**Réponse questions e-commerce (Partie complément)**

**Question 1:**

**a) Décrire en les comparant les trois protocoles de messagerie électronique SMTP, POP3 et IMAP.**

**b) Quelles différences IMAP apporte-t-il par rapport à POP3 et expliquer en termes d'agents le cheminement d'un message électronique depuis sa rédaction par l'auteur jusqu'à sa lecture par le destinataire.**

**c) Expliquer comment ces agents laissent des traces dans les headers des messages et comment on peut les retrouver en programmation Java pour un message reçu.**

a) Comparaison des protocoles SMTP, POP3 et IMAP :

* **SMTP (Simple Mail Transfer Protocol) :**
  + Utilisé pour l'envoi de courriels.
  + Fonctionne sur le port 25 en TCP.
  + Protocole applicatif de transfert de messages électroniques.
* **POP3 (Post Office Protocol version 3) :**
  + Protocole de récupération des messages électroniques.
  + Fonctionne sur le port 110 en TCP.
  + Principales commandes incluent user, pass, retr, dele.
  + Permet de télécharger les e-mails sur l'appareil local de l'utilisateur.
* **IMAP (Internet Message Access Protocol) :**
  + Protocole de récupération des messages électroniques.
  + Fonctionne sur les ports 143 (non chiffré) ou 993 (chiffré) en TCP.
  + Permet une gestion plus flexible des e-mails par rapport à POP3, comme l'accès à plusieurs boîtes aux lettres, la gestion des drapeaux de messages, etc.

b) Différences entre IMAP et POP3 et cheminement d'un message électronique :

* **Différences entre IMAP et POP3 :**
  + IMAP permet de gérer les e-mails directement sur le serveur, sans nécessiter un téléchargement immédiat. Cela permet aux utilisateurs d'accéder à leurs e-mails depuis plusieurs clients et de partager des dossiers.
  + POP3, en revanche, télécharge les e-mails sur l'appareil local de l'utilisateur et, souvent, les supprime du serveur.
* **Cheminement d'un message électronique :**
  + **Rédaction :** L'auteur compose un e-mail dans un client de messagerie (MUA - Mail User Agent).
  + **Envoi :** Le MUA envoie le message au serveur de messagerie de l'expéditeur via SMTP.
  + **Réception :** Le serveur de messagerie du destinataire reçoit le message et le stocke.
  + **Lecture :** Le destinataire utilise un MUA configurer avec IMAP ou POP3 pour récupérer et lire le message depuis le serveur de messagerie.

c) Traces dans les headers des messages et récupération en Java :

* **Traces dans les headers :** Les agents laissent des traces sous forme de headers dans les e-mails, indiquant les chemins par lesquels le message est passé. Ces headers incluent des informations comme l'adresse IP du serveur d'envoi, le chemin pris par le message, et d'autres métadonnées.
* **Récupération en Java :** Pour retrouver ces informations dans un message reçu, on peut utiliser le package JavaMail (javax.mail et javax.mail.internet). En utilisant des classes comme MimeMessage, on peut accéder aux headers d'un message. Par exemple, la méthode getAllHeaders() de MimeMessage permet d'obtenir une énumération de tous les headers d'un message. Pour chaque header, on peut ensuite récupérer le nom et la valeur associée.

**Question 2:**

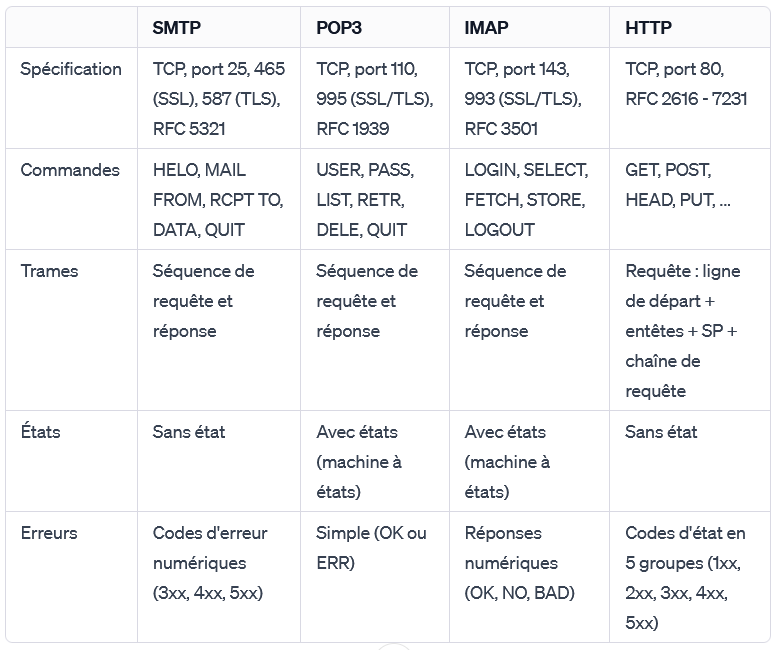
**a) Dans les librairies javax.mail, décrire et expliquer les rôles des interfaces, classes abstraites et classes instanciables nécessaires à l'envoi d'un message sans pièces attachées.**

