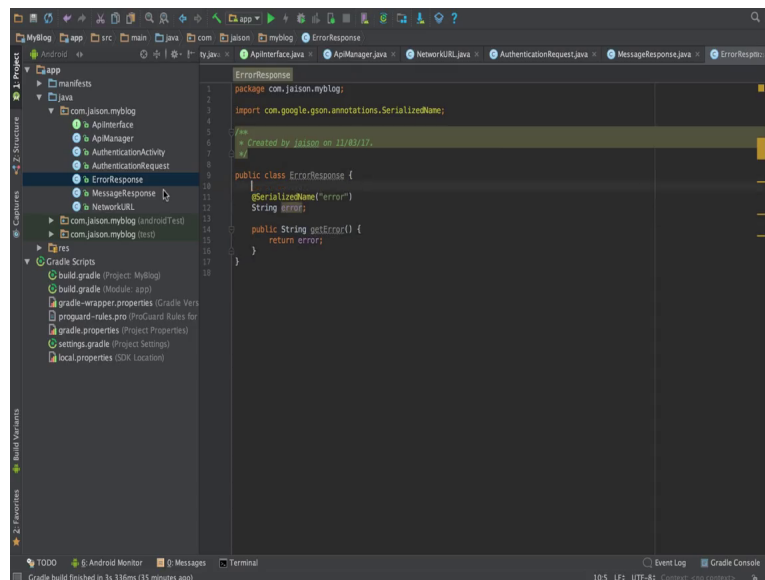
(Refer Slide Time: 26:04)



```
1 package com.jaison.myblog;
2
3 import com.google.gson.annotations.SerializedName;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

The serialized name is the name for the json, name for the json keys should be username should be a password. Let us create a constructor all right.

(Refer Slide Time: 26:43)



```
1 package com.jaison.myblog;
2
3 import com.google.gson.annotations.SerializedName;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

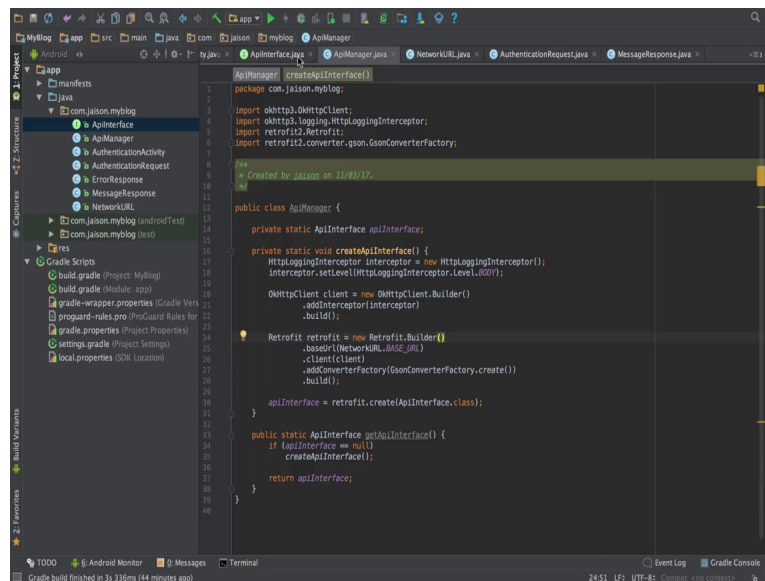
Now, next let us create let us just create an object to handle the message response and other one to handle the error response and another one for the error response. Now that is done let us now create these APIs in our API interface. So, the login request is a post request let us call let us access this at login and call is an object to (Refer Time: 27:39) retrofit it takes in a generic type. So, the response over here for our login API would be

the message response let us import this and a let us call this method login which takes in a body of type authentication request ok.

Similarly, registration were also be the same, another thing that we need to do is to go back to API manager and change this to our base URL this we run our app let us put a login interceptor in place so that we can see logs of all the API calls made by our app. So, this we need to create an http login interceptor let us call interceptor, and let us set this login interceptor level to body you can see what each of these stands for by going into the method.

Next is create an http client and add the interceptor to this and then finally, build it. So, here let us add this to our client (Refer Time: 29:09) object. So, now, we have our login interceptor also set in place.

(Refer Slide Time: 29:21)



```
1 package com.jaison.myblog;
2
3 import okhttp3.OkHttpClient;
4 import okhttp3.logging.HttpLoggingInterceptor;
5 import retrofit2.Retrofit;
6 import retrofit2.converter.gson.GsonConverterFactory;
7
8 Created by jaison on 11/03/17.
9
10
11
12 public class ApiManager {
13     private static ApiInterface apiInterface;
14
15     private static void createApiInterface() {
16         HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
17         interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
18         OkHttpClient client = new OkHttpClient.Builder()
19             .addInterceptor(interceptor)
20             .build();
21
22         Retrofit retrofit = new Retrofit.Builder()
23             .baseUrl(NetworkURL.BASE_URL)
24             .client(client)
25             .addConverterFactory(GsonConverterFactory.create())
26             .build();
27
28         apiInterface = retrofit.create(ApiInterface.class);
29     }
30
31     public static ApiInterface getApiInterface() {
32         if (apiInterface == null)
33             createApiInterface();
34         return apiInterface;
35     }
36 }
37
38
39
40
41
42
43
44
45
46
47
48
```

Now let us make this API calls in our authentication activity let me choose all of these tabs. Let us now go to a perform sign in method let us take these out and let us call our API manager get the API interface and call the log in and then pass the authentication request body with username and password.

