NFCCasino

Table des matières

I. Présentation du projet

Le système a pour but de supprimer les billets et les croupiers dans les casinos afin que pole-Emploi soit content et qu'on réduise les vols.

Pour cela nous faudra une réaliser un nouveau système de payement :

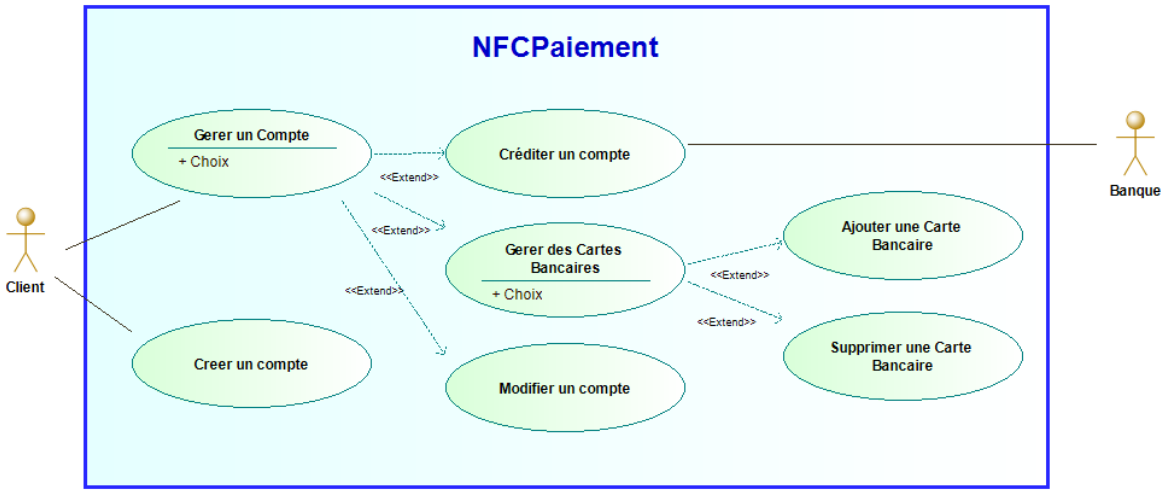
Ce système axé sur les Téléphone NFC permettra de de créer un compte sur le casino et de crédité ce compte une fois créer. Une fois que le compte est crédité, le client peux rejoindre les différentes tables de jeu (Poker, BlackJack, Roulette, …).

De plus, il faudra réaliser un des jeux:

Pour rejoint la table, il faut approcher son téléphone vers un lecture NFC sur une place libre. Le système affichera une interface permettant de miser, de s'informer sur les joueurs de la table ou les règles du jeu en question.