**b) Dresser le tableau synoptique de description synthétique des protocoles classiques SMTP, POP3, IMAP, HTTP.**

a) Rôles des interfaces, classes abstraites et classes instanciables dans javax.mail pour l'envoi d'un message sans pièces attachées :

* **Interfaces et Classes Abstraites** :
  + **javax.mail.Session** : Représente une session de messagerie, utilisée pour configurer les propriétés et obtenir les objets Store et Transport.
  + **javax.mail.Message** : Classe abstraite représentant un message électronique. Elle définit les attributs de base d'un message (comme les adresses de l'expéditeur et du destinataire, le sujet, etc.).
  + **javax.mail.internet.MimeMessage** : Sous-classe de Message, utilisée pour créer des messages MIME avec du texte, des images, etc. Pour un message simple sans pièces jointes, seul le texte est utilisé.
* **Classes Instanciables** :
  + **javax.mail.Transport** : Utilisée pour envoyer des messages. C'est une classe abstraite, mais elle fournit des méthodes statiques pour l'envoi de messages.
  + **javax.mail.internet.InternetAddress** : Représente une adresse de messagerie Internet. Utilisée pour spécifier les adresses e-mail des expéditeurs et des destinataires.

b) Tableau synoptique des protocoles SMTP, POP3, IMAP, HTTP :



**Question 3:**

**a) Dans les librairies javax.activation, décrire et expliquer les rôles des interfaces, classes abstraites et classes instanciables nécessaires à l'envoi d'un message avec pièces attachées.**

**b) Donnez un exemple de programmation pour la récupération d’un mail contenant plusieurs pièces attachées (fichiers texte, image).**

**c) Expliquer en quoi l'examen des types MIME et de la "disposition" des pièces attachées intervient dans la programmation de cette réception de messages composites.**

a) Dans les librairies javax.activation, les rôles des interfaces, classes abstraites et classes instanciables nécessaires à l'envoi d'un message avec pièces attachées sont :

* **DataSource** : C'est une interface qui représente une source de données, capable d'être lue (InputStream) ou écrite (OutputStream). Elle est utilisée pour décrire les données contenues dans une pièce jointe.
  + getContentType() renvoie le type MIME de la donnée.
  + getName() renvoie le nom de l'objet qui représente la donnée.
  + getInputStream() et getOutputStream() permettent d'accéder aux données.
* **DataHandler** : Cette classe instanciable joue le rôle d'intermédiaire entre une DataSource et un consommateur de données, fournissant les données à un BodyPart ou à un MimeMessage.
* **FileDataSource** : C'est une implémentation concrète de l'interface DataSource qui lit les données d'un fichier.
* **MimeBodyPart** et **MimeMultipart** : MimeBodyPart est une classe concrète qui représente une partie du message, qui peut être du texte, une image, etc. MimeMultipart est une classe instanciable qui contient plusieurs BodyPart et gère des messages composites.

b) Voici un exemple de code en Java pour la récupération d’un mail contenant plusieurs pièces attachées :

**// ... (Configuration de la session et connexion)**

**Message[] messages = folder.getMessages(); // Récupération des messages**

**for (Message message : messages) {**

**Multipart multipart = (Multipart) message.getContent(); // Casting en Multipart**

**BodyPart bodyPart = multipart.getBodyPart(j); // Accès à chaque partie**

**if (Part.ATTACHMENT.equalsIgnoreCase(bodyPart.getDisposition())) {**

**InputStream is = bodyPart.getInputStream(); // Récupération du flux de données**

**// ... (Enregistrement du flux dans un fichier)**

**}**

**}**

**// ... (Fermeture du Folder et du Store)**

c) L'examen des types MIME et de la "disposition" des pièces attachées intervient dans la programmation de la réception de messages composites pour déterminer la nature des différentes parties du message. Le type MIME indique le format des données (texte, image, etc.), et la disposition (comme "attachment" pour une pièce jointe ou "inline" pour un contenu intégré) indique comment la partie doit être traitée. Ces informations sont essentielles pour traiter correctement chaque partie du message, notamment pour savoir si une partie doit être enregistrée comme fichier ou affichée comme partie du message.