(Refer Slide Time: 29:49)

```

71     Toast.makeText(this, "Password cannot be left empty", Toast.LENGTH_LONG).show();
72     return false;
73 }
74
75     return true;
76 }
77
78     private void performSignIn() {
79         showProgressDialog(true);
80         ApiManager.getApiInterface().login(new AuthenticationRequest(username.getText().toString().trim(), password.getText())
81             .enqueue(new Callback<MessageResponse>() {
82                 @Override
83                 public void onResponse(Call<MessageResponse> call, Response<MessageResponse> response) {
84                     showProgressDialog(false);
85                     if (response.isSuccessful()) {
86                         showAlert("welcome", response.body().getMessage());
87                     } else {
88                         try {
89                             String errorMessage = response.errorBody().toString();
90                             try {
91                                 ErrorResponse errorResponse = new Gson().fromJson(errorMessage, ErrorResponse.class);
92                                 showAlert("Signin failed", errorResponse.getError());
93                             } catch (IOException e) {
94                                 JSONException jsonException = new JSONException(errorMessage);
95                                 showAlert("Signin failed", "Something went wrong");
96                             }
97                         } catch (IOException e) {
98                             e.printStackTrace();
99                             showAlert("Signin failed", "Something went wrong");
100                        }
101                    }
102                }
103            });
104
105     @Override
106     public void onFailure(Call<MessageResponse> call, Throwable t) {
107         showProgressDialog(false);
108         showAlert("Signin failed", "Something went wrong");
109     }
110 }
111
112     private void performRegistration() {
113         showProgressDialog(true);
114     }
115 }

```

Once that is done let us enqueue this request and handle it in their call back and the call back provided by retrofit, which basically is two interface methods call on response and on failure on failure gets called when the request has failed in the sense that maybe the internet connection is not working or something (Refer Time: 30:16) and if the API call gets any response from the server, the on response method gets called it is our duty in the on response method to check whether the response was successful and what this tells what this is successful method does is checks if this status code of the response is between 200 or 300, and if it is not then it is then it returns a false for is successful.

So here we will show an alert saying welcome or we can even access the body of the response, which should be of type message response else, if the API call failed to some changes in which ideally should be if the username or password is incorrect in this case we we need to convert the response into the error response body. So, that we would be using Gson you would be doing this accessing the error message given by the response object error body string.

Since this string method throws a throws an io (Refer Time: 31:28) exception we have to wrap this in a try catch statement let us do that. Once we have done this what is next convert this error message into or json; for this two we have to wrap the method in try catch statement and do a error response error response equals new Gson dot from json string json would be error message and type would be error response dot class. So, now, we have our error response, let show alert to user saying sign in failed and error response dot get error sign in failed error response.

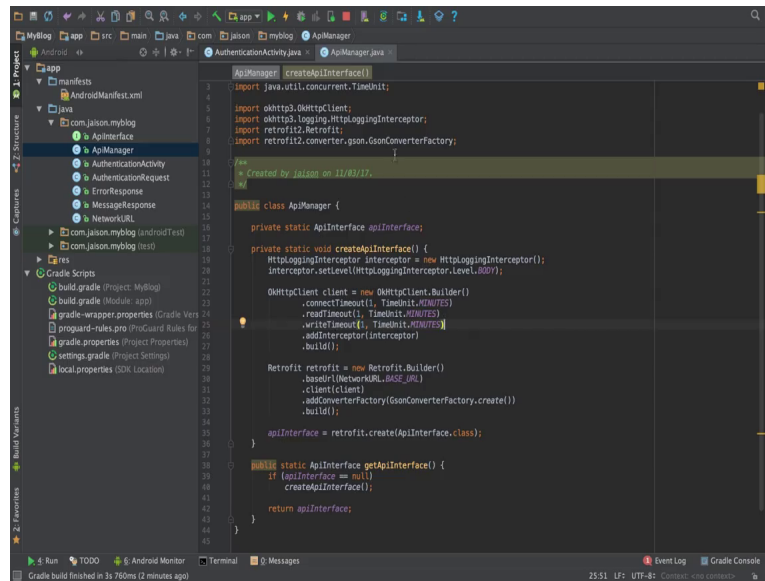
In case there is an exception let us print that stack trace and show the alert to the user saying sign in failed something went wrong similarly over here and one more here.

Another thing that we need to add is the progress indicator. So, show progress dialog true and highlight here. Since I registration call is also same let us do the same thing for here I am going to copy this and paste it here replace this with registration everything else would be the same except registration failed and let us. So, now, and also let us add the perform registration method into here perform registration all right well let us run the app and see let us type in some an username and password (Refer Time: 33:58) as you can see the app has crashed.

Let us see why that is the android monitor and it says no class def found error failed resolution of http internal platform. So, you may know that there is something to do with the http log in inceptor, let us see this mostly would be because of mismatch in the version as 3.4.1 would be the latest one. A simple Google search would reveal this to you let us rerun the app now, let us do the same thing again the app crashed again let us see why this is this time. It says security exception permission denied missing internet connection. So, this means that we have forgot to add the internet permission in our android (Refer Time: 34:53). So, let us do that let us add uses permission android name internet and that is it and let us run the app again.