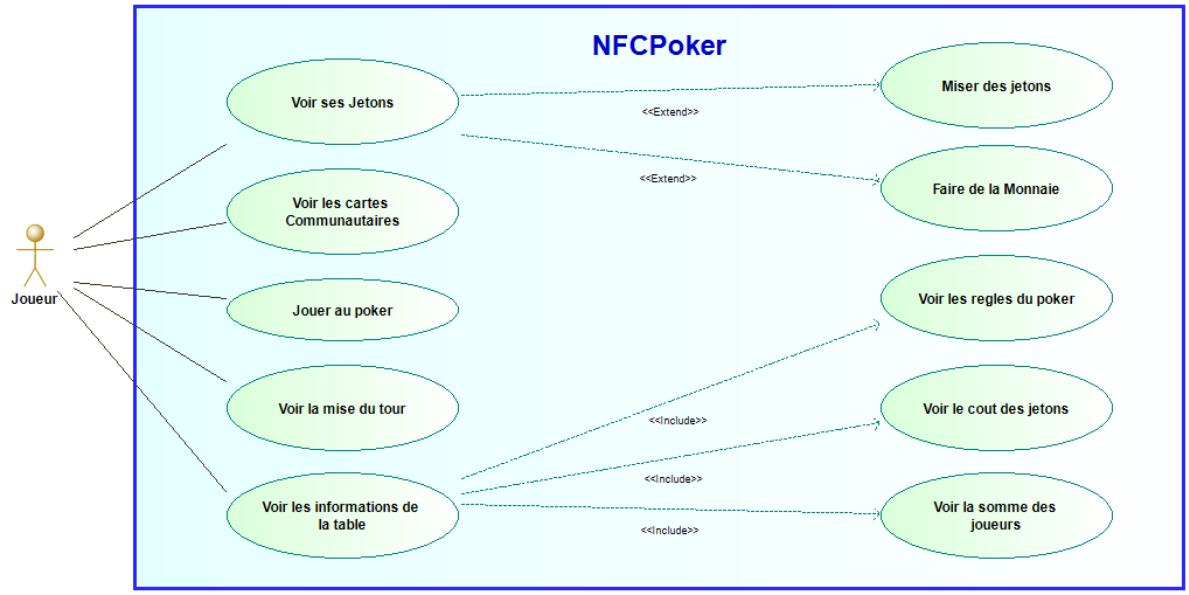
II. Présentation des systèmes

1) NFCPaiement



Le Client doit pouvoir créer un compte en entrant son nom, prénom et un mot de passe. De plus il doit pouvoir modifier ses informations, gérer ses cartes bancaires. La gestion de carte bancaire est simple, on peut sois ajouter une carte avec son numéro de carte, la date d'expiration et le cryptogramme visuel ou en supprimer. Si le compte possède une carte bancaire alors on peut créditer son compte.

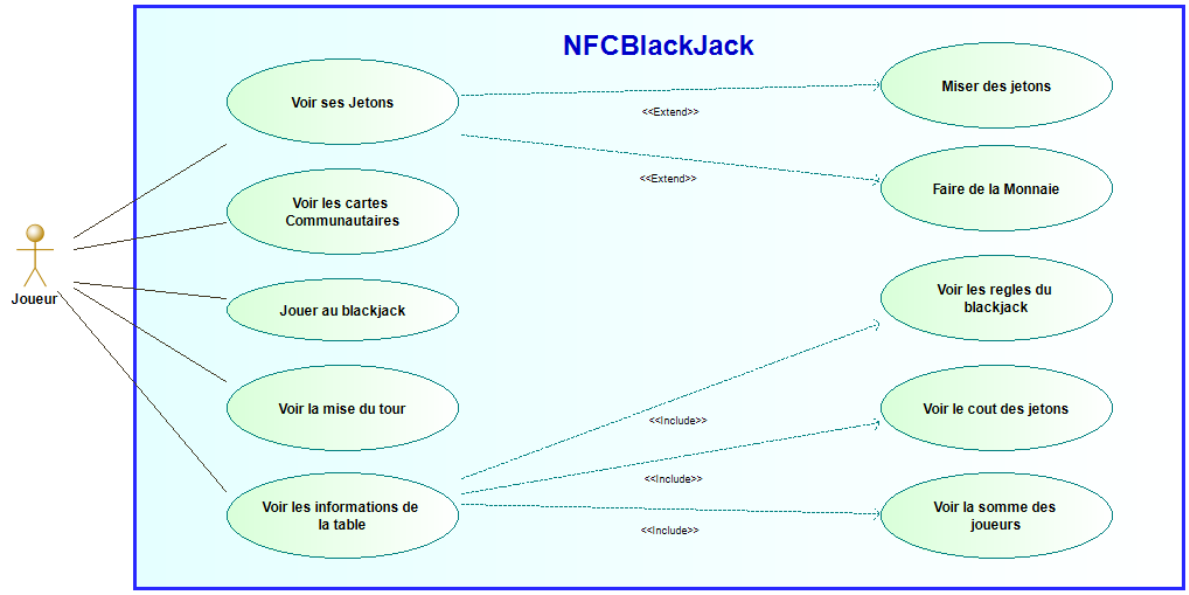
2) NFCPoker



Le Client doit jouer au poker, miser des jetons, faire de la monnaie. De plus il doit pouvoir voir les règles du poker, le cout des jetons et les sommes totales des joueurs se trouvant sur la table.

De plus tous les joueurs doivent pouvoir voir les cartes communautaires (flop, turn, river) et la mise totale du tour.