**Question 4:**

**Décrire et expliquer le contenu d'un fichier apk.**

**Quels sont les outils qui interviennent dans sa construction ?**

**En particulier, expliquer le contenu et le rôle de la classe R.java.**

**Contenu d'un fichier APK** :

Un fichier APK (Android Package) est le package de distribution final d'une application Android​​. Il contient :

* Un répertoire META-INF qui inclut les signatures et la liste des ressources de l'application.
* Un fichier .dex contenant le bytecode Dalvik exécutable sur la machine virtuelle Android.
* Un fichier AndroidManifest.xml qui décrit les permissions, les activités, les services, etc., de l'application.
* Les fichiers de ressources dans le répertoire res et un fichier .arsc qui contient la table des ressources en forme binaire, aussi connue comme la "ressource compilée".

**Outils de construction d'un APK** :

Pour construire un fichier APK, plusieurs outils sont utilisés durant le processus de développement :

* **Android SDK Tools** : Comprennent le compilateur aapt pour empaqueter, compiler les ressources et générer le fichier R.java, l'outil dx pour convertir les fichiers .class en .dex.
* **Android Studio / Eclipse** : Les IDEs intégrés avec le SDK pour le développement, le débogage et la gestion des projets Android.
* **Gradle / Apache Ant / Maven** : Outils de construction et de gestion de dépendances qui automatisent le processus de compilation et empaquetage.

**Classe R.java** :

La classe R.java est une classe générée automatiquement pendant le processus de build qui contient des identifiants uniques pour toutes les ressources de l'application (layout, strings, widgets, etc.)​​. Elle joue un rôle crucial en permettant l'accès à ces ressources dans le code Java. Par exemple, si on a un bouton dans notre fichier XML de layout avec l'ID @+id/bouton\_ok, on peut accéder à ce bouton en Java avec R.id.bouton\_ok. Cela permet de garder une séparation claire entre le code et les ressources graphiques/textuelles, facilitant la maintenance et la localisation des applications.

**Question 5:**

**Décrire l'architecture Android à 5(6) couches en en décrivant les éléments principaux et décrire le bytecode particulier utilisé par les JVMs d'Android.**

**Expliquer les caractéristiques des JVM Dalvik et ART et ce qui les différencie de la JVM classique.**

1. **Architecture Android à 5(6) couches** : L'architecture Android est présentée comme un ensemble de couches, chacune contenant des composants spécifiques :
   * **Couche Applications** : Applications Android comme les contacts, le navigateur.
   * **Couche Framework d'Application** : Inclut des managers (comme l'Activity Manager) et des services (comme les services de téléphonie).
   * **Couche Bibliothèques** : Bibliothèques natives C/C++ utilisées par divers composants du système.
   * **Couche Runtime Android** : Comprend les machines virtuelles Dalvik et ART.
   * **Couche HAL (Hardware Abstraction Layer)** : Interface pour le matériel.
   * **Couche Linux Kernel** : Noyau Linux gérant les fonctions de base du système.
2. **Bytecode particulier utilisé par les JVMs d'Android** : Le bytecode utilisé par les JVMs d'Android est différent de celui utilisé en Java. Android utilise un format de fichier DEX (Dalvik Executable), spécialement conçu pour être efficace en termes de mémoire et de traitement sur des appareils mobiles.
3. **Caractéristiques des JVM Dalvik et ART** :
   * **Dalvik** : Première JVM utilisée par Android, optimisée pour les systèmes avec des ressources limitées. Dalvik utilise un format DEX et exécute les applications en utilisant la technique de JIT (Just-In-Time) compilation.
   * **ART (Android Runtime)** : Introduit plus tard, remplace Dalvik. ART améliore les performances en utilisant la compilation AOT (Ahead-Of-Time), qui compile le bytecode en code machine lors de l'installation de l'application, réduisant ainsi le temps de démarrage des applications.
4. **Différences avec la JVM classique** : Contrairement à la JVM classique, Dalvik et ART sont conçus pour des appareils avec des ressources limitées en termes de processeur et de mémoire. Ils utilisent un format de fichier différent (DEX) et des techniques de compilation différentes pour optimiser les performances sur les appareils mobiles.

