Central University of Technology, Free State

ASSIGNMENT 2 – 2020

DUE DATE: 13 July 2020 **GROUP ASSIGNMENT** **SUBJECT:** TECHNICAL PROGRAMMING II and TECHNICAL PROGRAMMING IIIA	**ISSUED ON: 26 June 2020** **CODE:** TPG20AB / TPG316C

INSTRUCTIONAL PROGRAMME: N. DIPLOMA IN IT and DIPLOMA IN IT

INSTRUCTIONAL PROGRAMME CODE: EINDSD / EXNDIS and DP_ITC / EX_ITC

INSTRUCTIONS TO CANDIDATES:

1. Please submit your assignment documents in a folder titled your Group Name **ONLY!! E.g. funky_coders/your project folder**. In the folder, provide your signed APK file and project folder as well as a document with the surnames and student numbers of all group members (max 5 per group).
2. **Duplicate code or cheating** will be **penalised** as per CUT rules and regulations provided in your study guide.
3. **All submissions** to be made via the link provided on **e-Thuto**.
4. **No late assignments** will be accepted.
5. It is **your responsibility** to sort out any submission problems with your lecturer **well in advance before the submission time**.
6. Only **running/executing code** will be graded, and you may use **Azure Repos or Git** for version control and remote collaboration.
7. You must make use of **strings.xml, styles.xml, dimens.xml** and **colors.xml** files to reference strings, styles, dimens and colors.
8. Use defensive programming techniques, all **unhandled exceptions** will be penalized.

ASSESSORS: MR. JG JURRIUS, MR. MHLABA

Due date: 13 July 2020 **Total marks:** 100

BACKGROUND OF THE ASSIGNMENT

This is a Job Management application that provides a simulation of a real-world application with functions such as profile registration, account login, posting job advert and listing of job vacancy and giving users an option to apply for or withdraw a job application.

Please watch the video demonstration first before reading this document.
NB: You may use Azure Repos or Git for version control and remote collaboration. Use API 29 to complete this task and always adhere to good programming practices.

SPECIFICATIONS

1. Create a new Android App called JobManagement and add the following packages inside your main project folder **activities**, **app_utilities**, **data_models**, **db_operations**, **db_repositories**. Inside package db_repositories nest the following sub-packages **job_advert**, **job_application** and **job_profile** as shown in figure 1 below. Data for all your dropdown views can be accessed here and must be placed in the values folder. Add a class called **AppUtility** inside package app_utilities and this class extends android's Application class. This is where all the reusable functions will reside. Sample code to design the views to look like the example app in the video can be accessed here
2. Locate package **data_models** and create a **JobProfile** class that will hold data for every user's job profile. The **JobProfile** class has fields below and must be annotated with appropriate attributes to make it a compatible room api entity.

Field Name	Data Type
id: PK and autogenerated	long
email: unique	String
password	String
name	String
surname	String
cellphone	String
identityNumber: unique	String
qualification	String
education	String

3. Locate package **data_models** and create a **JobAdvert** class that will hold data for every job advertisement. The JobAdvert class has fields below and must be annotated with appropriate attributes to make it a compatible room api entity.

Field Name	Data Type
id: PK and autogenerated	String
jobTitle	String
jobSalary	String
jobLocation	String
appointmentType	String
jobPosition	String
jobCompany	String
jobDescription	String
isLicense	boolean
jobQualification	String

4. Locate package **data_models** and create a **JobApplication** class that will hold data for every job application done by the user. One user can apply for one or more job advertisements and one job advertisement can have one or more users applying for it. A user can not apply more than once for the same job, so create a composite primary key using **jobId** and **profileId**. The **JobApplication** class has fields below and must be annotated with appropriate attributes to make it a compatible room api entity.

Field Name	Data Type
jobId: PK - FK -> parent: JobAdvert	long
profileId: PK- FK -> parent: JobProfile	long

   - Please note that if the user deletes or updates an instance of JobAdvert or JobProfile, all job application records referencing either instance must be deleted from or updated in the database. This should be achieved by annotating your JobApplication model with the correct room api attributes.

5. Locate package **db_operations** and create **GenericDao** interface remember to add <T> to the interface to make it generic. The interface must have abstract methods to **insert**, **delete** and **update** entities, and these methods have a generic type parameter. Also add **JobProfileDao, JobApplicationDao, JobAdvertDao, AppDatabase** and **Connections** to the same package. All DAO interfaces extend **GenericDao** giving access to insert, delete and update. The name of the database is **db_job_management**. Add **@Query** attribute to

respective abstract methods in each DAO to provide the query that executes against the database.

