Question 1

a) Three reasons why companies have mobile applications instead of depending on web or desktop applications are:

1. Ubiquitous access: Mobile applications allow users to access services and information anytime and anywhere using their smartphones or tablets. This level of convenience and accessibility is not possible with web or desktop applications, which require a stable internet connection and a specific device.

2. Enhanced user experience: Mobile applications can be optimized specifically for mobile devices, providing a better user experience compared to web or desktop applications. They can leverage device-specific features such as touch gestures, camera, GPS, and push notifications, resulting in a more interactive and personalized experience.

3. Market reach and engagement: With the proliferation of smartphones, mobile applications offer a significant opportunity for companies to reach a large and engaged user base. Mobile apps can be distributed through app stores, where users actively search for and discover new applications, increasing the chances of user acquisition and engagement.

b) Mobile application development refers to the process of creating software applications that are designed to run on mobile devices such as smartphones and tablets. It involves designing, developing, testing, and deploying applications specifically tailored for the mobile platform.

Mobile application development typically includes activities such as:

- User interface (UI) design: Creating visually appealing and intuitive interfaces optimized for mobile devices, considering factors like screen size, touch interactions, and usability.
- Front-end development: Implementing the user interface using programming languages and frameworks like Java, Kotlin (for Android) or Swift, Objective-C (for iOS).
- Back-end development: Building server-side components, databases, and APIs that enable communication between the mobile app and server infrastructure.
- Testing and quality assurance: Conducting rigorous testing to ensure the app functions as intended, is compatible with different devices and operating systems, and delivers a satisfactory user experience.
- Deployment and maintenance: Releasing the app to app stores or enterprise distribution channels, and providing ongoing support, bug fixes, and updates.

c) Three common features of mobile applications are:

1. Push notifications: Mobile apps can send notifications directly to users' devices, even when the app is not actively running. Push notifications are used to deliver important information, updates, alerts, or personalized messages to keep users engaged and informed.

2. Offline access: Many mobile apps offer offline functionality, allowing users to access certain features or content even without an internet connection. This feature is particularly useful in scenarios where network connectivity is limited or unreliable.

3. Device integration: Mobile apps can leverage various built-in device features and capabilities to enhance user experience. These may include access to the camera for photo/video capture, GPS for location-based services, accelerometer for motion detection, and microphone for audio input, among others.

d) Three reasons for mobile application failures include:

1. Poor performance: Mobile apps that are slow, unresponsive, or crash frequently can lead to user frustration and dissatisfaction. Performance issues can result from inefficient coding practices, inadequate testing, insufficient device compatibility checks, or resource-intensive operations.

2. Lack of user engagement: If a mobile app fails to engage users or provide compelling value, it may struggle to retain an active user base. Factors such as a poorly designed user interface, lack of relevant content or features, or ineffective user onboarding can contribute to low engagement and eventual abandonment of the app.

3. Security vulnerabilities: Mobile applications can be susceptible to security breaches if proper security measures are not implemented. Inadequate data encryption, weak authentication mechanisms, and vulnerabilities in third-party libraries or APIs can expose user data to unauthorized access or compromise, leading to reputational damage and loss of user trust.

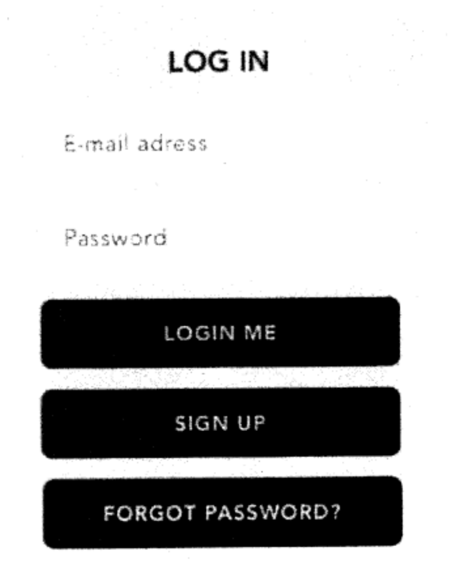
e) Three factors to consider before starting the development of a mobile application are:

1. Target audience and market analysis: It is crucial to identify the target audience for your app and understand their needs, preferences, and behaviors. Conducting

Question 2

a) UI is concerned with the visual and interactive design elements, while UX focuses on the overall experience and satisfaction of the user. UI is a subset of UX, as it contributes to the overall user experience by providing an aesthetically pleasing and intuitive interface.

b) Criticize the useability aspects of the given user interface (3 Marks) I do not know….



?

