

UNIVERSITI TUNKU ABDUL RAHMAN
Faculty of Information and Communication Technology



UCCD3223 Mobile Applications Development
(January 2022 Trimester)

Individual Practical Assignment

Name	Tan Xi En
Student ID	19ACB04098
Course	Computer Science
Practical Group	P12
Lecturer	Tan Chiang Kang

Marking scheme	Marks	Remarks
Correctness of output	× 3	
Design of UI	× 2.5	
User Friendliness	× 2.5	
Neat Program Documentation		
Report Format		
TOTAL		

Introduction

With the advancement of modern technology, digital transformation is altering every aspect of how today's business operate and complete. Internet services are being introduced at every second, and these internet services requires a certain level of security to prevent unauthorized access to these services. Password is used as a mean to protect user security and prevent unauthorized access. However, user-created password may either be too simple or too repetitive, as users may use the same password for each internet service. To combat this problem, password managers are introduced to manage password for different application. Therefore, for this assignment, I plan to propose a password manager that stores, generate and manage multiple passwords for multiple applications.

Problem Statement

1. Password do not have strong security

The main issues with a lot of passwords are that they are too simple. User generated password usually comprises of only number and alphabets. These passwords are too simple to be cracked as they have a low number of combinations. For a security tool used to prevent unauthorized access to important services, passwords need to meet a certain degree of security. Therefore, our password manager will generate strong combinations of password with certain length.

2. Some Passwords are repetitive

The main issue with user defined passwords are that user tend to use only one password for each internet service. This is because it is much easier memorising one passwords, rather multiple unique passwords for each internet service. A user should ideally have unique passwords for all of one's services to reduce the impact of compromised passwords. The implied effect is that if one of the services is compromised, none of the user's other services are.

App Functionality

While there are many instances of password managers in Google play Store (e.g. KeePass, DashLane etc.), there are universally agreed functionalities for a password manager. A basic password manager should be able to:


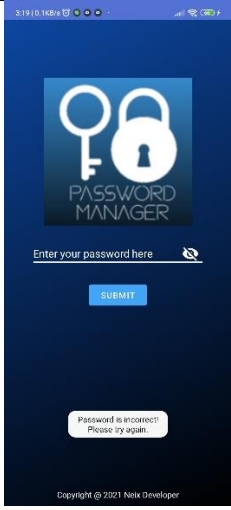
- Generate Secure password with multiple character sets and variable length
- Able to add, update, delete password for multiple websites
- Encrypt password when stored into database, decrypt password when retrieved from database.
- Search and sort list of passwords.
- Export Database as Backup

To store and retrieve password, we need to create a local database within our android phone. Normally, for application development, developers would use SQLite database, as it is provided in all android phone. However, for ease of development, I've decided to use Room Database to implement our database functionality. This is because the code are much cleaner and it is much easier to debug using Room Database.

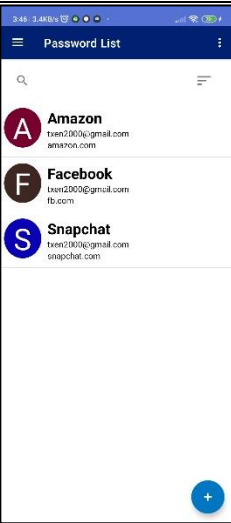
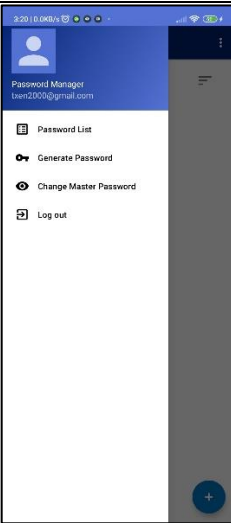
Secondly, to show our list of password, I have also decided to implement RecyclerView adapter. This is because RecyclerView have its own view model and life cycle that constantly recycle old view and reuse for new elements, hence the name recycle view. Recycle view allows our android app to run smoother, by constantly recycle old elements and remove unnecessary view.

To implement these functionalities, I have designed multiple activities and fragment that accommodates these functions. The UIs that I have designed are, activity_main.xml, fragment_gen_pwd.xml, fragment_change_mas_pwd.xml, activity_login.xml, activity_add_pwd.xml, activity_edit_pwd.xml.

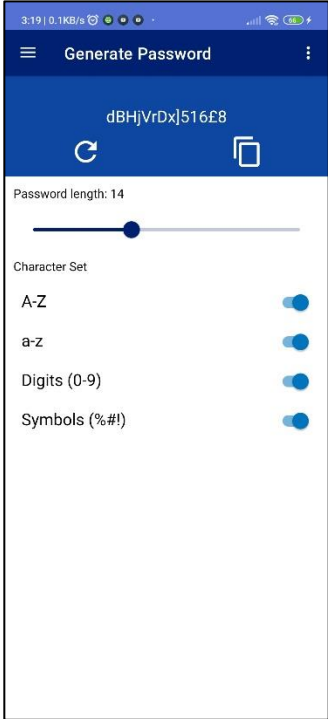
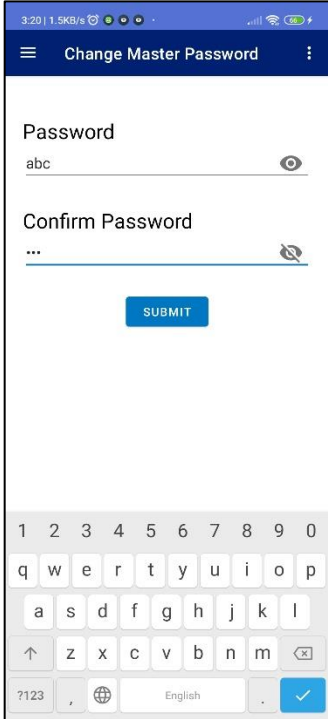
Login Page

	
<p>Default Login Page for Password Manager</p> <p>Have eye icon button to show or hide master password.</p>	<p>Shows toast message if master password is incorrect.</p>

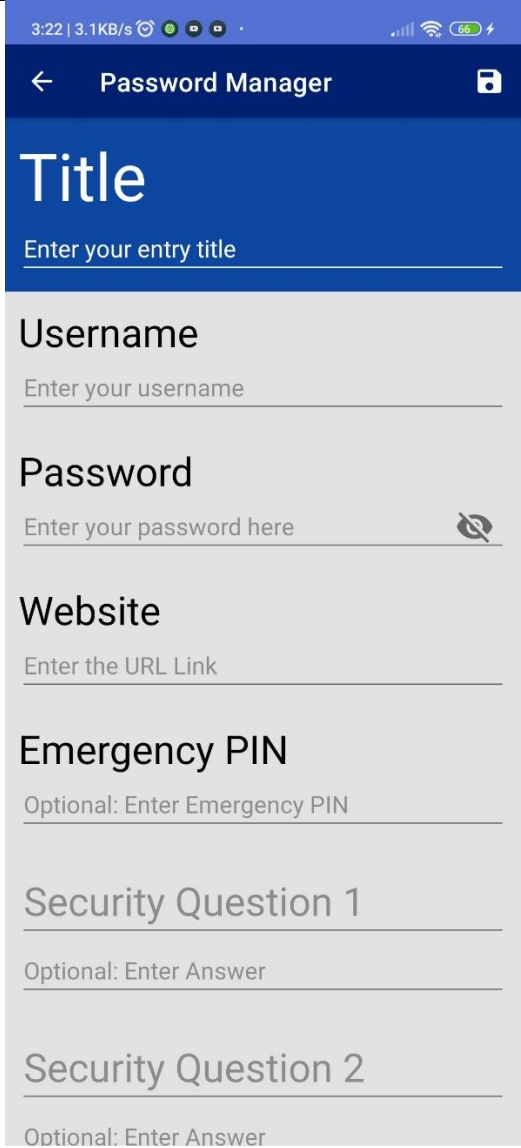
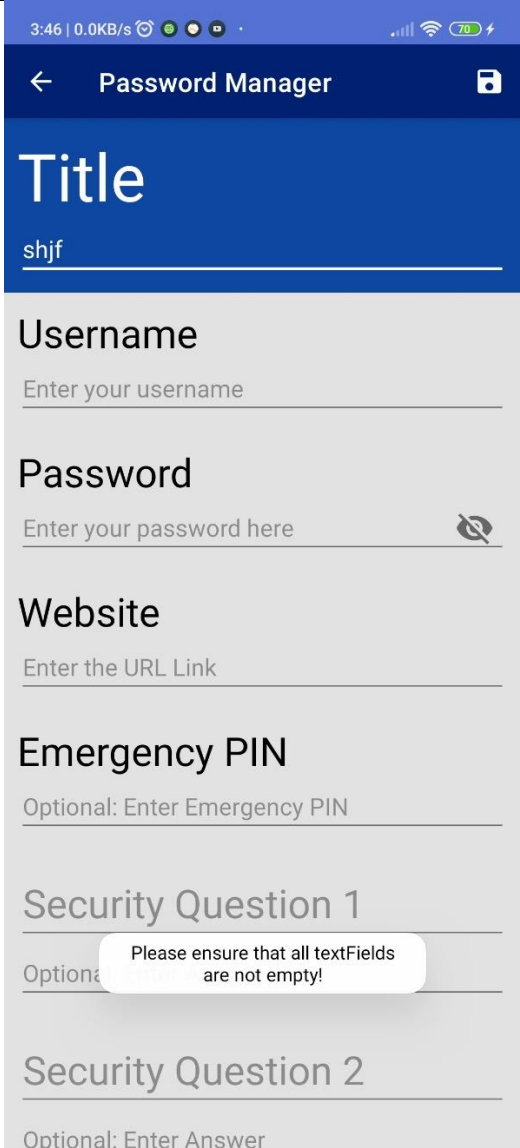
Main Page