JobProfileDao Abstract Methods	JobAdvertDao Abstract Methods
void delete(long id);	void delete(long id);
List<JobProfile> getAllJobProfiles();	List<JobAdvert> getAllJobAdverts();
JobProfile getJobProfileById(long id);	JobAdvert getJobAdvertById(long id);
JobProfile getJobProfileByEmail(String email);	

JobApplicationDao Abstract Methods
List<JobApplication> getAllJobApplications();
List<JobApplication> getUserApplications(long profileId);
void delete(long jobId, long applicantId);
JobApplication findByJobIdAndUserId(long jobId, long profileId);
List<JobAdvert> getUserAppliedJobAdverts(long profileId);

6. Locate package **db_repositories** and create **AsyncTaskCallback** interface and add<T> to the interface to make it generic. This interface has two void abstract methods **onSuccess** and **onException**. onSuccess method has one generic type parameter and onException has one Exception type parameter. Use the table below to add the respective async classes to the correct package nested inside db_repositories. Do not pass a Context object to any of these async classes, rather pass an instance of DAO to avoid memory leaks. All data is to be properly validated prior being passed to the respective query.

job_profile Package	job_advert Package	job_application Package
InsertJobProfileAsync	InsertJobAdvertAsync	InsertJobApplicationAsync
GetAllJobProfilesAsync	GetAllJobAdvertsAsync	GetAllApplicationsAsync
UpdateJobProfileAsync	UpdateJobAdvertAsync	GetUserApplicationsAsync
FindJobProfileAsync	FindJobAdvertAsync	GetUserJobAdvertsAsync
DeleteJobProfileAsync	DeleteJobAdvertAsync	FindJobApplicationAsync
		DeleteJobApplicationAsync

7. Rename the **MainActivity** to **LoginActivity**. For this screen, create 2 TextInputEditText widgets to accept the user's registered email address and password. Create a CheckBox or SwitchMaterial widget to allow the user to choose whether he/she wants to be kept logged in or not. Create two buttons to allow the user to login or register. Make use of Button views with an icon as shown in the video. Try to get the layout as close as possible to the layout in the video.
8. Add a **ProfileActivity** that will simulate a user profile on the application. **ProfileActivity** layout must have appropriate widgets or views: one for each field in the JobProfile class, use the video as guidance. Also include a button to submit the user's profile. ***Make sure all***

fields are properly validated soon after losing focus and show appropriate error upon detecting invalid input. If an error was shown before, make sure it gets cleared when the view receives focus. You are required to use Regular Expressions API to do validations for *email*, *password*, *cellphone* and *identity number*, learn more about this here

- An email address must be properly validated to ensure it contains (*account name, @ symbol and domain*) account@example.com and every email must be unique in the database. All these validations must take place once the email view loses focus, if validation fails, display either *Email already exists, or Email is not properly formatted*.
- A password must be at least 8 characters long with at least one lowercase, uppercase, digit and alphanumeric character else *Password too simple* error must be shown. Also make sure that the user enters data before he/she can submit the profile and perform one last validation of all fields. Try to get the layout to look as close as possible to the one in the video.

9. Add an adapter class called **DataAdapter** to the app_utility package and all other classes you may need should be added to this package. Configure your adapter class to initialise all views contained in the row_layout file. The layout file must be designed to resemble figure 2 shown below and all views are to be added to a CardView, and the image drawable file can be downloaded here. Do not pass an instance of Context class to your adapter.
   - Add an interface called **CardViewButtonClickListener** inside app_utility package. This interface is responsible for executing actions highlighted in figure 2 below and has the abstract methods as shown below. You should determine your own **method parameters**, but do not load data from the database to pass to these methods.
   - This interface will be implemented by **ListAdvertActivity which** you must register for click events inside your **DataAdapter**.