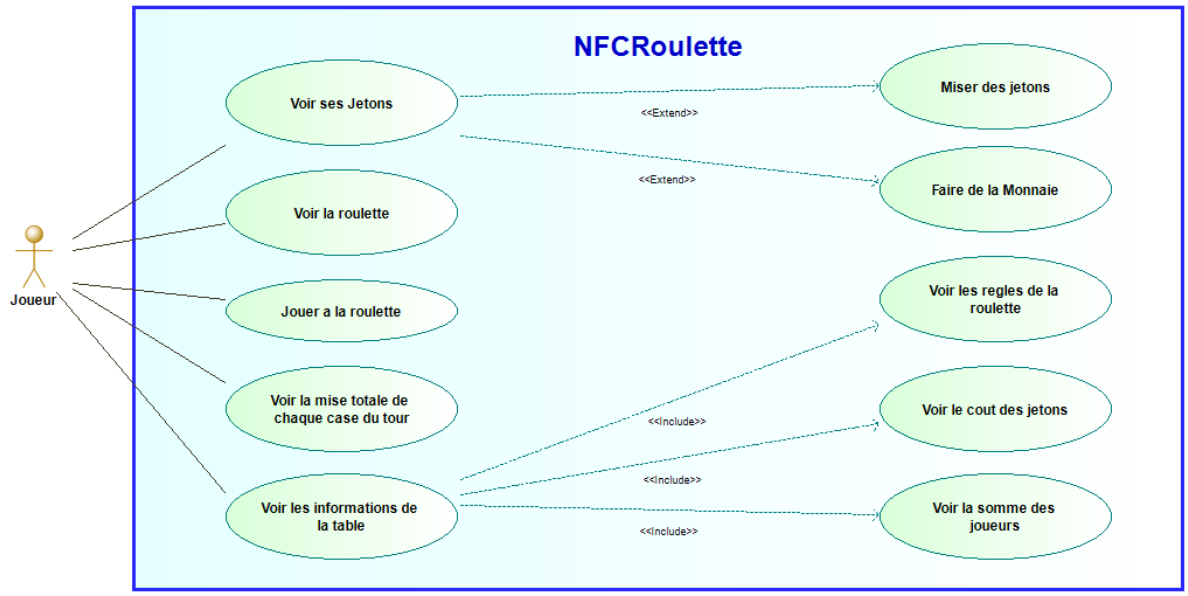
3) NFCBlackJack



Le Client doit jouer au blackjack, miser des jetons, faire de la monnaie. De plus il doit pouvoir voir les règles du blackjack, le cout des jetons et les sommes totales des joueurs se trouvant sur la table.

Tous les joueurs doivent pouvoir voir les cartes communautaires (doubler, split, assurance) et la mise de chaque joueur du tour.

4) NFCRoulette



Le Client doit jouer à la roulette, miser des jetons, faire de la monnaie. De plus il doit pouvoir voir les règles de la roulette, le cout des jetons et les sommes totales des joueurs se trouvant sur la table.

Tous les joueurs doivent pouvoir voir la mise totale de chaque case du tour et la roulette.

III. Gestion de projet

1) Répartition des taches

Nous avons reparti les taches selon la facilité de chaque personne :

- Gregory Vesic : Partie client Android
- Marc Foucault : Partie Serveur et client Console
- Ludovic Robez : Partie Serveur et client Console

2) Travail effectuer

- NFCPaiement :

	Gregory	Marc	Ludovic
Client Android	100%	0%	0%
Client Console	0%	50%	50%
Serveur	0%	20%	80%

- NFCPoker, NFCBlackJack, NFCRoulette :

	Gregory	Marc	Ludovic
Client Android	0%	0%	2%
Serveur	0%	0%	2%

3) Les langages

- Java : Coté client Android, Coté serveur et client Console.
- SQL : Base de données.

4) Les Outils

- Git : Outil de versioning.
- MySql : Système de gestion de base de données.
- Maven : Outil pour la gestion et l'automatisation de production des projets logiciels.
- Jetty : Serveur HTTP et moteur de servlet.
- Android : Système d'exploitation mobile.

5) Les frameworks

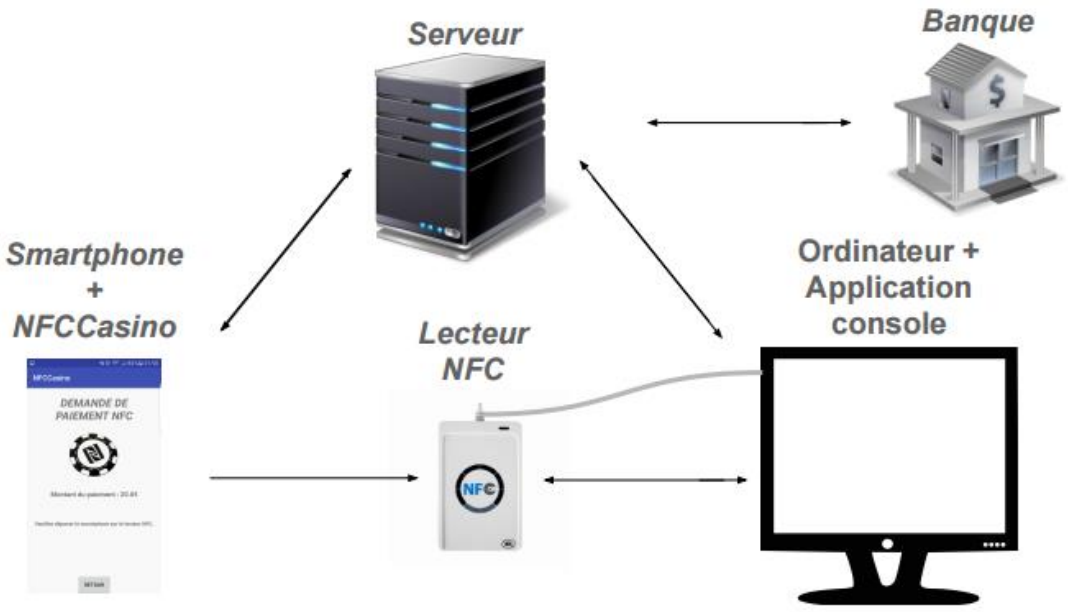
- EasyMock : Framework pour moque des objets qui reproduisent le comportement d'objets réels de manière contrôlée.
- PowerMock : Framework pour moque des objets qui reproduisent le comportement d'objets réels de manière contrôlée.
- Junit : Framework de test unitaire.