Let us try this time it says username or password is incorrect and sign in failed. Let us check out our logs. So, as you can see we have made a post request to login end point with this json object and response has been 403 with error username or password is incorrect now try a registration API. So, let us create a new account let us call it jaison password can be jaison and let us click on register. So, it says something went wrong now that should not really happen. So, let us see why this happened it says http failed socket timeout exception it means that the socket is timing out. So, let us increase the timeout duration in our client. So, say connect timeout can be one and time unit dot minutes similarly we have read timeout which will be one time unit minutes and write timeout one time unit minutes.

(Refer Slide Time: 36:08)



```
ApiManager createApiInterface() {
    3   import java.util.concurrent.TimeUnit;
    4
    5   import okhttp3.OkHttpClient;
    6   import okhttp3.logging.HttpLoggingInterceptor;
    7   import retrofit112.Retrofit;
    8   import retrofit112.converter.gson.GsonConverterFactory;
    9
10
11   * created by jaison on 11/03/17.
12
13
14   class ApiManager {
15
16   private static ApiInterface apiInterface;
17
18   private static void createApiInterface() {
19       HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
20       interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
21
22       OkHttpClient client = new OkHttpClient.Builder()
23           .connectTimeout(1, TimeUnit.MINUTES)
24           .readTimeout(1, TimeUnit.MINUTES)
25           .writeTimeout(1, TimeUnit.MINUTES)
26           .addInterceptor(interceptor)
27           .build();
28
29       Retrofit retrofit = new Retrofit.Builder()
30           .baseUrl(NetworkURL.BASE_URL)
31           .client(client)
32           .addConverterFactory(GsonConverterFactory.create())
33           .build();
34
35       apiInterface = retrofit.create(ApiInterface.class);
36
37
38   static ApiInterface getApiInterface() {
39       if (apiInterface == null)
40           createApiInterface();
41
42       return apiInterface;
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
}
```

So, this basically sets the connection read and write timeout to one minute let us read on the app now let us try our registration API again. So, now, the error says that the duplicate key value violates unique constraint. So, I guess I have already created a user with a name jaison. So, let us create another user named imad and with the password imad, so this user ok.

So, this API call works as well we can check the response it says user successfully created 200 response and post perfect let us try sign in with the same details. So, everything works now. So, on successful sign in let us now create a new activity to be shown to the user which will show him a list of articles. So, that right click over here go to activity click on empty activity let us call this activity blog list activity or rather article list activity click on finish. Another thing that we did not notice is that android studio will automatically add this activity into our (Refer Time: 37:39) this is very important in case this is not added here the app will not run and it will be crashed.

Let us go back to our authentication activity and create a new method called navigate to article list activity. To open up a new activity we need to use something known as intents.

(Refer Slide Time: 38:00)

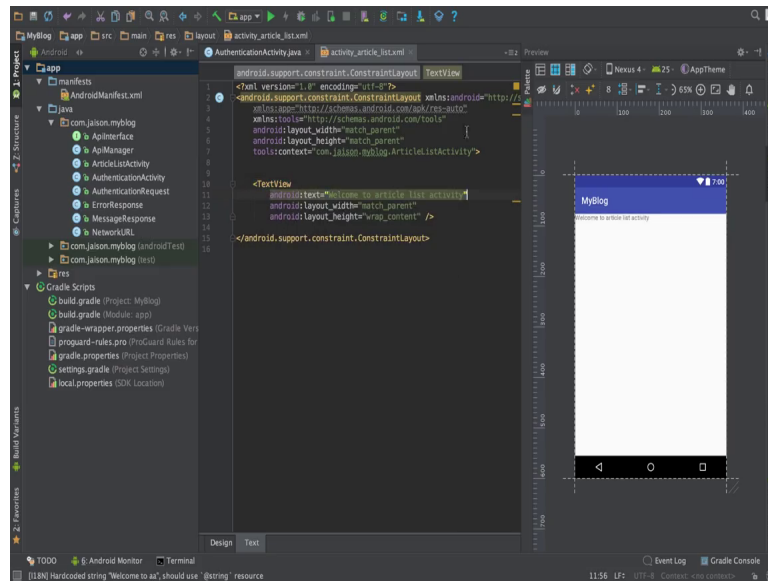
Intent

- Asynchronous message.
- Used to activate components like activities, services and broadcast receivers.
- Created with an Intent object.
- Two types :
 - Explicit Intent : defines a message to activate a specific component.
 - Implicit Intent : activates a specific type of component.

Components like activities are activated by an asynchronous message called intent, an intent is created with the intent object there are two types of intents explicit intents and implicit intents. When we explicitly specify an action for example, in our app opening up the article list activity on successful sign, in this action is performed with the help of an explicit intent. Of the other hand if you wanted to capture an image using the camera you would use an implicit intent to specify the intent to capture an image, and the android system will find the app like I do this for you either in your app or outsider app.