CardViewButtonClickListener Methods	Purpose
void onEditAdvertClick	Opens JobAdvertActivity and edits its data and saves it to the database. The data to be displayed **should not** be fetched from the database.
void onViewAdvertClick	Opens JobDetailActivity and displays its data, the data to be displayed **should not** be fetched from the database.

void onDeleteAdvertClick	Asynchronously deletes JobAdvert instance from database and refreshes the recycler view.
void onJobApplicationClick	Asynchronously inserts JobApplication instance into the database and refreshes the recycler view to indicate that the user has applied for this advert. This is achieved by updating the icon from this ▯ to this 🖤

10. Add ListAdvertActivity, this will be the main landing page after the user has successfully logged into the system. The activity fetches JobAdvert records from the database and display them on a **RecyclerView**. The objective is to read and display job adverts from the database and allow users to apply for jobs, create or edit job adverts, edit user profile and withdraw applications. This activity also implements **CardViewButtonClickListener** interface.
    - Add a menu file called app_menu.xml inside the menu folder under your resources and design it to resemble figure 3 below. The menu will help navigate around different application components. The functionality of each menu item is described below.

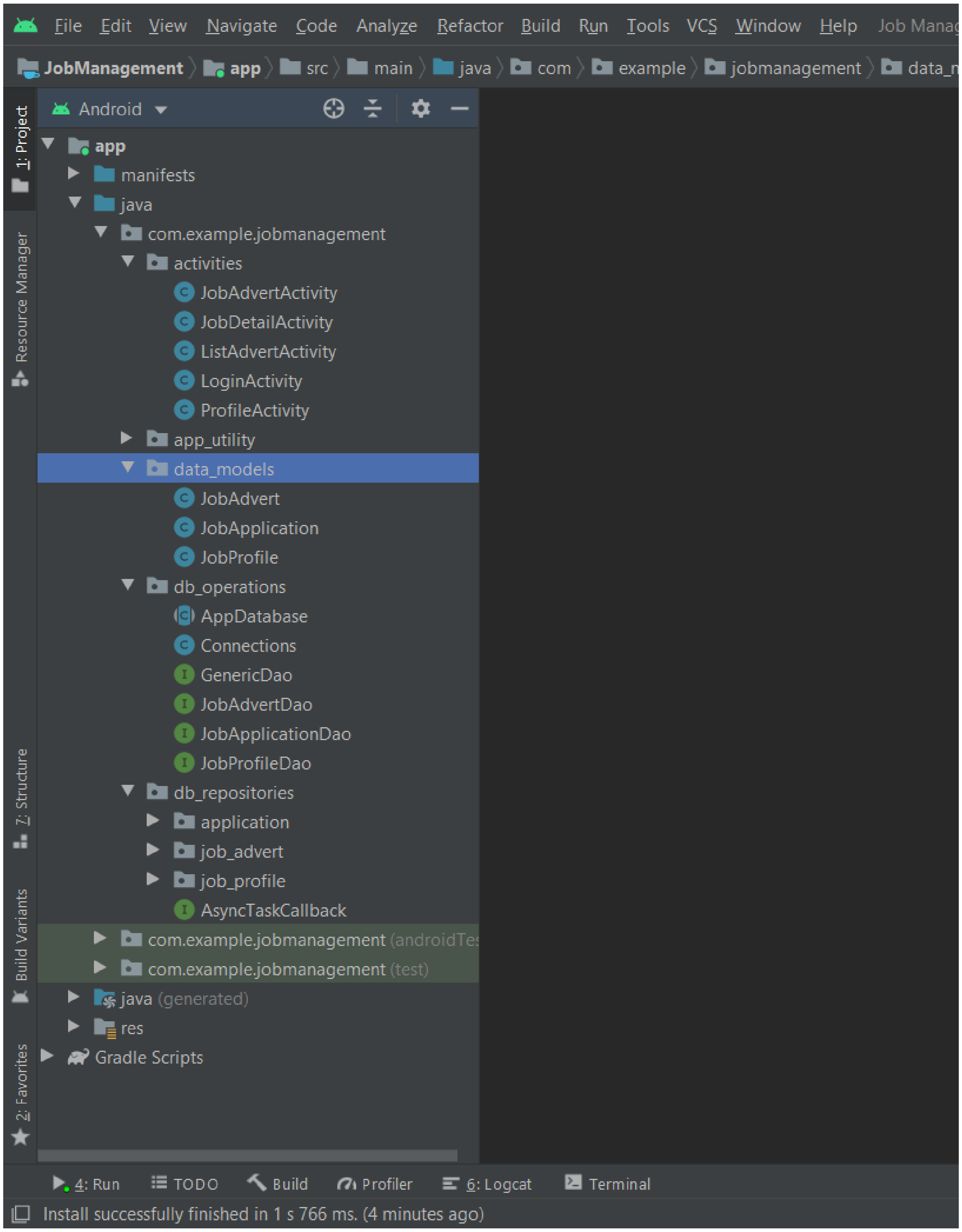
Menu Function	**Purpose**
▯	Clicking this menu item should fetch only applications that the user has applied for and display them in the recycler view and the recycler view should show the apply button with 🖤 background. Clicking apply button with 🖤 background should pop up an application withdrawal dialog as shown in the video.
⊕	Clicking this menu item should open JobAdvertActivity to allow the user to add a new advert record to the database. If the insertion was successful, the recycler view must display this advert.
Edit Profile	Clicking this menu item should open the ProfileActivity that would in turn display data in edit mode.
Logout	Clicking this menu item should clear the SharedPreference and close **ListAdvertActivity** and redirect user to **LoginActivity**.