	
<p>Default Homepage for Password Manager.</p> <p>Shows list of password that users can edit.</p> <p>Shows Floating Action Button that users can press to add new password</p> <p>Use Recycle View Adapter to show List of password.</p>	<p>Custom Navigation Drawer on the right.</p> <p>When user press on the navigation icon, the drawer will pop up and allow user to navigate to different sections of the application</p>

Generate Password Page and Change Master Password Page

Generate Password Page	Change Master Password Page
	
<p>Helps user to auto generate password based on selected character Sets.</p> <p>Offers 4 different types of Character set, that is Uppercase Characters, Lowercase Characters, Digit Characters, Symbol Characters</p> <p>Allow User to generate password on variable length</p> <p>Have copy to clipboard button that user may press to copy to clipboard</p> <p>Have reset button that user may press to generate a new password</p>	<p>Allow User to change Master Password</p> <p>Shows incorrect toast message if password and confirm password are not the same</p> <p>Shows success toast message if password and confirm password are the same.</p> <p>Have eye icon to hide or show password</p>

Add Password Page

	
<p>The following activity comprises of the basic property of each password. Title, Username, Password and Website.</p> <p>If title, username, passwords and website are not filled in, an incorrect toast message will be shown to act as validation.</p> <p>Optional field such as Emergency PIN, security questions are provided. User may choose to fill in.</p> <p>Have eye icon to hide or show password field</p> <p>Have save Icon to save password</p>	

Edit Password Page

--	--	--

The following edit page provides the list of field based on the password element selected.

There are three variations of Edit Page. This is based on the number of optional fields that the user have entered. For example, “Facebook” does not contain any optional field, whereas “Snapchat” and “Amazon” does.

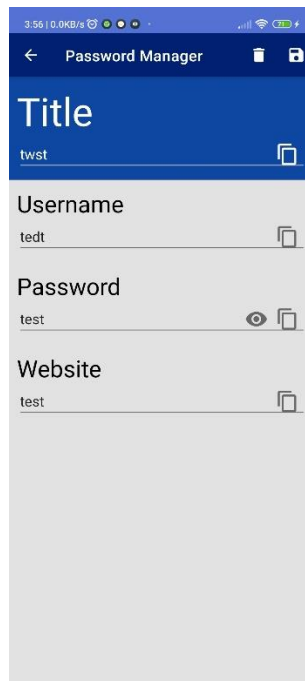
Each of the field have a copy clipboard button, that copies the respective field into the clipboard

Each password field have a show / hide eye icon, that shows or hide password.

Each edit page have a delete and save button function, that performs its respective function.

Encrypted Password

One of the main priorities of a password function is its ability to encrypt password. Encrypted password is important because in the scenario the original database is hacked, hackers would be unable to get the password, as they are not stored in plaintext. To achieve this, we have decided to use AES encryption with a randomly generated key that is stored within the user shared preference file. To demonstrate the functionality, we have provided two screenshot.



Using app inspection (tool that is provided in android studio), we are able to debug the information in our room database. As we can observe, there is a random cipher that is generate from our AES encryption. Therefore, we are successful in storing encrypted password in our Room Database model.

A screenshot of the Android Studio interface showing the Room Database model. The "Live updates" checkbox is checked. The database contains a table with the following data:

	id	title	username	password	websi...	pin_n...
1	3	Facebook	txen2000@gmæ	d00UdIoZRtjTtegg9o7OtHD/AL3jWRLZGXNMNHk7Z/a	fb.com	
2	4	Snapchat	txen2000@gmæ	3zrkrVOKF5TkA6mGY490wAJZUX5Vqj7G+/FAAHPIYiyE2	snapchat.cc	1234
3	5	Amazon	txen2000@gmæ	/F3zLpysltrWS20DDeLclzHJlmlUa7+LNdGTSqOXNJ3TpM	amazon.co	
4	6	twst	tedt	JEOs86A7OU14yPpBcs2u5Q==	test	

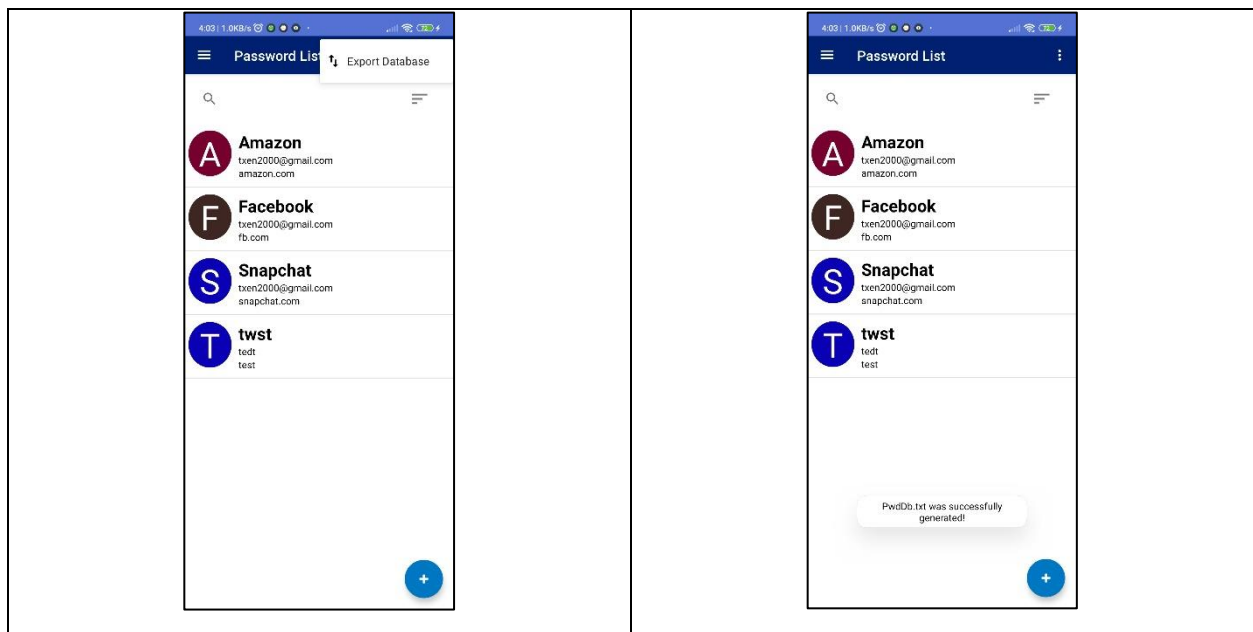
AES Key:

The randomly generated key is stored in our shared preference file. The key will be retrieved and used when the password manager is required to decrypt the password.

```
PwdListFragment.java × EditPwdActivity.java × MainActivity.java × mySharedPreferences.xml ×
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="decrypt_key">nyBZPhGIPi5n5mDh</string>
  <string name="master_password">root</string>
</map>
|
```

Exporting Database

By selecting our option menu, users are provided with the following option to export database. This option will create “PwdDb.txt”, that stores all the password in our database. This text file can be used as a backup, in the event the application is uninstalled for unspecified reason. User can then refer to the backup text file for their list of passwords stored.