IV. Projet NFCPaiement

1) Schéma du projet

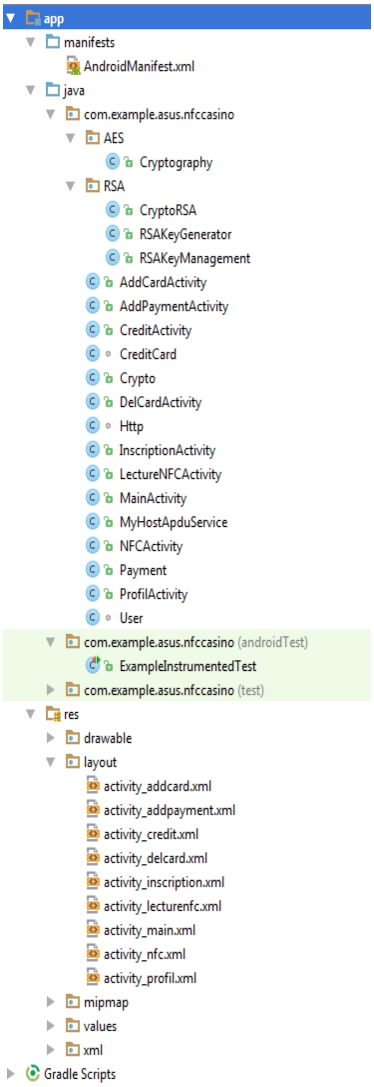
Le projet NFCPaiement est décomposé de 3 systèmes : le serveur, un client android et un client console. Chaque communication avec le serveur doit être crypter afin de garantir la sécurité des données transmit. Donc chaque client doit tout d'abord demander la clé publique du chiffrement asymétrique qu'il va déchiffrer grâce à la clé secrète du chiffrement symétrique. La clé publique du chiffrement asymétrique permettra de chiffrer et déchiffrer les données transmit par/au serveur.

La communication entre les 2 client se fait par NFC. Il s’agira du token de paiement qui est généré aléatoirement par le serveur.



2) Partie Client Android

1. Architecture Logiciel

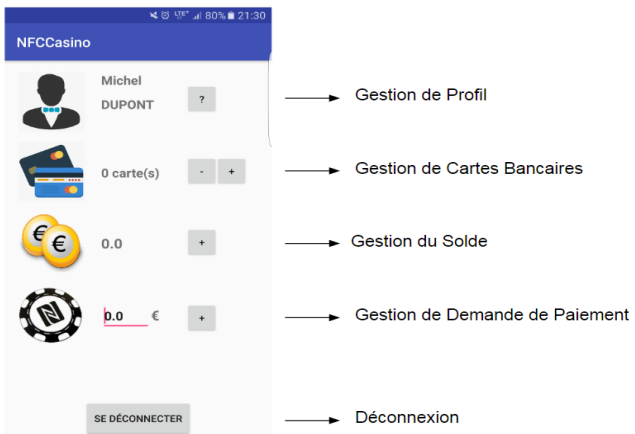


app

- manifests : permissions application applications
- java : classes (activity, autre)
- res : ressources
  - drawable : images
  - layout : disposition (design)
  - mipmap : images responsive
  - values : valeurs (string, colors, …)
  - xml : supplément (autres)

Gradle Scripts – Structure du projet

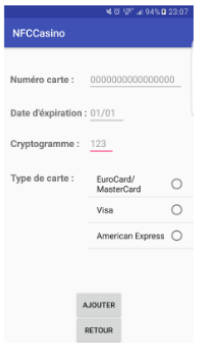
2. Aperçu général de l'Application



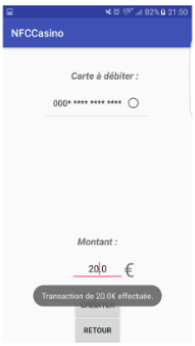
3. Fonctionnement de l'Application (après connexion)