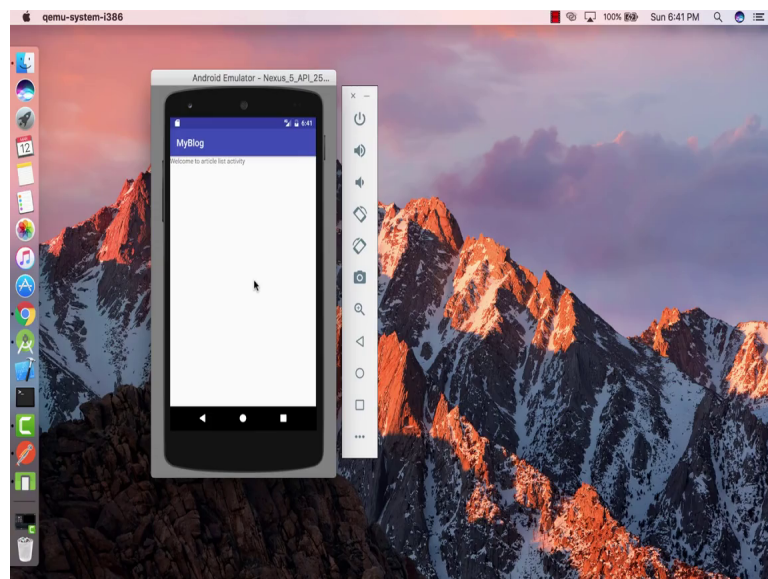
Let us create an intent to start our article list activity, if you pass in the context and then you pass in the class definition of the activity that you want to start, and then call the start activity method and pass the intent into it. Also let us go to our successful sign in hazard and replace the alert with navigator article list activity method all right.

(Refer Slide Time: 39:08)



So, now, let us just add a text here, just to see there is a (Refer Time: 39:17). So, now, let us run our app let us use imad and imad as a username password which we registered with earlier let us click on. So, that works our article list activity is going to show a list of articles that is used as added each item in this list is going to be shown inside a card view the article name written on it.

(Refer Slide Time: 39:32)

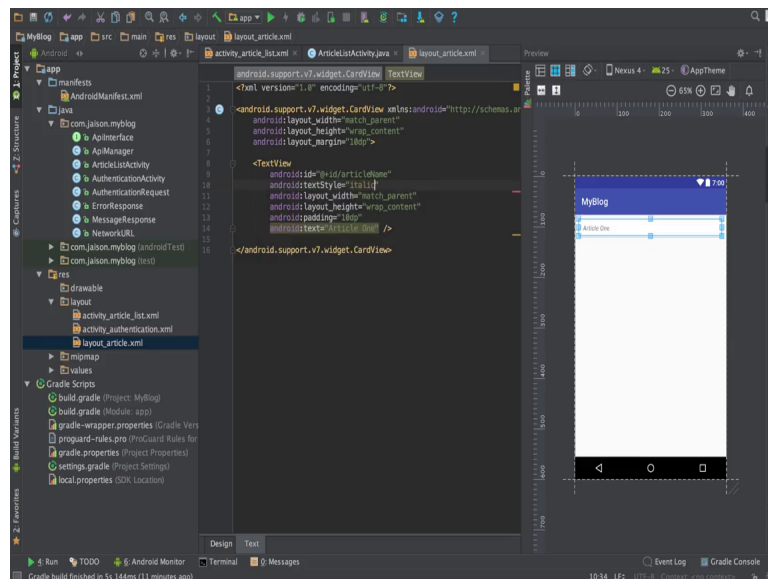


For this we need to integrate two libraries using gradle, one is called a recycler view and other one is called the card view. The card view was introduced as part of the android

material design which came after under lollipop. So, let us see how we can do that both the recycler view and the card view are the part of the android support library. So, what you do that is simply type in compile android 1.0 similarly let us add the card view and let us sync. So, that is done.

As you can see the these libraries are highlighted by android studio. So, what it means is that it is the android studios trying to say that there is a newer version of these libraries which are available which is 25.2.0. So, let us instead use the latest version and sync with (Refer Time: 40:41) again ok.

(Refer Slide Time: 40:54)



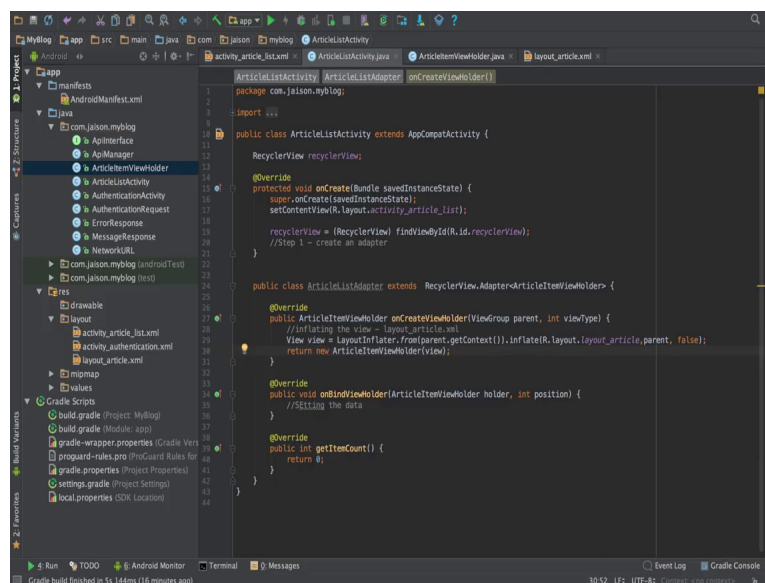
So, now that is done let us add the recycler view to our as you see showing a list let us also add an id to this, let us reference this in our activity now let us close this and let us go here.