The contents of “PwdDb.txt” is shown below:

```
PwdListFragment.java × EditPwdActivity.java × MainActivity.java × mySharedPreferences.xml × PwdDb.txt × activity_main_drawer.xml × PasswordElem.java ×
id=5, title='Amazon', username='txen2000@gmail.com', password='/F3zLpysItrWS20DDeLcIzHJImUa7+LNdGTSq0XNJ3TpMIqUvmMTcKJ71',
id=3, title='Facebook', username='txen2000@gmail.com', password='d00UdloZrtjTtegg9o70tHD/AL3jWRLZGXNMNHk37Z/aVrfro94cR0SiS6mqL9gq',
id=4, title='Snapchat', username='txen2000@gmail.com', password='3zrkvrVOKF5TkA6mGY490wAJZUX5Vqj7G+/FAAHPiYiyE2BfYpt/piqR9asQG3eYg',
id=6, title='twst', username='tedt', password='JE0s86A70U14yPpBcs2u5Q==', website='test', created_date=java.util.GregorianCalendar[
|
```

Conclusion

In conclusion, our password manager application comprises of all the basic functionalities needed. This password manager app is able to accommodate for variety of websites and generate strong password up to the length of 50. It provides 4 characters sets, which is sufficient security to not be cracked easily. Furthermore, each password are encrypted using AES database and a randomly generated key unique for each android phone. Therefore, our password manager app meets the basic requirements for a functioning password manager.

Appendix:

PwdListAdapter.java

```
public class PwdListAdapter extends RecyclerView.Adapter<PwdListAdapter.PwdViewHolder> implements Filterable {

    public static final String TAG = PwdListAdapter.class.getSimpleName();

    private List<PasswordElem> mPwdList;
    private List<PasswordElem> mPwdListAll;
    private LayoutInflater mInflater;
    private String[] colors;

    public PwdListAdapter(Context mContext) {
        this.mPwdList = new ArrayList<PasswordElem>();
        this.mPwdListAll = new ArrayList<PasswordElem>();
        this.mInflater = LayoutInflater.from(mContext);
        this.colors = mContext.getResources().getStringArray(R.array.icon_colors);
    }

    @NonNull
    @Override
    public PwdListAdapter.PwdViewHolder onCreateViewHolder(ViewGroup parent, int position) {
        View mView = mInflater.inflate(R.layout.pwd_list_item, parent, false);
        return new PwdViewHolder(mView, this);
    }

    @Override
    public void onBindViewHolder(PwdListAdapter.PwdViewHolder viewHolder, int position) {
        PasswordElem curPwd = mPwdList.get(position);

        // Update Icon TextView
        int f_val = (int) curPwd.title.toUpperCase().charAt(0);
        int ind = (int) ((f_val - 65) / 26.0 * colors.length);

        viewHolder.icon_item_txt.setBackground().setColorFilter(Color.parseColor(colors[ind]), PorterDuff.Mode.SRC_ATOP);
        viewHolder.icon_item_txt.setText(curPwd.title.substring(0, 1).toUpperCase());

        // Bind Password Element to RecyclerView List Item
        viewHolder.title_item_txt.setText(curPwd.title);
        viewHolder.name_item_txt.setText(curPwd.username);
        viewHolder.website_item_txt.setText(curPwd.website);
    }

    @Override
    public int getItemCount() {
```

```

        return mPwdList.size();
    }

    public void updateList(List<PasswordElem> pwdList) {
        PwdListDiffCallback diffCallback = new PwdListDiffCallback(this.mPwdList, pwdList);
        DiffUtil.DiffResult diffResult = DiffUtil.calculateDiff(diffCallback);

        this.mPwdList.clear();
        this.mPwdList.addAll(pwdList);

        this.mPwdListAll.clear();
        this.mPwdListAll.addAll(pwdList);

        diffResult.dispatchUpdatesTo(this);
    }

    @Override
    public Filter getFilter() {
        return Searched_Filter;
    }

    private Filter Searched_Filter = new Filter() {
        @Override
        protected FilterResults performFiltering(CharSequence charSequence) {
            String charString = charSequence.toString();
            List<PasswordElem> filteredList = new ArrayList<>();

            if (charString.isEmpty()) {
                filteredList.addAll(mPwdListAll);
            } else {
                for (PasswordElem pwd : mPwdListAll) {
                    if (pwd.getTitle().toLowerCase().contains(charString.toLowerCase())) {
                        filteredList.add(pwd);
                    }
                }
            }
        }

        FilterResults filterResults = new FilterResults();
        filterResults.values = filteredList;
        return filterResults;
    }

    @Override
    protected void publishResults(CharSequence constraint, FilterResults results) {
        mPwdList.clear();
        mPwdList.addAll((ArrayList) results.values);
        notifyDataSetChanged();
    }

```

```

    }
};

class PwdViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener, View.OnCreateContextMenuListener{
    TextView icon_item_txt;
    TextView title_item_txt;
    TextView name_item_txt;
    TextView website_item_txt;

    PwdListAdapter mAdapter;

    public PwdViewHolder(View itemView, PwdListAdapter mAdapter) {
        super(itemView);
        icon_item_txt = itemView.findViewById(R.id.icon_item_txt);
        title_item_txt = itemView.findViewById(R.id.title_item_txt);
        name_item_txt = itemView.findViewById(R.id.name_item_txt);
        website_item_txt = itemView.findViewById(R.id.website_item_txt);

        this.mAdapter = mAdapter;

        itemView.setOnClickListener(this);
        itemView.setOnCreateContextMenuListener(this);
    }

    @Override
    public void onClick(View view) {
        // Get the position of the item that was clicked.
        int mPosition = getLayoutPosition();

        PasswordElem pwd = mPwdList.get(mPosition);

        // Spawn Edit page Activity
        Intent intent = new Intent(view.getContext(), EditPwdActivity.class);
        intent.putExtra("id", pwd.getId());
        view.getContext().startActivity(intent);
    }

    public void onCreateContextMenu(ContextMenu contextMenu, View view, ContextMenu.ContextMenuInfo contextMenuInfo) {
        int pos = getAdapterPosition();
        PasswordElem pwd = mPwdList.get(pos);
        MenuItem delete = contextMenu.add(Menu.NONE, 1, (int) pwd.getId(), "Delete");
    }
}
}

```

PwdListDiffCallback.java

```
public class PwdListDiffCallback extends DiffUtil.Callback{
    private List<PasswordElem> oldList;
    private List<PasswordElem> newList;

    public PwdListDiffCallback(List<PasswordElem> oldList, List<PasswordElem>
newList) {
        this.oldList = oldList;
        this.newList = newList;
    }

    @Override
    public int getOldListSize() {
        return this.oldList.size();
    }

    @Override
    public int getNewListSize() {
        return this.newList.size();
    }

    @Override
    public boolean areItemsTheSame(int oldItemPosition, int newItemPosition) {
        PasswordElem oldPwd = oldList.get(oldItemPosition);
        PasswordElem newPwd = newList.get(newItemPosition);
        return oldPwd.id == newPwd.id;
    }

    @Override
    public boolean areContentsTheSame(int oldItemPosition, int newItemPosition) {
        PasswordElem oldPwd = oldList.get(oldItemPosition);
        PasswordElem newPwd = newList.get(newItemPosition);
        return oldPwd.equals(newPwd);
    }
}
```

PwdListViewModel.java

```
public class PwdListViewModel extends AndroidViewModel {

    private PasswordElemRepository mRepository;
    private LiveData<List<PasswordElem>> pwdList;

    public PwdListViewModel(@NonNull Application app) {
        super(app);
        mRepository = new PasswordElemRepository(app);
    }

    public LiveData<List<PasswordElem>> getPwdList(){
        if(pwdList == null){pwdList = mRepository.getPasswordElemList();}
        return pwdList;
    }

    public LiveData<List<PasswordElem>> getPwdListDesc(){
        return mRepository.getPasswordElemListDesc();
    }

    public LiveData<List<PasswordElem>> getPwdListDate(){
        return mRepository.getPasswordElemListDate();
    }

    public LiveData<List<PasswordElem>> getPwdListDateDesc(){
        return mRepository.getPasswordElemListDateDesc();
    }

    public PasswordElem getPassword(long id) throws ExecutionException,
    InterruptedException {return mRepository.getPassword(id);}

    public void insertPassword(PasswordElem pwd) {mRepository.insertPassword(pwd);}
    public void updatePassword(PasswordElem pwd){mRepository.updatePassword(pwd);}
    public void deletePassword(PasswordElem pwd){mRepository.deletePassword(pwd);}
    public void deleteAllPassword() { mRepository.deleteAllPassword(); }
}
```

AppDatabase.java

```
@Database(entities = {PasswordElem.class}, version = 2, exportSchema = false)
@TypeConverters({Converters.class})
public abstract class AppDatabase extends RoomDatabase {
    public abstract PasswordElemDao pwdElemDao();

    private static AppDatabase instance = null;

    private static AppDatabase buildDatabase(Context context) {
        return Room.databaseBuilder(context.getApplicationContext(),
            AppDatabase.class, "password_db")
            // Wipes and rebuilds instead of migrating
            // if no Migration object.
            // Migration is not part of this practical.
            .fallbackToDestructiveMigration()
            .build();
    }

    public static AppDatabase getDatabase(Context context){
        if(instance == null){
            synchronized (AppDatabase.class){
                if(instance == null)
                    instance = buildDatabase(context);
            }
        }
        return instance;
    }
}
```