**Question 6:**

**Décrire la structure des répertoires (et leur principal contenu) qui correspondent à une application Android comportant une activité classique avec images, couleurs, menus, textes personnalisés selon la langue et la manière d'en faire usage dans le code Java d'une activité.**

* **Répertoire 'src'** : Contient le code source Java de l'application, y compris les activités (classes dérivées de Activity).
* **Répertoire 'res'** : Stocke les ressources non-code, subdivisées en plusieurs dossiers :
  + **'layout'** : Fichiers XML définissant l'interface utilisateur.
  + **'drawable'** : Images et graphiques.
  + **'values'** : Ressources telles que les chaînes de caractères, dimensions, et couleurs. Il peut inclure des sous-dossiers pour la personnalisation selon la langue.
  + **'menu'** : Définitions XML des menus de l'application.

Dans le code Java d'une activité, ces ressources sont référencées principalement via la classe auto-générée R. Par exemple, pour accéder à un layout, on utilise R.layout.nom\_du\_layout, et pour une chaîne de caractères, R.string.nom\_de\_la\_chaine. Cette structure permet une organisation efficace et une référence facile aux différentes ressources utilisées dans l'application.

Haut du formulaire

Bas du formulaire

**Question 7:**

**Sous Android, décrire à partir d'un exemple d'application ne comportant qu'une seule activité, les diverses manières de programmer la réponse à un événement graphique et la manière de gérer un composant comme une ListView selon le modèle MVC.**

**Réponse à un événement graphique** :

1. **Gestion avec un attribut dans le fichier XML** : On peut définir un gestionnaire d'événement directement dans le fichier de layout XML en utilisant l'attribut android:onClick. Par exemple, si on a un bouton dans notre fichier XML, on peut ajouter android:onClick="nomDeLaMethode" et définir cette méthode dans notre activité avec la signature public void nomDeLaMethode(View view).
2. **Gestion avec un listener explicite** : On peut définir un listener dans notre code Java. Dans ce cas, l'activité peut implémenter l'interface OnClickListener, et on peut enregistrer l'activité comme listener pour le bouton avec button.setOnClickListener(this)​​.

**Gestion d'une ListView selon le modèle MVC** :

* **Modèle** : Représenté par une classe de données, par exemple, Consultation si notre application gère des consultations médicales. Cette classe contiendrait les informations relatives à une consultation.
* **Vue** : La ListView elle-même, définie dans le fichier XML de layout, qui affiche les données à l'utilisateur.
* **Contrôleur** : Un adaptateur qui fait le pont entre les données et la vue. En Android, cela peut être un ArrayAdapter ou un adaptateur personnalisé, qui prend les données du modèle et les convertit en vues à afficher dans la ListView.

Dans le code de l'activité, on instancie notre adaptateur avec les données du modèle et l'associe à la ListView avec listView.setAdapter(monAdaptateur).

**Question 8:**

**Sous Android, décrire à partir d'un exemple d'application ne comportant qu'une seule activité l'utilisation d'un AsyncTask qui accède à un serveur vendant des actions.**

**Expliquer le fonctionnement du modèle MVC pour les composants graphiques d'Android à partir de l'affichage par l'activité des actions achetées.**

1. **Utilisation d'un AsyncTask dans une Application Android** :
   * Un AsyncTask est utilisé pour effectuer des opérations en arrière-plan sans bloquer l'interface utilisateur. Dans une application avec une seule activité, on peut utiliser un AsyncTask pour accéder à un serveur et récupérer des informations sur des actions.
   * Dans le doInBackground de l'AsyncTask, on fera la requête au serveur, et dans onPostExecute, on met à jour l'interface utilisateur avec les résultats obtenus.
2. **Modèle MVC pour les Composants Graphiques d'Android** :
   * Le modèle MVC (Modèle-Vue-Contrôleur) peut être illustré par la manière dont les données (Modèle) sont affichées dans l'interface utilisateur (Vue) et gérées par la logique de l'application (Contrôleur).
   * Par exemple, pour afficher des actions achetées, le modèle contiendrait les données des actions, la vue serait l'interface utilisateur affichant ces données, et le contrôleur serait la logique dans l'activité qui récupère les données du modèle et les transmet à la vue.

**Question 9:**

**Expliquer comment une communication réseau est sécurisée par SSL/TLS (SSL-Record), une fois le SSL-Handshake réalisé (donc sans le décrire).**