So this is a recycler view recycler view before we start using the recycler view let us make the view for the item that is going to be repeated in the list, for that let us go to our resources could layout and let us make a new layout resource and I will call it layout underscore article underscore layout no let us call it layout underscore article it is a linear layout a vertical linear layout, as I add our card view to this card view and let us copy this you can take out this and then let us add this card view under this all right.

So, and let us change it is I to wrap content. So, now, we have linear layout which is vertical inside which we can add a (Refer Time: 42:16) we do not need a linear layout we just need a text view. So, we just now show the title let us add the width be match parent and the height be wrap content id be article name. So, now, let us put a dummy value of article one. Let us add a margin to this of 10 dp and let us add a pad into this of 10 dp is not that looks much better, we can add a style text, style and have it bold, bold let us put the color primary dark ok.

Insider view, view is a view holder pattern and so those of you who are interested to know what a view holder pattern is and I am going to add a link to an explanation of view holder part in the slides. So, now, let us create a view holders let us call it article item view holder which extends recycler view dot view holder android studio will prompt us to implement a (Refer Time: 43:52) methods which in this case is constructor which is called super and over here.

(Refer Slide Time: 43:55)



```
ArticleListActivity ArticleListAdapter onCreateViewHolder
package com.jason.myblog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.RecyclerView;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.recyclerview.widget.RecyclerView.Adapter;
import androidx.recyclerview.widget.RecyclerView.ViewHolder;

public class ArticleListActivity extends AppCompatActivity {
    RecyclerView recyclerView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_article_list);
        recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
        //Step 1 - create an adapter
    }

    public class ArticleListAdapter extends RecyclerView.Adapter<ArticleItemViewHolder> {

        @Override
        public ArticleItemViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
            //Inflating the view = layout_article_item
            View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.layout_article_parent, false);
            return new ArticleItemViewHolder(view);
        }

        @Override
        public void onBindViewHolder(ArticleItemViewHolder holder, int position) {
            //Setting the data
        }

        @Override
        public int getItemCount() {
            return 0;
        }
    }
}
```

Let us have a reference to the text view article name, and let us (Refer Time: 43:58) our text view here saying text view item view dot find view by id dot article name all right.

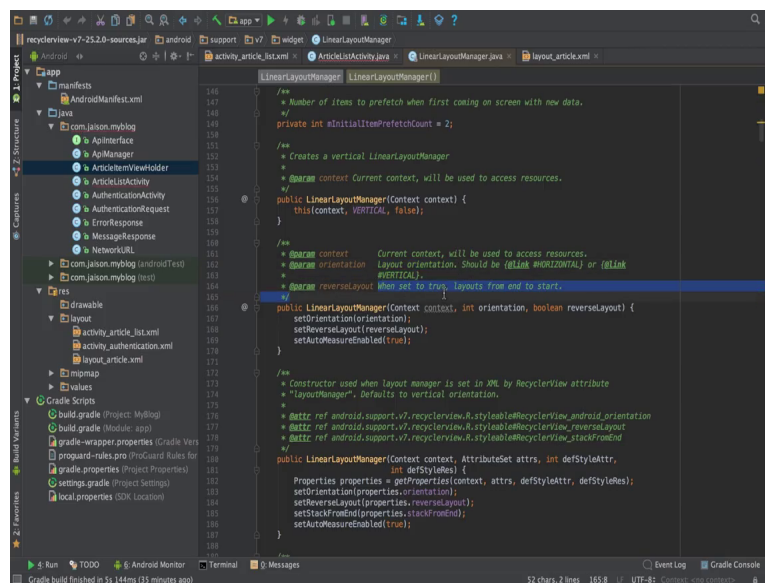
So, view holder is also very now let us look at how to implement a recycler view there are two steps to implementing a recycler view; step one is to create an adaptor let us do that let us call this article list adaptor which extends recycler view dot adaptor and as you can see it access a view holder which is the article item holder, and let us implement the

methods all right and let us have a look at all these methods. The first one is on create view holder which basically ask us to return an object of type article item view holder this is where we inflate the layout that which is created.

Next one is on time view holder this method is called after the view has been bound to the list, this is that we would be setting the data and this is where inflating the view which would be layout underscore article dot xml and finally, you get item count is basically you have to tell the recycler view how many items we have in this list all right. So, let us see how we do this for this we set view let us import view equal layout inflater from parent dot get context inflate r dot layout dot layout underscore article, and we return a article item view holder all right. So, basically we are passing the inflated view to the article item view holder which is then getting a reference to r to the which is used inside the view.

Next is unbind view holder, this is how we set the data before we do that let us make some mock details. So, let us create a list of articles and let us say holder dot article name dot set text would be article list dot get, article list of position. Online view holder gives us a view holder as an argument along with the exposition. So, we are basically setting the name of the article based on a position also let us return the count as the length of article list all right. So, our adaptor has been set let us set this to our recycler view set adaptor new article list of adaptor all right step two is to set a layout manager.