Converters.java

```
public class Converters {
    @TypeConverter
    public long calendarToTimestamp(Calendar calendar) {
        return calendar.getTimeInMillis();
    }
    @TypeConverter
    public Calendar timestampToCalendar(long value) {
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(value);
        return calendar;
    }
}
```


PasswordElem.java

```
@Entity(tableName = "password_tbl")
public class PasswordElem {
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id")
    public long id;

    @ColumnInfo(name = "title")
    public String title;

    @ColumnInfo(name = "username")
    public String username;

    @ColumnInfo(name = "password")
    public String password;

    @ColumnInfo(name = "website")
    public String website;

    @ColumnInfo(name = "pin_number")
    public String pin_number;

    @ColumnInfo(name = "security_question_1")
    public String security_question_1;

    @ColumnInfo(name = "security_answer_1")
    public String security_answer_1;

    @ColumnInfo(name = "security_question_2")
    public String security_question_2;

    @ColumnInfo(name = "security_answer_2")
    public String security_answer_2;

    @ColumnInfo(name = "security_question_3")
    public String security_question_3;

    @ColumnInfo(name = "security_answer_3")
    public String security_answer_3;

    @ColumnInfo(name = "created_date")
    public Calendar created_date;

    @ColumnInfo(name = "last_updated_date")
    public Calendar last_updated_date;
```

```
public PasswordElem(){
    this.title = "";
    this.username = "";
    this.password = "";
    this.website = "";
    this.pin_number = "";

    this.security_question_1 = "";
    this.security_question_2 = "";
    this.security_answer_3 = "";

    this.security_answer_1 = "";
    this.security_answer_2 = "";
    this.security_answer_3 = "";

    this.created_date = Calendar.getInstance();
    this.last_updated_date = Calendar.getInstance();
}

public long getId() {
    return id;
}

public void setId(long id) {
    this.id = id;
}

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getPassword() {
```

```
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getWebsite() {
        return website;
    }

    public void setWebsite(String website) {
        this.website = website;
    }

    public Calendar getCreated_date() {
        return created_date;
    }

    public void setCreated_date(Calendar created_date) {
        this.created_date = created_date;
    }

    public Calendar getLast_updated_date() {
        return last_updated_date;
    }

    public void setLast_updated_date(Calendar last_updated_date) {
        this.last_updated_date = last_updated_date;
    }

    public String getPin_number() {
        return pin_number;
    }

    public void setPin_number(String pin_number) {
        this.pin_number = pin_number;
    }

    public String getSecurity_question_1() {
        return security_question_1;
    }

    public void setSecurity_question_1(String security_question_1) {
        this.security_question_1 = security_question_1;
    }
}
```

```
}

public String getSecurity_answer_1() {
    return security_answer_1;
}

public void setSecurity_answer_1(String security_answer_1) {
    this.security_answer_1 = security_answer_1;
}

public String getSecurity_question_2() {
    return security_question_2;
}

public void setSecurity_question_2(String security_question_2) {
    this.security_question_2 = security_question_2;
}

public String getSecurity_answer_2() {
    return security_answer_2;
}

public void setSecurity_answer_2(String security_answer_2) {
    this.security_answer_2 = security_answer_2;
}

public String getSecurity_question_3() {
    return security_question_3;
}

public void setSecurity_question_3(String security_question_3) {
    this.security_question_3 = security_question_3;
}

public String getSecurity_answer_3() {
    return security_answer_3;
}

public void setSecurity_answer_3(String security_answer_3) {
    this.security_answer_3 = security_answer_3;
}

public void updateValue(String[] arr){
    this.title = arr[0];
    this.username = arr[1];
}
```

```

        this.password = arr[2];
        this.website = arr[3];
        this.last_updated_date = Calendar.getInstance();
    }

    public void updateOptionalValue(String[] arr){
        this.pin_number = arr[0];
        this.security_question_1 = arr[1];
        this.security_answer_1 = arr[2];
        this.security_question_2 = arr[3];
        this.security_answer_2 = arr[4];
        this.security_question_3 = arr[5];
        this.security_answer_3 = arr[6];
        this.last_updated_date = Calendar.getInstance();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        PasswordElem that = (PasswordElem) o;
        return id == that.id
            && title.equals(that.title)
            && username.equals(that.username)
            && password.equals(that.password)
            && website.equals(that.website);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, title, username, password, website, created_date,
last_updated_date);
    }

    @Override
    public String toString() {
        return "id=" + id +
            ", title='" + title + '\'' +
            ", username='" + username + '\'' +
            ", password='" + password + '\'' +
            ", website='" + website + '\'' +
            ", created_date=" + created_date +
            ", last_updated_date=" + last_updated_date;
    }
}

```

PasswordElemDao.java

```
@Dao
public interface PasswordElemDao {
    @Query("SELECT * FROM password_tbl ORDER BY title")
    LiveData<List<PasswordElem>> getPasswordElemList();

    @Query("SELECT * FROM password_tbl ORDER BY title DESC")
    LiveData<List<PasswordElem>> getPasswordElemListDesc();

    @Query("SELECT * FROM password_tbl ORDER BY created_date DESC")
    LiveData<List<PasswordElem>> getPasswordElemListDate();

    @Query("SELECT * FROM password_tbl ORDER BY created_date")
    LiveData<List<PasswordElem>> getPasswordElemListDateDesc();

    @Query("SELECT * FROM password_tbl WHERE id = :pwd_id")
    PasswordElem getPasswordElem(long pwd_id);

    @Insert
    long insertPassword(PasswordElem pwd);

    @Update
    void updatePassword (PasswordElem pwd);

    @Delete
    void deletePassword (PasswordElem pwd);

    @Query("DELETE FROM password_tbl")
    void deleteAllPassword();
}
```

PasswordElemRepository.java

```
public class PasswordElemRepository {
    private PasswordElemDao mPasswordElemDao;
    private LiveData<List<PasswordElem>> passwordElemList;
    public PasswordElemRepository(Application app){
        AppDatabase db = AppDatabase.getDatabase(app);
        mPasswordElemDao = db.pwdElemDao();
        passwordElemList = mPasswordElemDao.getPasswordElemList();
    }
    public LiveData<List<PasswordElem>> getPasswordElemList() {
        return passwordElemList;
    }
    public LiveData<List<PasswordElem>> getPasswordElemListDesc() {
        return mPasswordElemDao.getPasswordElemListDesc();
    }
    public LiveData<List<PasswordElem>> getPasswordElemListDate() {
        return mPasswordElemDao.getPasswordElemListDate();
    }
    public LiveData<List<PasswordElem>> getPasswordElemListDateDesc() {
        return mPasswordElemDao.getPasswordElemListDateDesc();
    }
    public PasswordElem getPassword(long pwdId) throws ExecutionException, InterruptedException {
        return new getPasswordAsync(mPasswordElemDao).execute(pwdId).get();
    }
    public void insertPassword (PasswordElem pwd) {
        new insertPasswordAsync(mPasswordElemDao).execute(pwd);
    }
    public void updatePassword(PasswordElem pwd){
        new updatePasswordAsync(mPasswordElemDao).execute(pwd);
    }
    public void deletePassword(PasswordElem pwd){
        new deletePasswordAsync(mPasswordElemDao).execute(pwd);
    }
    public void deleteAllPassword(){
        new deleteAllPasswordAsync(mPasswordElemDao).execute();
    }
    private static class getPasswordAsync extends AsyncTask<Long, Void, PasswordElem> {
        private PasswordElemDao mPasswordElemDaoAsync;
        public getPasswordAsync(PasswordElemDao mPasswordElemDaoAsync) {
            this.mPasswordElemDaoAsync = mPasswordElemDaoAsync;
        }
        @Override
        protected PasswordElem doInBackground(Long... ids) {
            return mPasswordElemDaoAsync.getPasswordElem(ids[0]);
        }
    }
}
```

```

}
private static class insertPasswordAsync extends AsyncTask<PasswordElem, Void, Long>{
    private PasswordElemDao mPasswordElemDaoAsync;
    public insertPasswordAsync(PasswordElemDao mPasswordElemDaoAsync) {
        this.mPasswordElemDaoAsync = mPasswordElemDaoAsync;
    }
    @Override
    protected Long doInBackground(PasswordElem... passwordElems) {
        long id = mPasswordElemDaoAsync.insertPassword(passwordElems[0]);
        return id;
    }
}
private static class updatePasswordAsync extends AsyncTask<PasswordElem, Void, Void>{
    private PasswordElemDao mPasswordElemDaoAsync;
    public updatePasswordAsync(PasswordElemDao mPasswordElemDaoAsync) {
        this.mPasswordElemDaoAsync = mPasswordElemDaoAsync;
    }
    @Override
    protected Void doInBackground(PasswordElem... passwordElems) {
        mPasswordElemDaoAsync.updatePassword(passwordElems[0]);
        return null;
    }
}
private static class deletePasswordAsync extends AsyncTask<PasswordElem, Void, Void>{
    private PasswordElemDao mPasswordElemDaoAsync;
    public deletePasswordAsync(PasswordElemDao mPasswordElemDaoAsync) {
        this.mPasswordElemDaoAsync = mPasswordElemDaoAsync;
    }
    @Override
    protected Void doInBackground(PasswordElem... passwordElems) {
        mPasswordElemDaoAsync.deletePassword(passwordElems[0]);
        return null;
    }
}
private static class deleteAllPasswordAsync extends AsyncTask<Void, Void, Void>{
    private PasswordElemDao mPasswordElemDaoAsync;

    public deleteAllPasswordAsync(PasswordElemDao mPasswordElemDaoAsync) {
        this.mPasswordElemDaoAsync = mPasswordElemDaoAsync;
    }
    @Override
    protected Void doInBackground(Void... voids) {
        mPasswordElemDaoAsync.deleteAllPassword();
        return null;
    }
}
}