**Pourquoi parle-t-on de "sous-protocole(s)" ?**

**Qu'entend-on par "variables de session" et "variable de connexion".**

1. **Fonctionnement de SSL/TLS après le SSL-Handshake** :
   * **SSL Record** : Après l'établissement d'une connexion sécurisée via le SSL-Handshake, les données sont transmises sous forme de records SSL.
   * **Processus de Sécurisation** : Les données sont d'abord compressées, puis un HMAC (Hash-based Message Authentication Code) est généré pour assurer l'intégrité et l'authentification des données. Ensuite, les données sont chiffrées en utilisant la clé secrète négociée pendant le handshake.
   * **Structure du Record SSL** : Chaque record SSL contient un en-tête (header) avec le type de contenu, la version du protocole et la longueur du message, suivi du bloc de données chiffrées.
2. **"Sous-protocole(s)" dans SSL/TLS** :
   * Les "sous-protocoles" dans SSL/TLS font référence à des composants ou des fonctions spécifiques au sein du protocole SSL/TLS global. Par exemple, le SSL Record est un sous-protocole qui définit la manière de traiter et de sécuriser les données.
3. **Variables de Session et de Connexion** :
   * **Variables de Session** : Ces variables sont établies lors du handshake initial et restent inchangées pendant toute la durée de la session. Elles comprennent le numéro de session, les certificats des pairs, l'algorithme de chiffrement (Cipher spec), le secret principal (Master secret), la méthode de compression et si la session est réutilisable (Is resumable)​​.
   * **Variables de Connexion** : Ces variables sont spécifiques à chaque connexion individuelle établie pendant une session SSL/TLS. Elles comprennent des éléments tels que les nombres aléatoires échangés lors du handshake, les secrets HMAC, les clés de chiffrement et les vecteurs d'initialisation pour le chiffrement CBC. Ces variables sont mises à jour à chaque nouvelle connexion ou reconnexion​​.

**Question 10:**

**Décrire en détails le SSL-Handshake.**

**En particulier, expliquer en quoi on peut considérer qu'il est adaptable à différentes situations.**

**Pourquoi parle-t-on de "sous-protocoles" ?**

**Le SSL-Handshake** :

* **Objectif** : Le processus de handshake SSL est conçu pour négocier les paramètres de sécurité entre le client et le serveur pour une session sécurisée. Il établit les clés de cryptage et d'authentification, ainsi que les algorithmes à utiliser.

**Les étapes du SSL-Handshake :**

1. **ClientHello** : Le client envoie un message "ClientHello" pour initier une session SSL/TLS. Ce message comprend la version du protocole SSL/TLS que le client supporte, une liste de suites cryptographiques proposées, une liste de méthodes de compression supportées, et un identifiant de session pour une reprise de session éventuelle.
2. **ServerHello** : En réponse, le serveur envoie un message "ServerHello" qui sélectionne la suite cryptographique, la méthode de compression et la version du protocole SSL/TLS à utiliser. Le serveur peut également envoyer un identifiant de session pour la reprise de session.
3. **Certificate (Serveur)** : Le serveur envoie son certificat au client. Ce certificat est utilisé pour authentifier le serveur et contient la clé publique du serveur.
4. **ServerKeyExchange** : Si la suite cryptographique sélectionnée requiert un échange de clés supplémentaire (par exemple, pour les suites cryptographiques basées sur Diffie-Hellman), le serveur envoie un message "ServerKeyExchange" contenant les informations nécessaires à l'échange de clés.
5. **ServerHelloDone** : Le serveur envoie un message "ServerHelloDone" pour indiquer que la phase initiale du handshake est terminée.
6. **ClientKeyExchange** : Le client envoie un message "ClientKeyExchange" qui contient les informations cryptographiques nécessaires pour établir la clé de session.
7. **ChangeCipherSpec (Client vers Serveur)** : Le client envoie un message "ChangeCipherSpec" pour indiquer que les messages suivants seront protégés avec les clés négociées.
8. **ChangeCipherSpec (Serveur vers Client)** : Le serveur envoie également un message "ChangeCipherSpec" pour indiquer que les messages suivants seront protégés avec les clés négociées.
9. **Echanges de Données Cryptées** : Les messages suivants entre le client et le serveur sont cryptés et sécurisés en utilisant les clés négociées.

**Adaptabilité** :

Le handshake est adaptable car il permet aux deux parties de négocier les détails de la connexion, comme les algorithmes de cryptage et de hachage, ainsi que la gestion des certificats et des clés. Cela permet d'adapter la sécurité en fonction des capacités et des besoins des deux entités communicantes.

**Sous-protocoles dans SSL/TLS** :

SSL/TLS utilise des "sous-protocoles" pour gérer différentes parties du processus de communication sécurisée. Par exemple, le "Handshake Protocol" est utilisé pour négocier la suite cryptographique et les clés, tandis que le "ChangeCipherSpec Protocol" est utilisé pour signaler un changement dans la transmission des données cryptées. Ces sous-protocoles permettent une gestion modulaire et efficace des différentes étapes de la communication sécurisée.