Question 03

a)

Entity Diagram for User: Entity Diagram for FruitJuice: Entity Diagram for Feedback:

User Properties - userId - username - password - email - phoneNumber - address	FruitJuice - juiceId - name - description - price	Feedback - feedbackId - userId - juiceId - rating - comment - date
Methods: - createUserAccount() - login() - purchaseFruitJuice() - rateAndAddFeedback() - viewPurchaseHistory() - addToFavorites()	Methods: - addFruitJuice() - editFruitJuice() - deleteFruitJuice()	Methods: - viewCustomerFeedback() - replyToCustomerFeedback()

b)

1. Login Activity

- TextView: "Login"
- EditText: Username
- EditText: Password
- Button: Login
- Button: Register

2. Menu Activity

- TextView: "Fresh Juices Menu"
- RecyclerView: List of fruit juices

3. Juice Details Activity

- TextView: Juice name
- TextView: Juice description
- TextView: Juice price
- RatingBar: Rating for the juice
- Button: Add to Cart

4. Cart Activity

- TextView: "Cart"
- RecyclerView: List of selected juices
- Button: Checkout

5. Checkout Activity

- TextView: "Checkout"
- EditText: Name
- EditText: Address
- EditText: Payment details
- Button: Confirm Payment

6. Purchase History Activity

- TextView: "Purchase History"
- RecyclerView: List of previous purchases

7. Favorites Activity

- TextView: "Favorites"
- RecyclerView: List of favorite fruit juices

8. Feedback Activity

- TextView: "Provide Feedback"
- EditText: Feedback comment
- RatingBar: Rating for the juice
- Button: Submit Feedback

c) Login Activity:

  a. Click on "Login" button: Navigate to Menu Activity.
  b. Click on "Register" button: Navigate to Register Activity.

Menu Activity:

  c. Click on a specific fruit juice from the RecyclerView: Navigate to Juice Details Activity.
  d. Click on the cart icon or "Checkout" button: Navigate to Cart Activity.
  e. Click on the "Purchase History" button: Navigate to Purchase History Activity.
  f. Click on the "Favorites" button: Navigate to Favorites Activity.
  g. Click on the "Provide Feedback" button: Navigate to Feedback Activity.

Juice Details Activity:

  h. Click on the "Add to Cart" button: Navigate back to Menu Activity.
  i. Cart Activity:
  j. Click on the "Checkout" button: Navigate to Checkout Activity.
  k. Click on a specific item in the RecyclerView: Navigate to Juice Details Activity.

Checkout Activity:

  l. Click on the "Confirm Payment" button: Navigate back to Menu Activity.
  m. Purchase History Activity:
  n. No specific navigation mentioned.

Favorites Activity:

  o. Click on a specific fruit juice from the RecyclerView: Navigate to Juice Details Activity.

Feedback Activity:

  p. Click on the "Submit Feedback" button: Navigate back to Menu Activity.

Question 04

a) 1.) Register Button Click Event:

```
Button registerButton = findViewById(R.id.registerButton);
registerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Perform necessary actions to navigate to RegisterActivity
        Intent intent = new Intent(MainActivity.this, RegisterActivity.class);
        startActivity(intent);
    }
});
```

2.) Login Button Click Event (Admin Login):

```
Button loginButton = findViewById(R.id.loginButton);
loginButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Get the entered username and password
        EditText usernameEditText = findViewById(R.id.usernameEditText);
        EditText passwordEditText = findViewById(R.id.passwordEditText);
        String username = usernameEditText.getText().toString();
        String password = passwordEditText.getText().toString();

        // Check if the entered username and password match the admin credentials
        if (username.equals("Admin") && password.equals("admin@123")) {
            // Successful login, navigate to the admin profile
            Intent intent = new Intent(MainActivity.this, AdminProfileActivity.class);
            startActivity(intent);
        } else {
            // Invalid credentials, display a toast message
            Toast.makeText(MainActivity.this, "Invalid username or password", Toast.LENGTH_SHORT).show();
        }
    }
});
```

3.) Display Toast Message for Wrong Password:

```
// Inside the loginButton onClick() event after checking the credentials
if (username.equals("Admin") && password.equals("admin@123")) {
    // Successful login, navigate to the admin profile
    Intent intent = new Intent(MainActivity.this, AdminProfileActivity.class);
    startActivity(intent);
} else {
    // Invalid credentials, display a toast message
    Toast.makeText(MainActivity.this, "Invalid username or password", Toast.LENGTH_SHORT).show();
}
```

b.) 1.)

```
class User(var username: String, var email: String, var password: String) {
    // Getter methods
    fun getUsername(): String {
        return username
    }

    fun getEmail(): String {
        return email
    }

    fun getPassword(): String {
        return password
    }

    // Setter methods (if needed)
    fun setUsername(username: String) {
        this.username = username
    }

    fun setEmail(email: String) {
        this.email = email
    }

    fun setPassword(password: String) {
        this.password = password
    }
}
```

2.)

```
import android.provider.BaseColumns

object UserContract {
    // Define the table name and column names as constants
    object UserEntry : BaseColumns {
        const val TABLE_NAME = "users"
        const val COLUMN_USERNAME = "username"
        const val COLUMN_EMAIL = "email"
        const val COLUMN_PASSWORD = "password"
    }
}
```