```


AddPwdActivity.java

```
public class AddPwdActivity extends AppCompatActivity {
    public static final String TAG = AddPwdActivity.class.getSimpleName();
    private ActivityAddPwdBinding binding;
    private PwdListViewModel viewModel;
    private EditText title_txt;
    private EditText name_txt;
    private EditText pwd_txt;
    private EditText website_txt;
    // Optional
    private EditText pinNo_txt;
    private EditText secQes1_txt;
    private EditText secQes2_txt;
    private EditText secQes3_txt;
    private EditText secAns1_txt;
    private EditText secAns2_txt;
    private EditText secAns3_txt;
    private ImageView show_pass_btn;
    private SharedPreferences pref;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityAddPwdBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        // Binding Java Element to XML
        title_txt = binding.titleTxt;
        name_txt = binding.nameTxt;
        pwd_txt = binding.pwdTxt;
        website_txt = binding.websiteTxt;

        // Optional Elements
        pinNo_txt = binding.pinNoTxt;

        secQes1_txt = binding.secQes1Txt;
        secQes2_txt = binding.secQes2Txt;
        secQes3_txt = binding.secQes3Txt;

        secAns1_txt = binding.secAns1Txt;
        secAns2_txt = binding.secAns2Txt;
        secAns3_txt = binding.secAns3Txt;

        show_pass_btn = binding.showPassBtn;
```

```

        pref = this.getSharedPreferences("mySharedPreferences", MODE_PRIVATE);

        // Initialize Objects
        viewModel = ViewModelProviders.of(AddPwdActivity.this).get(PwdListViewModel.class);

        // Add Back Button at ActionBar
        if (getSupportActionBar() != null) {
            getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        }

        show_pass_btn.setOnClickListener(v -> ShowHidePass(v));
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_add, menu);
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id == R.id.option_save) {

            String pwd_str = pwd_txt.getText().toString();
            String key = pref.getString("decrypt_key", "");

            // Encrypt Password using AES Encryption
            try {
                pwd_str = util.encrypt(pwd_str, key);
            } catch (Exception e) {
                e.printStackTrace();
            }

            String[] txt_arr = {
                title_txt.getText().toString(),
                name_txt.getText().toString(),
                pwd_str,
                website_txt.getText().toString()
            };

            // Validation => Check to ensure Edit Text Entry are not empty
            // If There is Empty Text Field

```

```

        if(util.checkIfAnyEmpty(txt_arr)) {
            Toast.makeText(this, "Please ensure that all textFields are not empty!",
Toast.LENGTH_SHORT).show();
            return super.onOptionsItemSelected(item);
        }

        String[] txt_optional_arr = {
            pinNo_txt.getText().toString().isEmpty() ? "" : pinNo_txt.getText().toString(),
            secQes1_txt.getText().toString().isEmpty() ? "" : secQes1_txt.getText().toString(),
            secAns1_txt.getText().toString().isEmpty() ? "" : secAns1_txt.getText().toString(),
            secQes2_txt.getText().toString().isEmpty() ? "" : secQes2_txt.getText().toString(),
            secAns2_txt.getText().toString().isEmpty() ? "" : secAns2_txt.getText().toString(),
            secQes3_txt.getText().toString().isEmpty() ? "" : secQes3_txt.getText().toString(),
            secAns3_txt.getText().toString().isEmpty() ? "" : secAns3_txt.getText().toString()
        };

        PasswordElem pwd = new PasswordElem();
        pwd.updateValue(txt_arr);
        pwd.updateOptionalValue(txt_optional_arr);
        viewModel.insertPassword(pwd);
        Toast.makeText(this, "Successfully Save Record!", Toast.LENGTH_SHORT).show();
        finish();
    }
    return super.onOptionsItemSelected(item);
}

public void ShowHidePass(View view){
    if(pwd_txt.getTransformationMethod().equals(PasswordTransformationMethod.getInstance())){
        show_pass_btn.setImageResource(R.drawable.ic_baseline_visibility_24);
        //Show Password
        pwd_txt.setTransformationMethod(HideReturnsTransformationMethod.getInstance());
    }
    else{
        show_pass_btn.setImageResource(R.drawable.ic_baseline_visibility_off_24);
        //Hide Password
        pwd_txt.setTransformationMethod(PasswordTransformationMethod.getInstance());
    }
}

@Override
public boolean onSupportNavigateUp() {
    finish();
    return true;
}
}

```

ChangeMasPwdFragment.java

```
public class ChangeMasPwdFragment extends Fragment {
    private FragmentChangeMasPwdBinding binding;
    private EditText pwd_txt;
    private EditText con_pwd_txt;
    private ImageView show_pass_btn;
    private ImageView show_con_pass_btn;
    private Button submit_btn;
    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable
Bundle savedInstanceState) {
        binding = FragmentChangeMasPwdBinding.inflate(inflater, container, false);
        View root = binding.getRoot();

        // Bind Java Elements to XML
        pwd_txt = binding.pwdTxt;
        con_pwd_txt = binding.conPwdTxt;

        show_pass_btn = binding.showPassBtn;
        show_con_pass_btn = binding.showConPassBtn;

        submit_btn = binding.submitBtn;

        show_pass_btn.setOnClickListener(v -> ShowHidePass(v));
        show_con_pass_btn.setOnClickListener(v -> ShowConHidePass(v));

        submit_btn.setOnClickListener(v -> submitPassword(v));

        return root;
    }

    public void submitPassword(View v){
        String pwd = pwd_txt.getText().toString();
        String con_pwd = con_pwd_txt.getText().toString();

        if(pwd.equals(con_pwd)){
            pwd_txt.setText("");
            con_pwd_txt.setText("");

            SharedPreferences pref = v.getContext().getSharedPreferences("mySharedPreferences",
MODE_PRIVATE);
            SharedPreferences.Editor prefEditor = pref.edit();

            prefEditor.putString("master_password", pwd);
```

```

        prefEditor.commit();

        Toast.makeText(v.getContext(), String.format("Master Password Successfully Updated!"),
Toast.LENGTH_SHORT).show();
    } else{
        pwd_txt.setText("");
        con_pwd_txt.setText("");
        Toast.makeText(v.getContext(), String.format("Both passwords are not the same!\nPlease try
again."), Toast.LENGTH_SHORT).show();
    }
}

public void ShowHidePass(View view){
    if(pwd_txt.getTransformationMethod().equals(PasswordTransformationMethod.getInstance())){
        show_pass_btn.setImageResource(R.drawable.ic_baseline_visibility_24);
        //Show Password
        pwd_txt.setTransformationMethod(HideReturnsTransformationMethod.getInstance());
    }
    else{
        show_pass_btn.setImageResource(R.drawable.ic_baseline_visibility_off_24);
        //Hide Password
        pwd_txt.setTransformationMethod(PasswordTransformationMethod.getInstance());
    }
}

public void ShowConHidePass(View view){
    if(con_pwd_txt.getTransformationMethod().equals(PasswordTransformationMethod.getInstance())){
        show_con_pass_btn.setImageResource(R.drawable.ic_baseline_visibility_24);
        //Show Password
        con_pwd_txt.setTransformationMethod(HideReturnsTransformationMethod.getInstance());
    }
    else{
        show_con_pass_btn.setImageResource(R.drawable.ic_baseline_visibility_off_24);
        //Hide Password
        con_pwd_txt.setTransformationMethod(PasswordTransformationMethod.getInstance());
    }
}

@Override
public void onDestroyView() {
    super.onDestroyView();
    binding = null;
}
}