**Question 11:**

**Expliquer le code Java utilisant SSL à substituer au code classique d'un serveur TCP/IP dont on désire sécuriser les communications.**

**En particulier, décrire les keystores nécessaires au bon fonctionnement de ce code et leur utilisation, côté serveur et côté client.**

**Pourquoi voit-on intervenir dans javax.net.\* un si grand nombre d'interfaces et de classes abstraites ?**

1. **Code Java utilisant SSL pour un Serveur TCP/IP** :
   * Pour sécuriser un serveur TCP/IP classique avec SSL, on remplace les sockets classiques par des sockets SSL (SSLSocket, SSLServerSocket) fournis par les classes SSLSocketFactory et SSLServerSocketFactory.
   * Le code initial de création et d'écoute sur un serveur socket doit être modifié pour utiliser ces classes SSL, ce qui assure une communication sécurisée.
2. **Keystores Nécessaires** :
   * **Côté Serveur** : Un keystore côté serveur contient la clé privée du serveur et le certificat public. Ce keystore est utilisé pour initialiser le SSLContext du serveur.
   * **Côté Client** : Un truststore (type de keystore) est utilisé côté client pour stocker les certificats de confiance, y compris le certificat public du serveur.
3. **Utilisation des Keystores** :
   * Les keystores sont chargés en mémoire et utilisés pour initialiser le SSLContext.
   * Ce SSLContext est ensuite utilisé pour créer des SSLSocketFactory ou SSLServerSocketFactory qui à leur tour créent des sockets SSL sécurisés.
4. **Pourquoi tant d'interfaces et de classes abstraites dans javax.net.ssl ?** :
   * Le package javax.net.ssl contient un grand nombre d'interfaces et de classes abstraites pour offrir une flexibilité et une extensibilité dans la gestion des protocoles de sécurité SSL et TLS.
   * Ces interfaces et classes abstraites permettent une implémentation personnalisée et la prise en charge de différents algorithmes de cryptographie et méthodes de gestion de clés, rendant le package adaptable à divers besoins de sécurité.

En résumé, pour sécuriser les communications d'un serveur TCP/IP avec SSL en Java, on remplace les sockets classiques par des sockets SSL et on utilise des keystores pour gérer les clés et certificats nécessaires. Les nombreuses interfaces et classes abstraites dans javax.net.ssl fournissent la flexibilité nécessaire pour gérer la sécurité des communications réseau.

**Question 12:**

**Expliquer ce qu'est un « intent » et les différents mécanismes qui permettent de les mettre en œuvre.**

**Définition d'un Intent** :

* Un intent est un message asynchrone qui spécifie une action à effectuer.
* Il contient l'URI (Uniform Resource Identifier) du composant à manipuler ou de l'action à initier.
* Les intents permettent une communication entre les activités (écrans d'une application), services (processus en arrière-plan) et autres composants applicatifs​​.

**Mécanismes de Mise en Œuvre** :

1. **Intents Explicites** :
   * Utilisés pour démarrer un composant spécifique au sein de votre propre application.
   * Par exemple, démarrer une nouvelle activité en indiquant explicitement la classe de l'activité cible.
   * Code exemple : Intent intent = new Intent(this, MaNouvelleActivite.class); startActivity(intent);
2. **Intents Implicites** :
   * Utilisés pour demander une action qui peut être effectuée par n'importe quel composant d'une autre application qui accepte l'intent.
   * Par exemple, afficher une page Web ou ouvrir une carte. L'intent ne spécifie pas directement le composant à utiliser ; il déclare une action à effectuer, et le système trouve le composant le plus approprié pour cette action.
   * Code exemple : Intent intent = new Intent(Intent.ACTION\_VIEW, Uri.parse("http://www.exemple.com")); startActivity(intent);
3. **Filtres d'Intent** :
   * Utilisés dans les fichiers AndroidManifest.xml des applications pour déclarer les capacités de réception des intents.
   * Permettent à une application de spécifier quelles actions elle peut entreprendre et sous quelles conditions.
   * Par exemple, un filtre d'intent peut indiquer qu'une activité est capable de visualiser des données de type image.
4. **Transmission de Données avec Intents** :
   * Les données peuvent être attachées aux intents sous forme de "extras" (données supplémentaires) à l'aide de méthodes comme putExtra().
   * Les données peuvent être récupérées dans l'activité cible avec des méthodes telles que getIntent().getStringExtra("maCle").
5. **Intentions de Résultat** :
   * Permettent de recevoir un résultat d'une autre activité.
   * Par exemple, lancer une activité pour prendre une photo et retourner l'image capturée.