11. Add **JobAdvertActivity**, that will simulate a job vacancy being advertised for users to apply. **JobAdvertActivity** layout must have appropriate widgets or views: one for each field in the JobAdvert class. Use the video as guidance. Also include a button to submit a job advert and perform one last validation of all fields.

12. Every time the app is opened, the user's preference should be tested. If the preference is set to stay logged in, the app must take the previous authenticated user to the **ListAdvertActivity** activity immediately and should not show the Login screen. If the preference is set not to stay logged in, the app must land the user on the Login screen. The sharedPreference can only be used to save three pieces of data (**userEmail**, **stayloggedIn** and **profileId**) and must be cleared when the user clicks logout.
13. Show appropriate **Custom Toast** messages as shown in the video demonstration. Reuse the **showToast method** in the Application class throughout the application to show toasts. Add code to read, edit and clear the sharedPreference file in **AppUtility** for reusability. All reusable code must be refactored into static modules or functions that reside in this class. Register **AppUtility** in AndroidManifest.xml and use the onCreate method to initialise your **SharedPreferences** instance.

DEMONSTRATION / SCREENSHOTS

See the video demonstration of the operation of the application.



```
▼ app
    ▶ manifests
    ▼ java
        ▼ com.example.jobmanagement
            ▼ activities
                JobAdvertActivity
                JobDetailActivity
                ListAdvertActivity
                LoginActivity
                ProfileActivity
            ▶ app_utility
            ▼ data_models
                JobAdvert
                JobApplication
                JobProfile
            ▼ db_operations
                AppDatabase
                Connections
                GenericDao
                JobAdvertDao
                JobApplicationDao
                JobProfileDao
            ▼ db_repositories
                ▶ application
                ▶ job_advert
                ▶ job_profile
                AsyncTaskCallback
        ▶ com.example.jobmanagement (androidTes
        ▶ com.example.jobmanagement (test)
    ▶ java (generated)
    ▶ res
▶ Gradle Scripts
```

Install successfully finished in 1 s 766 ms. (4 minutes ago)