```

EditPwdActivity.java

```
public class EditPwdActivity extends AppCompatActivity {

    public static final String TAG = EditPwdActivity.class.getSimpleName();

    private ActivityEditPwdBinding binding;
    private PwdListViewModel viewModel;
    private PasswordElem curPwd;
    private EditText title_txt;
    private EditText name_txt;
    private EditText pwd_txt;
    private EditText website_txt;
    private ImageView show_pass_btn;
    private ImageView title_copy_btn;
    private ImageView name_copy_btn;
    private ImageView pwd_copy_btn;
    private ImageView website_copy_btn;

    // Optional Elements
    private EditText pinNo_txt;
    private TextView secQes1_txtView;
    private TextView secQes2_txtView;
    private TextView secQes3_txtView;
    private EditText secAns1_txt;
    private EditText secAns2_txt;
    private EditText secAns3_txt;
    private ImageView pinNo_copy_btn;
    private ImageView secAns1_copy_btn;
    private ImageView secAns2_copy_btn;
    private ImageView secAns3_copy_btn;
    private LinearLayout pin_ll;
    private LinearLayout sec_1_ll;
    private LinearLayout sec_2_ll;
    private LinearLayout sec_3_ll;
    private SharedPreferences pref;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityEditPwdBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        // Add Back Button at ActionBar
        if (getSupportActionBar() != null) {
            getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        }

        // Initialize Objects
```

```
viewModel = ViewModelProviders.of(EditPwdActivity.this).get(PwdListViewModel.class);

// Binding Java Element to XML
title_txt = binding.titleTxt;
name_txt = binding.nameTxt;
pwd_txt = binding.pwdTxt;
website_txt = binding.websiteTxt;

show_pass_btn = binding.showPassBtn;

title_copy_btn = binding.titleCopyBtn;
name_copy_btn = binding.nameCopyBtn;
pwd_copy_btn = binding.pwdCopyBtn;
website_copy_btn = binding.websiteCopyBtn;

// Optional
pinNo_txt = binding.pinNoTxt;

secQes1_txtView = binding.secQes1TxtView;
secQes2_txtView = binding.secQes2TxtView;
secQes3_txtView = binding.secQes3TxtView;

secAns1_txt = binding.secAns1Txt;
secAns2_txt = binding.secAns2Txt;
secAns3_txt = binding.secAns3Txt;

pinNo_copy_btn = binding.pinNoCopyBtn;
secAns1_copy_btn = binding.secAns1CopyBtn;
secAns2_copy_btn = binding.secAns2CopyBtn;
secAns3_copy_btn = binding.secAns3CopyBtn;
pin_ll = binding.pinLl;
sec_1_ll = binding.sec1Ll;
sec_2_ll = binding.sec2Ll;
sec_3_ll = binding.sec3Ll;
pref = this.getSharedPreferences("mySharedPreferences", MODE_PRIVATE);
Intent mIntent = getIntent();
Long ind = (long) mIntent.getLongExtra("id", 0);
try {
    curPwd = viewModel.getPassword(ind);
} catch (ExecutionException e) {
    e.printStackTrace();
} catch (InterruptedException e) {
    e.printStackTrace();
}
title_txt.setText(curPwd.getTitle());
```

```

name_txt.setText(curPwd.getUsername());
// Decrypt Password
String pwd_str = curPwd.getPassword();
String key = pref.getString("decrypt_key", "");
// Encrypt Password using AES Encryption
try {
    pwd_str = util.decrypt(pwd_str, key);
} catch (Exception e) {
    e.printStackTrace();
}
pwd_txt.setText(pwd_str);
website_txt.setText(curPwd.getWebsite());
pinNo_txt.setText(curPwd.getPin_number());
if(curPwd.getPin_number().isEmpty()){
    pin_1l.setVisibility(View.GONE);
}
secQes1_txtView.setText(curPwd.getSecurity_question_1());
secAns1_txt.setText(curPwd.getSecurity_answer_1());
if(curPwd.getSecurity_question_1().isEmpty()){
    sec_1_1l.setVisibility(View.GONE);
}
secQes2_txtView.setText(curPwd.getSecurity_question_2());
secAns2_txt.setText(curPwd.getSecurity_answer_2());
if(curPwd.getSecurity_question_2().isEmpty()){
    sec_2_1l.setVisibility(View.GONE);
}
secQes3_txtView.setText(curPwd.getSecurity_question_3());
secAns3_txt.setText(curPwd.getSecurity_answer_3());
if(curPwd.getSecurity_question_3().isEmpty()){
    sec_3_1l.setVisibility(View.GONE);
}
show_pass_btn.setOnClickListener(v -> ShowHidePass(v));
// Copy to Clipboard
title_copy_btn.setOnClickListener(v -> util.copyCodeInClipBoard(
    v.getContext(), "Title has been copied to clipboard.", title_txt.getText().toString()));
name_copy_btn.setOnClickListener(v -> util.copyCodeInClipBoard(
    v.getContext(), "Username has been copied to clipboard.", name_txt.getText().toString()));
pwd_copy_btn.setOnClickListener(v -> util.copyCodeInClipBoard(
    v.getContext(), "Password has been copied to clipboard.", pwd_txt.getText().toString()));
website_copy_btn.setOnClickListener(v -> util.copyCodeInClipBoard(
    v.getContext(), "URL has been copied to clipboard.", website_txt.getText().toString()));
pinNo_copy_btn.setOnClickListener(v -> util.copyCodeInClipBoard(
    v.getContext(), "PIN Number has been copied to clipboard.",
pinNo_txt.getText().toString()));
secAns1_copy_btn.setOnClickListener(v -> util.copyCodeInClipBoard(

```



```

        v.getContext(), "Security Answer has been copied to clipboard.",
secAns1_txt.getText().toString());
        secAns2_copy_btn.setOnClickListener(v -> util.copyCodeInClipBoard(
            v.getContext(), "Security Answer has been copied to clipboard.",
secAns2_txt.getText().toString());
        secAns3_copy_btn.setOnClickListener(v -> util.copyCodeInClipBoard(
            v.getContext(), "Security Answer has been copied to clipboard.",
secAns3_txt.getText().toString());
    }
    @Override
    public boolean onSupportNavigateUp() {
        finish();
        return true;
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_edit, menu);
        return true;
    }
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id == R.id.option_save) {
            String pwd_str = pwd_txt.getText().toString();
            String key = pref.getString("decrypt_key", "");
            // Encrypt Password using AES Encryption
            try {
                pwd_str = util.encrypt(pwd_str, key);
            } catch (Exception e) {
                e.printStackTrace();
            }
            String[] txt_arr = {
                title_txt.getText().toString(),
                name_txt.getText().toString(),
                pwd_str,
                website_txt.getText().toString()
            };
            // Validation => Check to ensure Edit Text Entry are not empty
            // If There is Empty Text Field
            if(util.checkIfAnyEmpty(txt_arr)) {
                Toast.makeText(this, "Please ensure that all textFields are not empty!",
Toast.LENGTH_SHORT).show();
                return super.onOptionsItemSelected(item);
            }
            String[] txt_optional_arr = {

```

```

        pinNo_txt.getText().toString().isEmpty() ? "" : pinNo_txt.getText().toString(),
        secQes1_txtView.getText().toString().isEmpty() ? "" :
secQes1_txtView.getText().toString(),
        secAns1_txt.getText().toString().isEmpty() ? "" : secAns1_txt.getText().toString(),
        secQes2_txtView.getText().toString().isEmpty() ? "" :
secQes2_txtView.getText().toString(),
        secAns2_txt.getText().toString().isEmpty() ? "" : secAns2_txt.getText().toString(),
        secQes3_txtView.getText().toString().isEmpty() ? "" :
secQes3_txtView.getText().toString(),
        secAns3_txt.getText().toString().isEmpty() ? "" : secAns3_txt.getText().toString()
    );
    curPwd.updateValue(txt_arr);
    curPwd.updateOptionalValue(txt_optional_arr);

    viewModel.updatePassword(curPwd);
    Toast.makeText(this, "Successfully Edited Record!", Toast.LENGTH_SHORT).show();
    finish();
} else if(id == R.id.option_delete){
    // Alert Box
    new AlertDialog.Builder(this)
        .setMessage("Are you sure you want to delete?")
        .setCancelable(false)
        .setPositiveButton("Yes", (dialog, ind) -> {
            viewModel.deletePassword(curPwd);
            Toast.makeText(this, "Successfully Deleted Record!", Toast.LENGTH_SHORT).show();
            finish();
        })
        .setNegativeButton("No", (dialog, ind) -> dialog.cancel())
        .show();
}
return super.onOptionsItemSelected(item);
}

public void ShowHidePass(View view){
    if(pwd_txt.getTransformationMethod().equals(PasswordTransformationMethod.getInstance())){
        show_pass_btn.setImageResource(R.drawable.ic_baseline_visibility_24);
        //Show Password
        pwd_txt.setTransformationMethod(HideReturnsTransformationMethod.getInstance());
    }
    else{
        show_pass_btn.setImageResource(R.drawable.ic_baseline_visibility_off_24);
        //Hide Password
        pwd_txt.setTransformationMethod(PasswordTransformationMethod.getInstance());
    }
}
}
}