Les intents sont donc un outil essentiel pour la gestion de la communication et de l'interaction entre les composants dans les applications Android, offrant une grande flexibilité dans la manière dont les composants peuvent interagir.

**Question 13:**

**Expliquez et illustrez les différences (améliorations, mécanisme, …) entre les versions TLS précédentes la 1.3 et TLS 1.3**

**Différences entre TLS 1.2 et TLS 1.3** :

* + **Processus de Handshake Réduit** : TLS 1.3 a simplifié le processus de handshake, réduisant le nombre d'allers-retours nécessaires pour établir une connexion sécurisée. Cela améliore la vitesse et l'efficacité de la connexion.
  + **Améliorations de la Sécurité** : TLS 1.3 a supprimé des algorithmes de chiffrement plus anciens et moins sécurisés qui étaient encore présents dans TLS 1.2, comme RC4, DES, 3DES, et les chiffrements basés sur CBC.
  + **Support de Perfect Forward Secrecy (PFS)** : TLS 1.3 exige l'utilisation de la PFS, qui assure qu'une clé compromise ne peut pas être utilisée pour déchiffrer les sessions de communication passées.
  + **Chiffrements Authentifiés** : TLS 1.3 utilise des modes de chiffrement authentifiés comme AES-GCM, qui fournissent à la fois le chiffrement et l'intégrité des données.
  + **Simplification de la Négociation Cryptographique** : TLS 1.3 a éliminé la négociation complexe des suites cryptographiques, en s'en tenant à un ensemble réduit et plus sûr de suites.

Ces différences indiquent que TLS 1.3 est une avancée significative par rapport à ses prédécesseurs en termes de sécurité, d'efficacité et de rapidité. Il élimine les faiblesses connues des versions précédentes et se concentre sur des protocoles de chiffrement plus forts et plus efficaces.