(Refer Slide Time: 47:10)



```
LinearLayoutManager(LinearLayoutManager())
    + Number of items to prefetch when first coming on screen with new data.
    +
    private int mInitialListPrefetchCount = 2;

    /**
     * Creates a vertical LinearLayoutManager
     *
     * @param context Current context, will be used to access resources.
     */
    public LinearLayoutManager(Context context) {
        this(context, VERTICAL, false);
    }

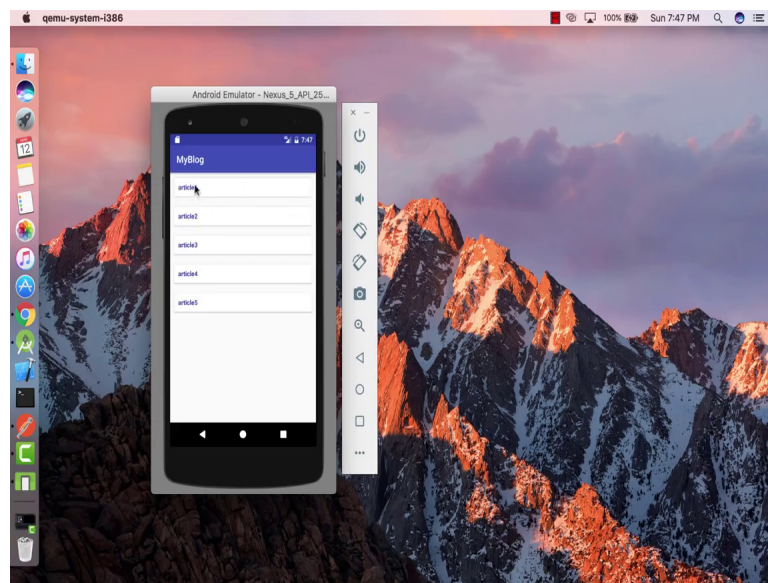
    /**
     * @param context Current context, will be used to access resources.
     * @param orientation Layout orientation, should be {@link #HORIZONTAL} or {@link
     * #VERTICAL}.
     * @param reverseLayout When set to true, layouts from end to start.
     */
    public LinearLayoutManager(Context context, int orientation, boolean reverseLayout) {
        setOrientation(orientation);
        setReverseLayout(reverseLayout);
        setAutoMeasureEnabled(true);
    }

    /**
     * Constructor used when layout manager is set in XML by RecyclerView attribute
     * "layoutManager". Defaults to vertical orientation.
     *
     * @attr ref android.support.v7.recyclerview.R.styleable#RecyclerView_android_orientation
     * @attr ref android.support.v7.recyclerview.R.styleable#RecyclerView_reverseLayout
     * @attr ref android.support.v7.recyclerview.R.styleable#RecyclerView_stackFromEnd
     */
    public LinearLayoutManager(Context context, AttributeSet attrs, int defStyleAttr,
        int defStyleRes) {
        Properties properties = getProperties(context, attrs, defStyleAttr, defStyleRes);
        setOrientation(properties.orientation);
        setReverseLayout(properties.reverseLayout);
        setStackFromEnd(properties.stackFromEnd);
        setAutoMeasureEnabled(true);
    }
}
```

So, there are different type of layout manager, but for this tutorial we are going to use linear layout manager let us say what it needs linear layout manager takes in two parameters when your; as it constructor one is the context the other one is the orientation and a Boolean flag indicating whether it is a reverse layout. So, as you can see the context and the current context that will be used the layout orientation can be two it can be either horizontal or vertical in this case we will be using vertical, and the reverse layout basically when it is set to true the layout starts from the end to start it will be listed in the opposite of order in which it is given.

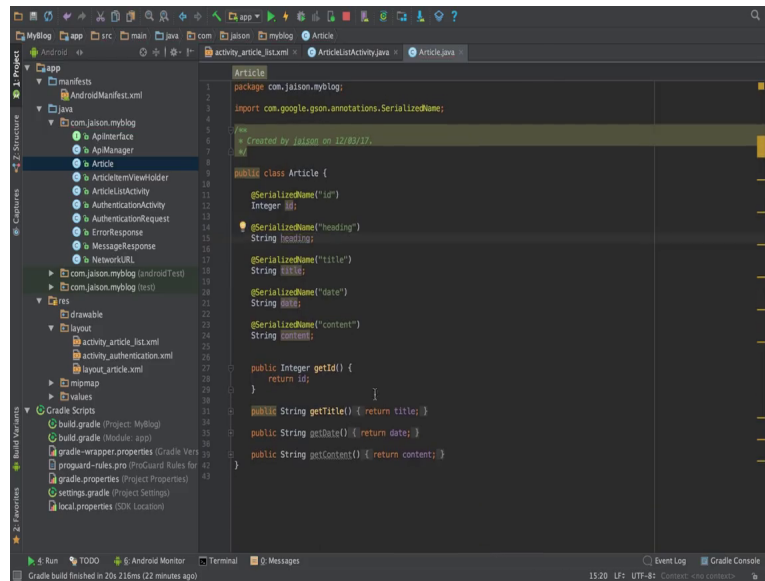
In our case we want the reverse layout to be false and the young orientation to be vertical. So, we are going to use the single constructor single argument constructor and we are simply to going to pass the context all right.

(Refer Slide Time: 48:24)



Now, let us run our app as you can see we have a list which stands from article one to article 5 which is shown inside a card and with the font and title that we have seen that.