```

GenPwdFragment.java

```
public class GenPwdFragment extends Fragment {
    private FragmentGenPwdBinding binding;
    private TextView pwd_txtView;
    private TextView pwd_len_txtView;
    private ImageView reset_pwd_btn;
    private ImageView copy_pwd_btn;
    private Slider pwd_slider;
    private SwitchMaterial uppercase_switch;
    private SwitchMaterial lowercase_switch;
    private SwitchMaterial digit_switch;
    private SwitchMaterial symbol_switch;
    private Context context;

    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        binding = FragmentGenPwdBinding.inflate(inflater, container, false);
        View root = binding.getRoot();
        context = getContext();
        // Bind Java Object to XML Element
        pwd_txtView = binding.pwdTxtView;
        pwd_len_txtView = binding.pwdLenTxtView;
        reset_pwd_btn = binding.resetPwdBtn;
        copy_pwd_btn = binding.copyPwdBtn;
        pwd_slider = binding.pwdSlider;
        uppercase_switch = binding.uppercaseSwitch;
        lowercase_switch = binding.lowercaseSwitch;
        digit_switch = binding.digitSwitch;
        symbol_switch = binding.symbolSwitch;
        // Set Default Password
        String password = util.genPassword(4, true, true, true, true);
        pwd_txtView.setText(password);
        // Update & Set Password Length
        pwd_slider.addOnChangeListener((slider, value, fromUser) -> {
            int pwd_len = (int) value;
            updatePwdAndLength(pwd_len);
        });
        uppercase_switch.setOnClickListener(v -> updatePwd());
        lowercase_switch.setOnClickListener(v -> updatePwd());
        digit_switch.setOnClickListener(v -> updatePwd());
        symbol_switch.setOnClickListener(v -> updatePwd());
        // Check if at least one option is selected
        reset_pwd_btn.setOnClickListener(v -> updatePwd());
        // Copy to Clipboard
        copy_pwd_btn.setOnClickListener(v -> util.copyCodeInClipBoard(v.getContext(), "Password has been copied to clipboard.", pwd_txtView.getText().toString()));
        return root;
    }
}
```

```

    }

    public boolean checkOptions(boolean needUpperCase, boolean needLowerCase, boolean needDigit, boolean needSymbol){
        return needUpperCase || needLowerCase || needDigit || needSymbol;
    }

    // Get Password Length
    public void updatePwdAndLength(int pwd_len){
        boolean needUpper = uppercase_switch.isChecked();
        boolean needLower = lowercase_switch.isChecked();
        boolean needDigit = digit_switch.isChecked();
        boolean needSymbol = symbol_switch.isChecked();
        // Check if there's at least one Option
        if(!checkOptions(needUpper, needLower, needDigit, needSymbol)){
            pwd_txtView.setText("Not enough usable characters defined");
            return;
        }

        // Update Password Length
        pwd_len_txtView.setText(String.format("Password length: %d", pwd_len));
        // Also generates New Password
        String password = util.genPassword(pwd_len, needUpper, needLower, needDigit, needSymbol);
        pwd_txtView.setText(password);
    }

    // Reset
    public void updatePwd(){
        boolean needUpper = uppercase_switch.isChecked();
        boolean needLower = lowercase_switch.isChecked();
        boolean needDigit = digit_switch.isChecked();
        boolean needSymbol = symbol_switch.isChecked();
        if(!checkOptions(needUpper, needLower, needDigit, needSymbol)){
            pwd_txtView.setText("Not enough usable characters defined");
            return;
        }

        int pwd_len = (int) pwd_slider.getValue();
        String password = util.genPassword(pwd_len, needUpper, needLower, needDigit, needSymbol);
        pwd_txtView.setText(password);
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        binding = null;
    }
}

```

LoginActivity.java

```
public class LoginActivity extends AppCompatActivity {

    public static final String TAG = LoginActivity.class.getSimpleName();

    private ActivityLoginBinding binding;

    private EditText pwd_txt;
    private Button submit_btn;
    private ImageView show_pass_btn;

    private SharedPreferences pref;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityLoginBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        // Bind Java Elements to XML Objects
        pwd_txt = binding.pwdTxt;
        submit_btn = binding.submitBtn;
        show_pass_btn = binding.showPassBtn;

        // Check if Saved Preference Exists
        pref = getSharedPreferences("mySharedPreferences", MODE_PRIVATE);

        // If Master Password Does Not Exist
        if(pref.getString("master_password", null) == null){
            SharedPreferences.Editor prefEditor = pref.edit();
            prefEditor.putString("master_password", "root");
            prefEditor.commit();
        }

        // If Decryption Key Does not Exist
        if(pref.getString("decrypt_key", null) == null){
            SharedPreferences.Editor prefEditor = pref.edit();
            String key = util.genPassword(16, true, true, true, false);
            prefEditor.putString("decrypt_key", key);
            prefEditor.commit();
        }

        show_pass_btn.setOnClickListener(v -> ShowHidePass(v));
    }
}
```

```

        // When Submit Button is clicked, the password is parsed to check if its
similar or something
        submit_btn.setOnClickListener(v -> submitPassword(v));
    }

    public void submitPassword(View v){
        String pwd = pwd_txt.getText().toString();

        String master_pwd = pref.getString("master_password", null);

        if(pwd.equals(master_pwd)){
            pwd_txt.setText("");
            Intent intent = new Intent(this, MainActivity.class);
            startActivity(intent);
        } else{
            pwd_txt.setText("");
            Toast.makeText(this, String.format("Password is incorrect!\nPlease
try again."), Toast.LENGTH_SHORT).show();
        }
    }

    public void ShowHidePass(View view){
        if(pwd_txt.getTransformationMethod().equals(PasswordTransformationMethod.
getInstance())){
            show_pass_btn.setImageResource(R.drawable.ic_baseline_visibility_24);
            //Show Password
            pwd_txt.setTransformationMethod(HideReturnsTransformationMethod.getInsta
nce());
        }
        else{
            show_pass_btn.setImageResource(R.drawable.ic_baseline_visibility_off_
24);
            //Hide Password
            pwd_txt.setTransformationMethod(PasswordTransformationMethod.getInsta
nce());
        }
    }
}

```

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    public static final String TAG = MainActivity.class.getSimpleName();

    private AppBarConfiguration mAppBarConfiguration;
    private ActivityMainBinding binding;
    private PwdListViewModel viewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        viewModel = ViewModelProviders.of(this).get(PwdListViewModel.class);

        setSupportActionBar(binding.appBarMain.toolbar);
        binding.appBarMain.toolbar.setOnMenuItemClickListener(item -> {
            int id = item.getItemId();
            if (id == R.id.option_export) {
                FileOutputStream fos;

                String fileName = "PwdDb.txt";

                try {
                    fos = openFileOutput(fileName, MODE_PRIVATE);
                    viewModel.getPwdList().observe(this, pwdList -> {
                        for (PasswordElem elem : pwdList) {
                            try {
                                fos.write(String.format("%s\n", elem.toString()).getBytes());
                            } catch (IOException e) {
                                e.printStackTrace();
                            }
                        }
                    });
                    fos.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }

                Toast.makeText(this, "PwdDb.txt was successfully generated!", Toast.LENGTH_SHORT).show();
            }

            return true;
        });

        DrawerLayout drawer = binding.drawerLayout;
        NavigationView navigationView = binding.navView;

        // Passing each menu ID as a set of Ids because each
```

```

// menu should be considered as top level destinations.
mAppBarConfiguration = new AppBarConfiguration.Builder(
    R.id.nav_pwd_list, R.id.nav_gen_pwd, R.id.nav_change_mas_pwd)
    .setOpenableLayout(drawer)
    .build();

NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment_content_main);
NavigationUI.setupActionBarWithNavController(this, navController, mAppBarConfiguration);
navigationView.setNavigationItemSelectedListener(item -> {
    if (item.getItemId() == R.id.nav_exit) {
        new AlertDialog.Builder(this)
            .setMessage("Are you sure you want to logout?")
            .setCancelable(false)
            .setPositiveButton("Yes", (dialog, id) -> finish())
            .setNegativeButton("No", (dialog, id) -> dialog.cancel())
            .show();
    } else {
        NavigationUI.onNavDestinationSelected(item, navController);
        drawer.closeDrawers();
    }
    return false;
});
}