**Question 14:**

**Termes à définir brièvement, remettre dans son contexte, acronyme de … , utilité(s), rôle(s), exemple d’application.**

1. **MFS (Message Format Specification)** :
   * Contexte : Utilisé dans la messagerie électronique et d'autres systèmes de communication.
   * Acronyme : Message Format Specification.
   * Utilité/Rôle : Définit le format et la structure des messages électroniques.
   * Exemple d'Application : Spécification de la structure des e-mails.
2. **RFC822** :
   * Contexte : Standard pour le format des messages électroniques.
   * Acronyme : Request For Comments 822.
   * Utilité/Rôle : Définit le format des en-têtes des messages électroniques.
   * Exemple d'Application : Utilisé comme base pour les en-têtes d'e-mails.
3. **SMTP (Simple Mail Transfer Protocol)** :
   * Contexte : Protocole de communication pour l'envoi d'e-mails.
   * Acronyme : Simple Mail Transfer Protocol.
   * Utilité/Rôle : Permet de transférer des e-mails entre serveurs.
   * Exemple d'Application : Envoi d'e-mails par des clients de messagerie.
4. **POP3 (Post Office Protocol version 3)** :
   * Contexte : Protocole de récupération d'e-mails.
   * Acronyme : Post Office Protocol version 3.
   * Utilité/Rôle : Permet de récupérer des e-mails depuis un serveur pour les stocker localement.
   * Exemple d'Application : Récupération d'e-mails par des clients comme Outlook.
5. **IMAP (Internet Message Access Protocol)** :
   * Contexte : Protocole de gestion d'e-mails.
   * Acronyme : Internet Message Access Protocol.
   * Utilité/Rôle : Permet de gérer les e-mails directement sur le serveur.
   * Exemple d'Application : Utilisé par les clients de messagerie pour gérer les e-mails sur plusieurs appareils.
6. **APK (Android Package)** :
   * Contexte : Format de fichier pour la distribution et l'installation d'applications sur Android.
   * Acronyme : Android Package.
   * Utilité/Rôle : Contient tous les fichiers nécessaires pour installer une application sur Android.
   * Exemple d'Application : Distribution d'applications via Google Play Store.
7. **RMTA (Remote Message Transfer Agent)** :
   * Contexte : Utilisé dans l'envoi d'e-mails.
   * Acronyme : Remote Message Transfer Agent.
   * Utilité/Rôle : Transfère les e-mails d'un serveur à un autre.
   * Exemple d'Application : Serveurs SMTP qui transfèrent des e-mails.
8. **FMTA (Foreign Mail Transfer Agent)** :
   * Contexte : Utilisé dans les systèmes de messagerie électronique.
   * Acronyme : Foreign Mail Transfer Agent.
   * Utilité/Rôle : Transfère les e-mails entre différents domaines ou systèmes de messagerie.
   * Exemple d'Application : Serveurs SMTP interconnectant différents domaines.
9. **RMDA (Remote Mail Delivery Agent)** :
   * Contexte : Utilisé dans les systèmes de messagerie électronique.
   * Acronyme : Remote Mail Delivery Agent.
   * Utilité/Rôle : Gère la livraison des e-mails aux boîtes aux lettres distantes.
   * Exemple d'Application : Livraison d'e-mails aux clients de messagerie.
10. **MRA (Mail Retrieval Agent)** :
    * Contexte : Utilisé dans la récupération d'e-mails.
    * Acronyme : Mail Retrieval Agent.
    * Utilité/Rôle : Récupère les e-mails depuis un serveur pour les livrer au client.
    * Exemple d'Application : Clients utilisant POP3 ou IMAP pour récupérer des e-mails.
11. **RMUA (Remote Mail User Agent)** :
    * Contexte : Utilisé dans les systèmes de messagerie électronique.
    * Acronyme : Remote Mail User Agent.
    * Utilité/Rôle : Interface utilisateur permettant d'accéder et de gérer des e-mails sur un serveur distant.
    * Exemple d'Application : Clients de messagerie comme Microsoft Outlook.
12. **SMUA (Simple Mail User Agent)** :
    * Contexte : Utilisé dans les systèmes de messagerie électronique.
    * Acronyme : Simple Mail User Agent.
    * Utilité/Rôle : Client de messagerie simplifié pour envoyer et recevoir des e-mails.
    * Exemple d'Application : Clients de messagerie basiques.
13. **R.java** :
    * Contexte : Utilisé dans le développement d'applications Android.
    * Acronyme : Non applicable.
    * Utilité/Rôle : Fichier généré automatiquement qui contient des références à toutes les ressources de l'application.
    * Exemple d'Application : Accès aux ressources comme les identifiants d'éléments d'interface dans le code Android.
14. **.arsc (Android Resource)** :
    * Contexte : Utilisé dans les applications Android.
    * Acronyme : Android Resource.
    * Utilité/Rôle : Fichier contenant les ressources compilées de l'application.
    * Exemple d'Application : Utilisé par le système Android pour accéder aux ressources de l'application.
15. **JVM/ART (Java Virtual Machine/Android Runtime)** :
    * Contexte : Utilisé dans l'exécution des applications Android.
    * Acronyme : Java Virtual Machine/Android Runtime.
    * Utilité/Rôle : Environnement d'exécution pour les applications Android.
    * Exemple d'Application : Exécution des applications Android avec optimisation de performance.
16. **FindViewById** :
    * Contexte : Utilisé dans le développement d'applications Android.
    * Acronyme : Non applicable.
    * Utilité/Rôle : Méthode pour retrouver une vue par son identifiant dans le code Java/Kotlin.
    * Exemple d'Application : Accès aux éléments d'interface utilisateur dans une activité Android.
17. **Intent** :
    * Contexte : Utilisé dans le développement d'applications Android.
    * Acronyme : Non applicable.
    * Utilité/Rôle : Mécanisme pour demander une action de la part d'autres composants ou applications.
    * Exemple d'Application : Démarrer une nouvelle activité ou appeler un service.
18. **AsyncTask** :
    * Contexte : Utilisé dans le développement d'applications Android.
    * Acronyme : Non applicable.
    * Utilité/Rôle : Permet d'exécuter des opérations en arrière-plan et de publier les résultats sur le thread de l'interface utilisateur.
    * Exemple d'Application : Téléchargement de données en arrière-plan.
19. **SSL-Record** :
    * Contexte : Utilisé dans le protocole SSL/TLS.
    * Acronyme : Secure Sockets Layer Record.
    * Utilité/Rôle : Format de données pour le chiffrement et l'authentification dans le protocole SSL/TLS.
    * Exemple d'Application : Sécurisation des données transmises sur une connexion SSL/TLS.
20. **SSL-Handshake** :
    * Contexte : Utilisé dans le protocole SSL/TLS.
    * Acronyme : Secure Sockets Layer Handshake.
    * Utilité/Rôle : Processus de négociation des paramètres de sécurité pour une connexion SSL/TLS.
    * Exemple d'Application : Établissement d'une connexion sécurisée pour le transfert de données.
21. **Keystore** :
    * Contexte : Utilisé dans la gestion de la sécurité des applications.
    * Acronyme : Non applicable.
    * Utilité/Rôle : Stockage sécurisé des clés cryptographiques et des certificats.
    * Exemple d'Application : Stockage des clés privées et des certificats dans les applications sécurisées.