Figure 1 : Project Structure

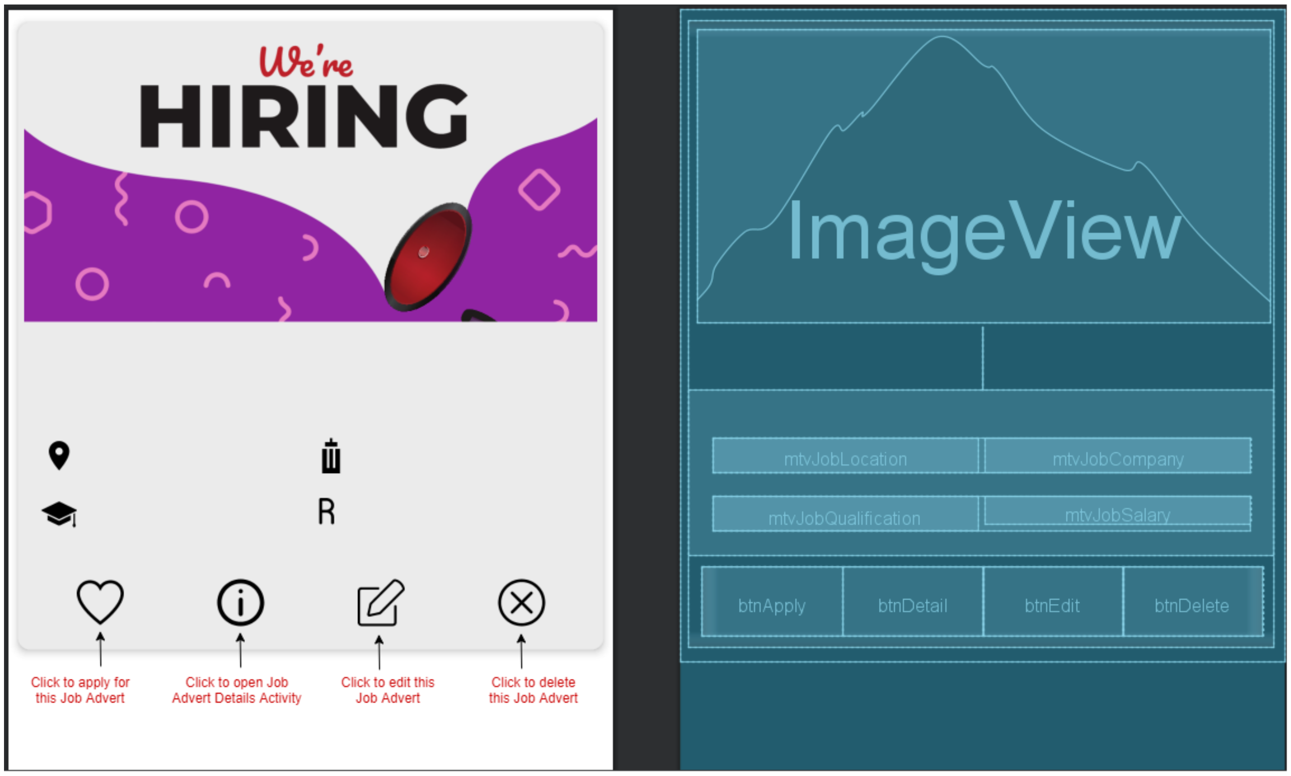

We're
HIRING
R
Click to apply for this Job Advert
Click to open Job Advert Details Activity
Click to edit this Job Advert
Click to delete this Job Advert
ImageView
mtvJobLocation
mtvJobCompany
mtvJobQualification
mtvJobSalary
btnApply
btnDetail
btnEdit
btnDelete

Figure 2: CardView Design

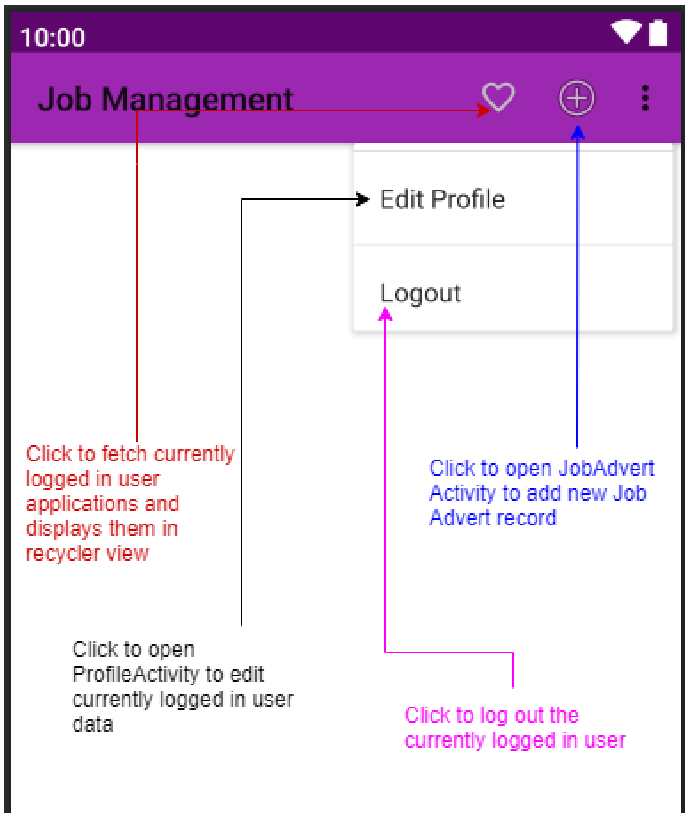

10:00
Job Management
Edit Profile
Logout
Click to fetch currently logged in user applications and displays them in recycler view
Click to open JobAdvert Activity to add new Job Advert record
Click to open ProfileActivity to edit currently logged in user data
Click to log out the currently logged in user

Figure 3: App Menu

GOOD LUCK!

----------End of Assignment 2---------

TOTAL MARKS = 100