3.)

```
import android.content.ContentValues
import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
import android.provider.BaseColumns

class UserDbHelper(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null,
DATABASE_VERSION) {

    companion object {
        private const val DATABASE_NAME = "user.db"
        private const val DATABASE_VERSION = 1
    }

    object UserEntry : BaseColumns {
        const val TABLE_NAME = "users"
        const val COLUMN_USERNAME = "username"
        const val COLUMN_EMAIL = "email"
        const val COLUMN_PASSWORD = "password"
    }

    override fun onCreate(db: SQLiteDatabase) {
        val createTableQuery = "CREATE TABLE ${UserEntry.TABLE_NAME} (" +
                "${BaseColumns._ID} INTEGER PRIMARY KEY," +
                "${UserEntry.COLUMN_USERNAME} TEXT," +
                "${UserEntry.COLUMN_EMAIL} TEXT," +
                "${UserEntry.COLUMN_PASSWORD} TEXT)"

        db.execSQL(createTableQuery)
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        // Handle database schema upgrade if needed
        // This method is called when DATABASE_VERSION is increased
    }

    fun addUser(username: String, email: String, password: String) {
        val db = writableDatabase
        val values = ContentValues().apply {
            put(UserEntry.COLUMN_USERNAME, username)
            put(UserEntry.COLUMN_EMAIL, email)
            put(UserEntry.COLUMN_PASSWORD, password)
        }
        db.insert(UserEntry.TABLE_NAME, null, values)
        db.close()
    }
}
```

c.) 1.)

```
 // Assuming you have a reference to the Button and FragmentManager

// Set an OnClickListener for the view juice button
viewJuiceButton.setOnClickListener {
  // Create an instance of the fragment for viewing juices
  val viewJuiceFragment = ViewJuiceFragment()

  // Begin a FragmentTransaction
  val transaction = supportFragmentManager.beginTransaction()

  // Replace the current fragment with the view juice fragment
  transaction.replace(R.id.fragment_container, viewJuiceFragment)

  // Commit the transaction
  transaction.commit()
}

// Set an OnClickListener for the add juice button
addJuiceButton.setOnClickListener {
  // Create an instance of the fragment for adding juices
  val addJuiceFragment = AddJuiceFragment()

  // Begin a FragmentTransaction
  val transaction = supportFragmentManager.beginTransaction()

  // Replace the current fragment with the add juice fragment
  transaction.replace(R.id.fragment_container, addJuiceFragment)

  // Commit the transaction
  transaction.commit()
}
```

2.)

```
class JuiceAdapter(private val juices: List<Juice>) : RecyclerView.Adapter<JuiceAdapter.JuiceViewHolder>() {

  inner class JuiceViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    // Define the views in the ViewHolder
    private val juiceNameTextView: TextView = itemView.findViewById(R.id.juiceNameTextView)
    private val juiceDescriptionTextView: TextView = itemView.findViewById(R.id.juiceDescriptionTextView)

    fun bind(juice: Juice) {
      // Bind the juice data to the views
      juiceNameTextView.text = juice.name
      juiceDescriptionTextView.text = juice.description
    }
  }
```

```
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): JuiceViewHolder {
        // Inflate the item layout and create a new JuiceViewHolder
        val view = LayoutInflater.from(parent.context).inflate(R.layout.item_juice, parent, false)
        return JuiceViewHolder(view)
    }

    override fun onBindViewHolder(holder: JuiceViewHolder, position: Int) {
        // Get the juice at the specified position and bind it to the ViewHolder
        val juice = juices[position]
        holder.bind(juice)
    }

    override fun getItemCount(): Int {
        // Return the total number of juices in the list
        return juices.size
    }
}
```

3.)

```
class ViewJuicesFragment : Fragment() {
    private lateinit var recyclerView: RecyclerView
    private lateinit var juiceAdapter: JuiceAdapter
    private val juiceList: MutableList<Juice> = mutableListOf()

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        val view = inflater.inflate(R.layout.fragment_view_juices, container, false)

        // Initialize the RecyclerView
        recyclerView = view.findViewById(R.id.recyclerView)
        juiceAdapter = JuiceAdapter(juiceList)
        recyclerView.adapter = juiceAdapter
        recyclerView.layoutManager = LinearLayoutManager(requireContext())

        // Populate the juiceList with sample data (replace with your actual data retrieval logic)
        retrieveJuices()

        return view
    }

    private fun retrieveJuices() {
        // Replace this with your actual data retrieval logic
        // For example, you can fetch the juice data from a database or an API
        // Here, we add some sample juices manually for demonstration purposes
        juiceList.add(Juice("Apple Juice", "Freshly squeezed apple juice"))
        juiceList.add(Juice("Orange Juice", "Naturally sweet orange juice"))
        juiceList.add(Juice("Watermelon Juice", "Refreshing watermelon juice"))
        // Notify the adapter that the data set has changed
        juiceAdapter.notifyDataSetChanged()
    }
}
```