1/ Profil utilisateur

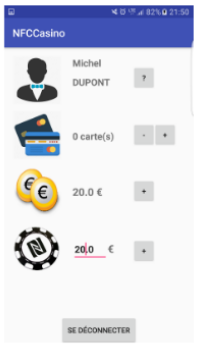
2/ Ajout carte bancaire



3/ Ajout au solde



4/ Montant du paiement



5/ Demande de paiement NFC

4. Fonctionnement de l'envoi des données vers lecteur NFC

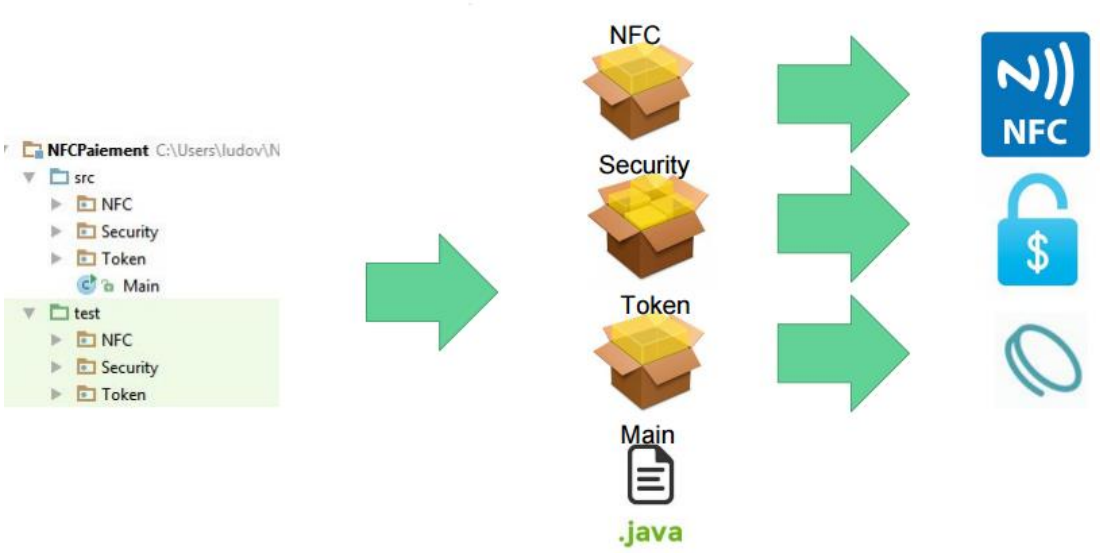
Pour pouvoir transmettre une donnée au lecteur NFC, il faut d'abord permettre à l'application d'utiliser le NFC du Smartphone.
Par la suite, il faut définir l'AID du lecteur. Dans notre cas, nous avons pris la valeur « F000000000124511 » correspondant à tous les lecteurs NFC de ce type.
Pour finir, il faut ensuite intégrer le fichier Java permettant la connexion ainsi que l'envoi des données en hébergeant le Service APDU. La donnée envoyée sera sous forme de tableau de byte, contenant dans les 2 derniers index les valeurs 0x90 et 0x00.

Voici le processus :

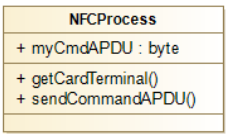
- Ajout de la classe MyHostApduService.java
- Ajout de la ressource apduservice.xml
- Déclaration Manifest
  - Permissions NFC
  - Service MyHostApduService dans le Manifest
    - Avec pour meta-data la ressource apduservice

3) Partie Client Console

1. *Architecture Logiciel*

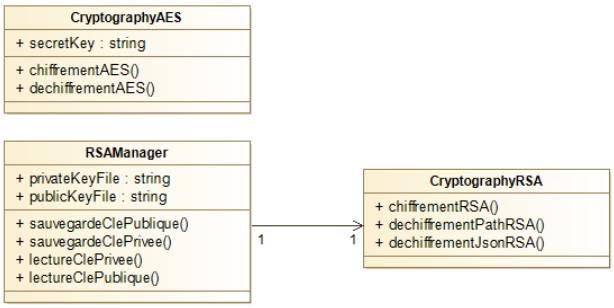


2. *Package NFC*



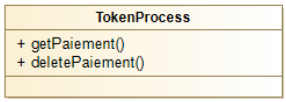
- NFCProcess : permet d'exécuter des commande APDU

3. *Package Security*



- CryptographyAES : permet de chiffrer et déchiffrer de manière symétrique
- CryptographyRSA : permet de chiffrer et déchiffrer de manière asymétrique