(Refer Slide Time: 48:28)

A screenshot of an IDE showing the code for an Android application. The left sidebar shows a project structure with folders for 'app', 'src', 'main', 'java', 'com', 'jason', 'myblog', and 'Article'. The main editor area displays the code for the 'Article' class. The code includes package declarations, imports for Gson, and a class definition with fields for id, heading, title, date, and content, each annotated with @SerializedName. It also has getter methods for id, title, date, and content.

```
1 package com.jason.myblog;
2
3 import com.google.gson.annotations.SerializedName;
4
5
6
7
8
9
10
11 @SerializedName("id")
12 Integer id;
13
14 @SerializedName("heading")
15 String heading;
16
17 @SerializedName("title")
18 String title;
19
20 @SerializedName("date")
21 String date;
22
23 @SerializedName("content")
24 String content;
25
26 public Integer getId() {
27     return id;
28 }
29
30 public String getTitle() { return title; }
31
32 public String getDate() { return date; }
33
34 public String getContent() { return content; }
35 }
```

Let us now make use of the get article API call that we created for our web app, I have converted the response of the API call to an array of json objects of type article which we have id title heading date and content assets json keys, let us make a (Refer Time: 48:42) call it article I have already created one let us with get us for each of the value, and now let us go ahead of API interface and create this API call.

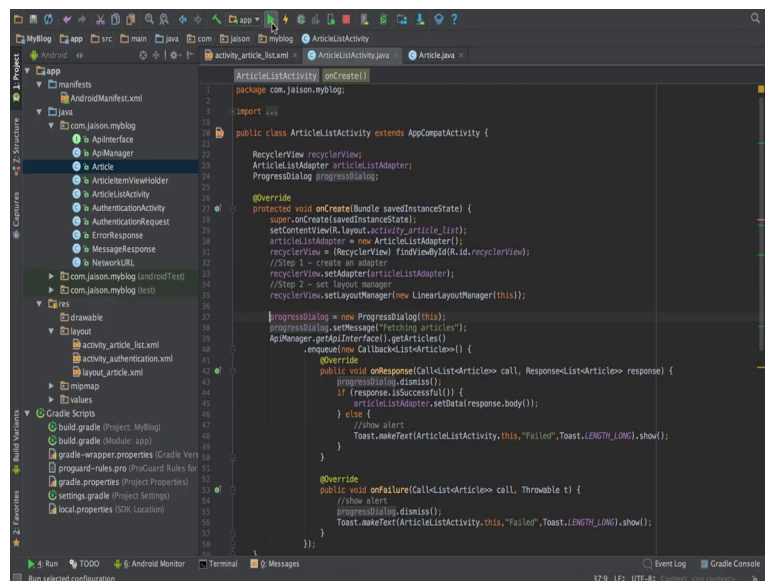
This is a get request also let us add this to our network q r let us call it get underscore articles among here get articles, the call would return the list of article and let us call it get articles. There is no body to be sent or let us say now that we have created a API call let us make using that nr article list activity, let us say API manager get API interface dot get articles dot enqueue new call back. There are two things that we can do now we get either keep the reference of the adaptor that we have set for the inceptor view and update it is value inside the adaptor and refresh the adaptor or simply set the adaptor after the responses come back.

A good way to do this is to hold a reference to the adaptor and reset the value of the adaptor. So, let us do that. So, call article list adaptor to list adaptor let us create a new one and let us toss that here let us then create another method in which article list adaptor called set data, which can be a list of article, let us delete this and let us do article list and let us do this recruit the data and yeah size and let us do a dot get position and let show get title. So, again start something on your article, I think there is a heading to be shown

as well let us add that and let us add that (Refer Time: 51:42) this as well and let us show the heading here instead of the title all right. So, I think that is done, now here let us show a progress indicator and hide them here all right.

So, now that we have the response as always you have to check if the response is successful if it is successful.

(Refer Slide Time: 52:24)



```
package com.jason.myblog;

import java.util.List;

import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.RecyclerView;
import androidx.recyclerview.widget.RecyclerView.Adapter;
import androidx.recyclerview.widget.RecyclerView.ViewHolder;
import androidx.appcompat.app.AlertDialog;

public class ArticleListActivity extends AppCompatActivity {

    RecyclerView recyclerView;
    ArticleListAdapter articleListAdapter;
    ProgressDialog progressDialog;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_article_list);
        articleListAdapter = new ArticleListAdapter();
        recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
        //Step 1 - create an adapter
        recyclerView.setAdapter(articleListAdapter);
        //Step 2 - set layout manager
        recyclerView.setLayoutManager(new LinearLayoutManager(this));

        progressDialog = new ProgressDialog(this);
        progressDialog.setMessage("Fetching articles");
        ApiManager.getApiInterface().getArticles()
            .enqueue(new Callback<List<Article>() {

                @Override
                public void onResponse(Call<List<Article> call, Response<List<Article> response) {
                    progressDialog.dismiss();
                    if (response.isSuccessful()) {
                        articleListAdapter.setData(response.body());
                    } else {
                        //show alert
                        Toast.makeText(ArticleListActivity.this, "Failed", Toast.LENGTH_LONG).show();
                    }
                }

                @Override
                public void onFailure(Call<List<Article> call, Throwable t) {
                    //show alert
                    progressDialog.dismiss();
                    Toast.makeText(ArticleListActivity.this, "Failed", Toast.LENGTH_LONG).show();
                }
            });
    }
}
```

Let us set the data to be response dot body and in case it is not let us show alert let us show an alert. To be quick I am just going to add a toast here let us just add the progress indicator as well before we run the app (Refer Time: 52:51) dialog progress dialog and progress dialog dot set message please wait or let us call it fetching articles and dismiss or let us instead make it a (Refer Time: 53:17) variable all right.