@Override
public boolean onSupportNavigateUp() {
    NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment_content_main);
    return NavigationUI.navigateUp(navController, mAppBarConfiguration)
        || super.onSupportNavigateUp();
}

@Override
public void onBackPressed() {
    new AlertDialog.Builder(this)
        .setMessage("Are you sure you want to exit?")
        .setCancelable(false)
        .setPositiveButton("Yes", (dialog, id) -> finishAffinity())
        .setNegativeButton("No", (dialog, id) -> dialog.cancel())
        .show();
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
}

```


PwdListFragment.java

```
public class PwdListFragment extends Fragment {

    public static final String TAG = PwdListFragment.class.getSimpleName();

    private FragmentPwdListBinding binding;

    private RecyclerView mRecyclerView;

    private PwdListViewModel viewModel;
    private PwdListAdapter mAdapter;

    private SearchView searchBar;
    private ImageView sort_btn;

    private FloatingActionButton fab;

    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

        binding = FragmentPwdListBinding.inflate(inflater, container, false);
        View root = binding.getRoot();

        mRecyclerView = binding.pwdRecView;
        searchBar = binding.searchBar;
        sort_btn = binding.sortBtn;
        fab = binding.fab;

        fab.setOnClickListener(v -> nAddPwd(v));

        viewModel = ViewModelProviders.of(this.getActivity()).get(PwdListViewModel.class);

        mAdapter = new PwdListAdapter(this.getActivity());

        mRecyclerView.setAdapter(mAdapter);

        // Give the RecyclerView a default layout manager.
        mRecyclerView.setLayoutManager(new LinearLayoutManager(this.getActivity()));

        DividerItemDecoration dividerItemDecoration = new DividerItemDecoration(this.getActivity(),
DividerItemDecoration.VERTICAL);
        mRecyclerView.addItemDecoration(dividerItemDecoration);

        // Updates Password List
        viewModel.getPwdList().observe(this, pwdList -> {
            mAdapter.updateList(pwdList);
        });
    }
}
```

```

    });

    searchBar.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
        @Override
        public boolean onQueryTextSubmit(String query) {
            return false;
        }

        @Override
        public boolean onQueryTextChange(String newText) {
            mAdapter.getFilter().filter(newText);
            return false;
        }
    });

    sort_btn.setOnClickListener(view -> {
        PopupMenu popup = new PopupMenu(this.getActivity(), sort_btn);
        popup.getMenuInflater().inflate(R.menu.menu_sort, popup.getMenu());

        //registering popup with OnMenuItemClickListener
        popup.setOnMenuItemClickListener(item -> {
            int id = item.getItemId();
            if(id == R.id.option_a_to_z){
                viewModel.getPwdList().observe(this, pwdList -> mAdapter.updateList(pwdList));
            } else if(id == R.id.option_z_to_a){
                viewModel.getPwdListDesc().observe(this, pwdList -> mAdapter.updateList(pwdList));
            } else if(id == R.id.option_new_to_old){
                viewModel.getPwdListDate().observe(this, pwdList -> mAdapter.updateList(pwdList));
            } else if(id == R.id.option_old_to_new){
                viewModel.getPwdListDateDesc().observe(this, pwdList -> mAdapter.updateList(pwdList));
            }
            return true;
        });

        popup.show();//showing popup menu
    });

    return root;
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    int pos = item.getOrder();

    // Alert Dialog

```

```

        new AlertDialog.Builder(this.getContext())
            .setMessage("Are you sure you want to delete?")
            .setCancelable(false)
            .setPositiveButton("Yes", (dialog, id) -> {
                PasswordElem pwd = null;
                try {
                    pwd = viewModel.getPassword((long) pos);
                } catch (ExecutionException e) {
                    e.printStackTrace();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

                viewModel.deletePassword(pwd);

                viewModel.getPwdList().observe(this, pwdList -> {
                    mAdapter.updateList(pwdList);
                });
            })
            .setNegativeButton("No", (dialog, id) -> dialog.cancel())
            .show();

        return super.onContextItemSelected(item);
    }

    @Override
    public void onResume() {
        super.onResume();
        viewModel.getPwdList().observe(this, pwdList -> {
            mAdapter.updateList(pwdList);
        });
    }

    public void nAddPwd(View v){
        Intent intent = new Intent(v.getContext(), AddPwdActivity.class);
        startActivity(intent);
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        binding = null;
    }
}

```

Util.java

```
public class util {
    private static final String ALGORITHM = "AES";

    public static boolean checkIfAnyEmpty(String[] arr){
        for(String s : arr){
            if(s.isEmpty()){
                return true;
            }
        }
        return false;
    }

    /**
     * Method to code text in clip board
     *
     * @param context context
     * @param text    text what wan to copy in clipboard
     * @param label   label what want to copied
     */
    public static void copyCodeInClipBoard(Context context, String label, String
text) {
        if (context != null) {
            ClipboardManager clipboard = (ClipboardManager)
context.getSystemService(Context.CLIPBOARD_SERVICE);
            ClipData clip = ClipData.newPlainText(label, text);
            if (clipboard == null || clip == null)
                return;
            clipboard.setPrimaryClip(clip);
            Toast.makeText(context, "Successfully Copied to Clipboard!",
Toast.LENGTH_SHORT).show();
        }
    }

    public static String genPassword(int pwd_len, boolean needUpperCase, boolean
needLowerCase, boolean needDigit, boolean needSymbol){
        String password = genPassword2(pwd_len, needUpperCase, needLowerCase,
needDigit, needSymbol);
        while(!check(password, needUpperCase, needLowerCase, needDigit,
needSymbol)){
            password = genPassword2(pwd_len, needUpperCase, needLowerCase,
needDigit, needSymbol);
        }
        return password;
    }
}
```

```

    public static String genPassword2(int pwd_len, boolean needUpperCase, boolean
needLowerCase, boolean needDigit, boolean needSymbol){
        String lowercase_str = "abcdefghijklmnopqrstuvwxyz";
        String uppercase_str = lowercase_str.toUpperCase();
        String digit_str = "012345689";
        String symbol_str = "#?!:;?%*£€$=+@{ }[]&()";

        String chars="";
        if(needUpperCase) chars += uppercase_str;
        if(needLowerCase) chars += lowercase_str;
        if(needDigit) chars += digit_str;
        if(needSymbol) chars += symbol_str;
        StringBuilder finalPassword = new StringBuilder();
        for(int i = 0; i < pwd_len; i++) {
            int index = ThreadLocalRandom.current().nextInt(0,chars.length());
            finalPassword.append(chars.charAt(index));
        }
        return finalPassword.toString();
    }

    public static boolean isAlphaNumeric(char char1) {
        return (char1 >= 'a' && char1 <= 'z') || (char1 >= 'A' && char1 <= 'Z')
|| (char1 >= '0' && char1 <= '9');
    }

    private static boolean check(String password, boolean needUpperCase, boolean
needLowerCase, boolean needDigit, boolean needSymbol) {
        //we declare our booleans
        boolean hasDigit = false;
        boolean hasSymbol = false;
        boolean hasLower = false;
        boolean hasUpper = false;

        for(char c : password.toCharArray()) {
            //we check that the password corresponds to a sufficient level of
security according to the selected options
            if (needUpperCase && !hasUpper) hasUpper = Character.isUpperCase(c);
            if (needLowerCase && !hasLower) hasLower = Character.isLowerCase(c);
            if (needDigit && !hasDigit) hasDigit = Character.isDigit(c);
            if (needSymbol && !hasSymbol) hasSymbol = !isAlphaNumeric(c);
        }
        return (!needUpperCase || hasUpper) && (!needLowerCase || hasLower) &&
(!needDigit || hasDigit) && (!needSymbol || hasSymbol);
    }

```

```

public static String encrypt(String value, String KEY) throws Exception
{
    Key key = generateKey(KEY);
    Cipher cipher = Cipher.getInstance(ALGORITHM);
    cipher.init(Cipher.ENCRYPT_MODE, key);
    byte [] encryptedByteValue = cipher.doFinal(value.getBytes("utf-8"));
    String encryptedValue64 = Base64.encodeToString(encryptedByteValue,
Base64.DEFAULT);
    return encryptedValue64;
}

public static String decrypt(String value, String KEY) throws Exception
{
    Key key = generateKey(KEY);
    Cipher cipher = Cipher.getInstance(ALGORITHM);
    cipher.init(Cipher.DECRYPT_MODE, key);
    byte[] decryptedValue64 = Base64.decode(value, Base64.DEFAULT);
    byte [] decryptedByteValue = cipher.doFinal(decryptedValue64);
    String decryptedValue = new String(decryptedByteValue,"utf-8");
    return decryptedValue;
}

private static Key generateKey(String KEY) throws Exception
{
    Key key = new SecretKeySpec(KEY.getBytes(), ALGORITHM);
    return key;
}
}

```