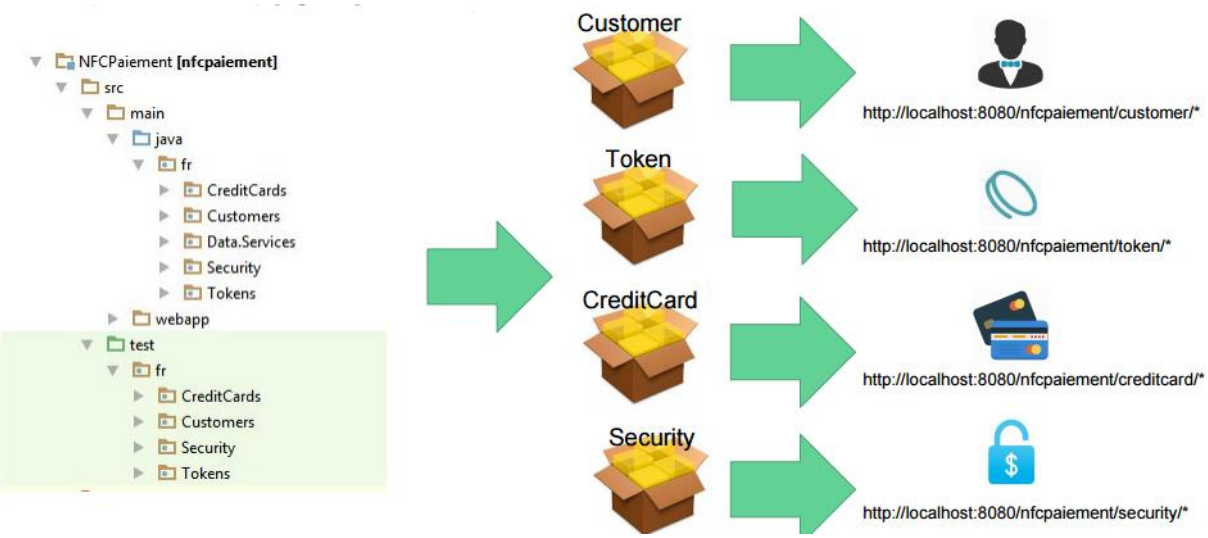
4. *Package Token*



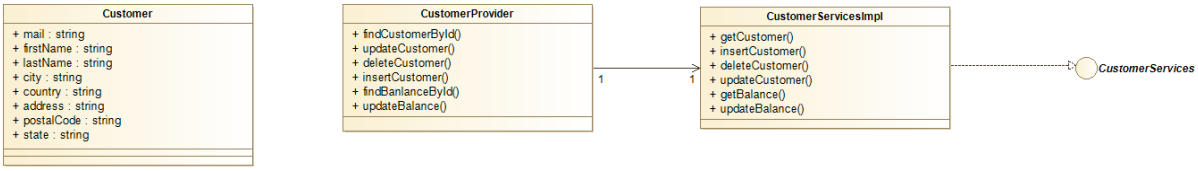
- TokenProcess : permet d'exécuter et de supprimer un paiement

4) Partie Serveur

1. *Architecture Logiciel*

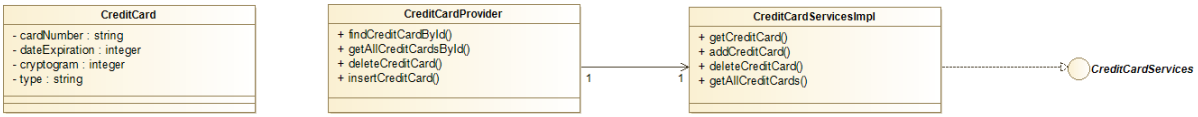


2. *Package Customer*



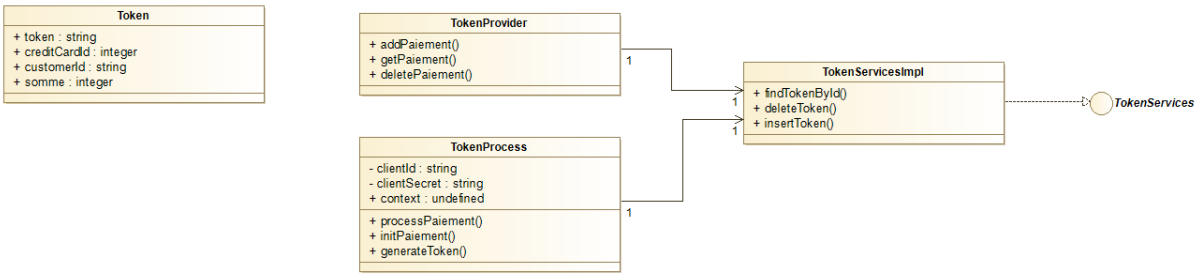
- Customer : Classe métier d'un client
- CustomerProvider : Ajout, modifie, supprime ou retourne les information d'un client dans la Base de données.
- CustomerServicesImpl : classe de gestion de client accessible avec l'url http://localhost:8080/nfcpaiement/customer/*