So, that is done, now let us run the app and see now again oh we forgot to show the progress dialog and so, we forgot two things here one thing is to show the progress dialog which would be after we set the data we need to call an another method called notify data set changed, to notify the adaptor and reload the recycler view let runs the app again.

Let us sign in. So, now, we have articles 2 1 shown the order is such because this is the order in which we receive the data from our API call next thing that we have to do is handle the click on these cards, and also show ripple effect on the click. So, for that let us

head over to our layout underscore article first let us click the card view and id let us call it card view and then for the ripple effect let us say android foreground and let us call the selectable item background that is a one thing to be noted here is that the dimensions and the string that we are using here have to be replaced by this thing resources ok.

And now let us access that in our view holder, now that is done get back to on bind view holder method holder dot card view dot set let us say the to this and then simply show a toast and the card view is clicked, let us show this all right. So, let us runs our app and see this in action. One thing that you can do to avoid sign in each time is to go back to your authentication activity and simply call the navigate to article this activity over in your own create method since our get article is not an independent log in let us now see how (Refer Time: 55:58) function works all right.

(Refer Slide Time: 56:08)

Recap

- Mocking network calls using Async Tasks
- Using Retrofit and Gson for Network APIs
- Intent to start another activity
- RecyclerView and CardView

As you can see the toast gives the article name and the ripple effect it is also shown, in this module we learnt how to mock network API calls using Async tasks, we used retrofit and Gson for making an network API calls we also.

(Refer Slide Time: 56:26)

Tasks

- Start ArticleDetailActivity on article click.
- Back button on Toolbar for ArticleDetailActivity.
- Send data from one activity to another (hint : Parcelable and Serializable).
- Logout button.
- Code clean up.

Complete code at <https://github.com/jaisontj/imad-android-app>

Now, know how to start another activity using intent and finally, we used the recycler view and card view in our app. Here there is a task assigned to you. Firstly, you need to create a new activity called article detailed activity and show the detailed view of the article being clicked by the user, this activity also needs to have a back button on its toolbar on clicking which the user should be taken back to the article list activity, since we are making API call to fetch the detail in the article list activity, we need to figure out the way to send the information of the selected article to the article detailed activity, and finally, implement a logout button in the article list activity on its toolbar. Also do some code clean up so that the repetitive code is avoided and everything is modularized the complete code to this app is in the repository which is provided in the slides.

(Refer Slide Time: 57:21)

Not included but important to know

- Fragments
- ViewPager
- NavigationView (Hamburger Icon)
- Shared Preferences
- Content Providers
- Broadcast receivers
- Services
- ActionBar
- Programming Design Patterns - MVC, MVP, MVVM etc
- Animations and Transitions
- Constraint Layout

Some of the things that we have not included in this module, but are important for android application developers are fragments view pagers the navigation view which is the hamburger icon that comes on left hand side top corner. Shared preferences which should be used to persist minimal data about the user offline content providers broadcast receivers services action bar programming design patterns like MVC, MVP, MVVM etcetera animations and transitions and constraint layout.

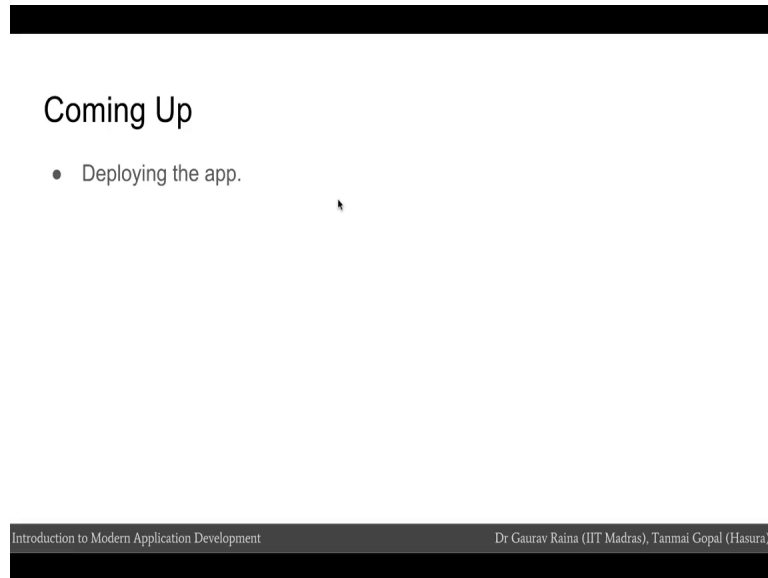
(Refer Slide Time: 57:55)

Reading Material

- ViewHolder Pattern - <https://willowtreeapps.com/ideas/android-fundamentals-working-with-the-recyclerview-viewholder-pattern/>
- developer.android.com

For those for you interested in understanding the recycler view and the holder pattern a little better here is a link to it, apart from that developer android dot com should give you the view information about everything.

(Refer Slide Time: 58:09)



That you need to know to develop a very good android application in the next we will we will finally, deploy this blog app to the Google play store.