3. *Package CreditCard*



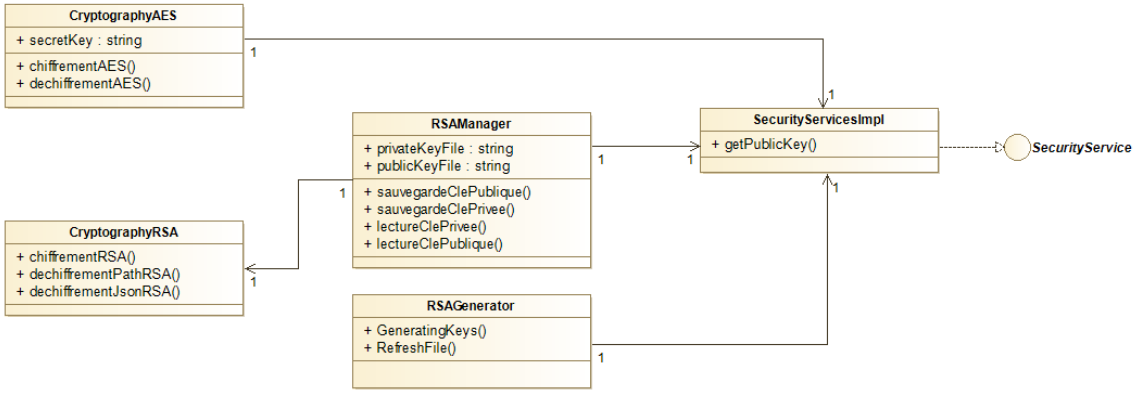
- CreditCard : Classe métier d'une Carte de crédit
- CreditCardProvider : Ajout, supprime ou retourne le ou les Carte(s) de crédit dans la Base de données.
- CreditCardServicesImpl : classe de gestion de Carte de crédit accessible avec l'url http://localhost:8080/nfcpaiement/creditcard /*

4. Package Token



- Token : Classe métier d'une Token.
- TokenProvider : Ajout, supprime ou retourne le token dans la Base de données.
- TokenProcess : génère un token, exécute et initialise un paiement
- TokenServicesImpl : classe de gestion de token accessible avec l'url http://localhost:8080/nfcpaiement/token /*

5. Package Security



- CryptographyAES : permet de chiffrer et déchiffrer de manière symétrique
- CryptographyRSA : permet de chiffrer et déchiffrer de manière asymétrique
- RSAManager : sauvegarde et lis les clés pour le chiffrement asymétrique
- RSAGenerator : permet de générer les clés pour le chiffrement asymétrique
- SecurityServicesImpl : retourne la clé publique du chiffrement asymétrique chiffré de manière symétrique accessible avec l'url http://localhost:8080/nfcpaiement/security /*

6. Test Unitaire

Pour les tests unitaires, les tests des différents packages se ressemble énormément voici 2 exemple :

- CreditCardProviderTest :

```java
@RunWith(PowerMockRunner.class)
@PrepareForTest({ CreditCardProviders.class, DataBaseAccess.class, DataBaseAccessImpl.class })
public class CreditCardProvidersTest {

    //region PROPS
    DataBaseAccess dbMock;
    //endregion

    //region CONFIG
    public Application configure() {

        return new ResourceConfig(CreditCardProviders.class);
    }

    @Before
    public void setUp() throws Exception {
        dbMock = createNiceMock(DataBaseAccess.class);

        PowerMock.mockStatic(DataBaseAccessImpl.class);
        expect(DataBaseAccessImpl.getDbConnection()).andReturn(dbMock);
    }

    @Test
    public void findCreditCardById() throws Exception {
        Map<String, String> creditCard = new HashMap<String, String>();
        creditCard.put("test", "test");
        expect(dbMock.findOneAsMap(anyString())).andReturn(creditCard);
        PowerMock.replayAll(dbMock);
        Assert.assertEquals("should return the map", creditCard, CreditCardProviders.findCreditCardById(1,"test"));
        PowerMock.verifyAll();
    }

    @Test
    public void getAllCreditCardsById() throws Exception {

        ArrayList<Map<String, String>> creditCards = new  ArrayList<Map<String, String>>(){{
            new HashMap<String, String>(){{
                put("test", "test");
            }};
            new HashMap<String, String>(){{
                put("test", "test");
            }};

        }};

        expect(dbMock.findAllAsMap(anyString())).andReturn(creditCards);
        PowerMock.replayAll(dbMock);
        Assert.assertEquals("should return the list", creditCards, CreditCardProviders.getAllCreditCardsById("test"));
        PowerMock.verifyAll();
    }
```

- CreditCardServicesImpl :

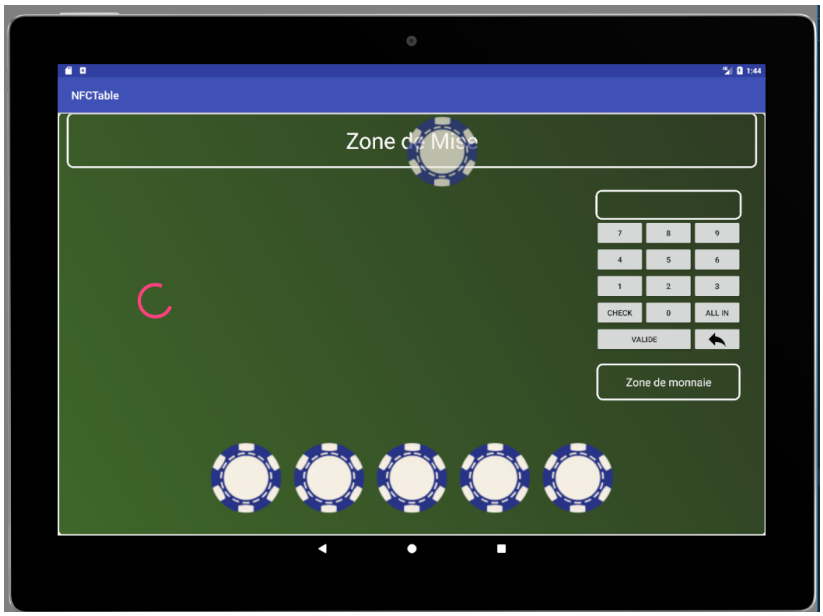
```java
@RunWith(PowerMockRunner.class)
@PrepareForTest({ CreditCardServicesImpl.class, CreditCardProviders.class})
public class CreditCardServicesImplTest {

    public Application configure() { return new ResourceConfig(CreditCardServicesImpl.class); }

    Client client;

    @Before
    public void setUp() throws Exception {
        client = ClientBuilder.newClient();
    }

    @Test
    public void getAllCreditCards() throws Exception {

        PowerMock.mockStatic(CreditCardProviders.class);
        expect(CreditCardProviders.getAllCreditCardsById(anyString())).andReturn(new ArrayList<Map<String, String>>());

        Response output = client.target("http://localhost:8080/nfcpaiement/creditcard/" + Cryptography.chiffrementRSA("1")).request().get();

        Assert.assertEquals("should return status 200", 200, output.getStatus());

        TestCase.assertNotNull("Should return list", output.getEntity());
    }

    @Test
    public void addCreditCard() throws Exception {

        PowerMock.mockStatic(CreditCardProviders.class);
        expect(CreditCardProviders.insertCreditCard((Map<String, String>) anyObject())).andReturn(true);
        JSONObject json = new JSONObject(new HashMap<String, String>());

        Response output = client.target("http://localhost:8080/nfcpaiement/creditcard").request().post(Entity.entity(MediaType.APPLICATION_JSON_TYPE,json.toString()));

        Assert.assertEquals("should return status 201", 201, output.getStatus());
    }

    @Test
    public void getCreditCard() throws Exception {

        PowerMock.mockStatic(CreditCardProviders.class);
        expect(CreditCardProviders.findCreditCardById(anyInt(),anyString())).andReturn(new HashMap<String, String>());

        Response output = client.target("http://localhost:8080/nfcpaiement/creditcard").request().get();

        Assert.assertEquals("should return status 200", 200, output.getStatus());

        TestCase.assertNotNull("Should return list", output.getEntity());
    }
```

V. Projet NFCRoulette, NFCPoker, NFCBlackJack

Ce projet est en bonus.

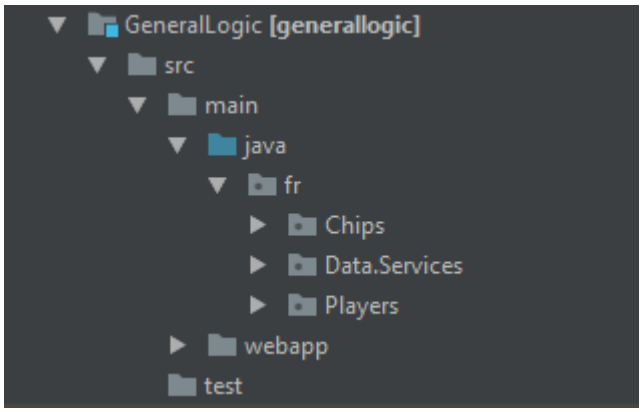
1) Partie Client Android

Nous n'avons pas eu assez de temps pour développer les fonctionnalités générales commun à tous les jeux mais voici a quoi ressemblerai l'IHM.



1) Partie Serveur

Voici aussi l'architecture des fonctionnalités général commun à tous les jeux :



Nous avons divisé le serveur en 3 packages :

- Chips : Gestion des jetons
- Player : Gestion des joueurs
- Data : Gestion de la Base de données

VI. Conclusion

Pour conclure, Le chiffrement a causé énormément de problème durant l'ensemble du projet. De plus avec un peu plus de travail l'ensemble du projet aurait pu aboutir. Comme tous le projet est axé sur le chiffrement, cela rend le projet non fonctionnel.