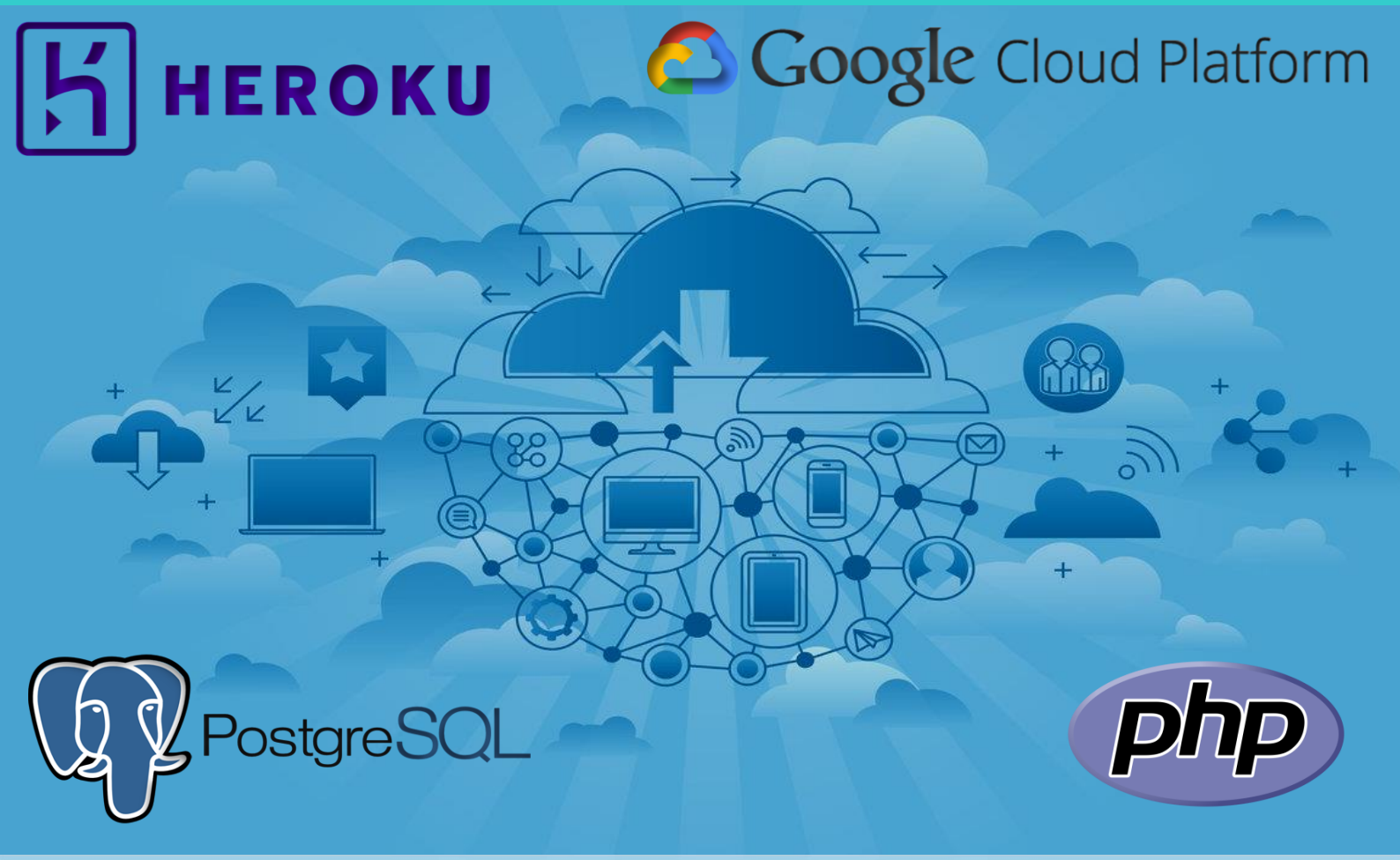
# DOCUMENTATION PROJET CLOUD




CAILLOT Alexandre
HEBERT Florian

---

**Licence Professionnelle : Développement d'application web**

**URL GitHub du Projet**

*https://github.com/ShiroeSama/cloud-programming-project*

# Base de données

La base de données est sous le système de gestion de base de données relationnelle et objet **PostgreSQL** qui est fournit et héberger par Heroku. Pour l'alimenter nous avons utilisé l'outil de gestion des bases de données **PgAdmin III.** Depuis cette interface nous avons créé des séquences afin de permettre l'ajout de données dans les tables avec des ID unique dans les tables ci-dessous. La base de données contient les tables concernant les comptes bancaires, et les demandes de crédit associé à ceux-ci. Il nous à sembler plus judicieux de créer une seule base de données afin de faciliter la jointure de données entre un compte et les demandes de crédits. De même que le choix de la base de données PostGreSQL fournit directement par Heroku.

- ***Données de connexion à la base PostgreSQL***

| | |
|---|---|
| Host | ec2-54-235-90-200.compute-1.amazonaws.com |
| Database | dclfn1iueip14h |
| User | wsccixbtuidyta |
| Port | 5432 |
| Password | 1f0e386b68be62083503cae79a231e7888ca2a8870932f7e58a9cf65fd10ad84 |
| URI | postgres://wsccixbtuidyta:1f0e386b68be62083503cae79a231e7888ca2a8870932f7e58a9cf65fd10ad84@ec2-54-235-90-200.compute-1.amazonaws.com:5432/dclfn1iueip14h |
| Heroku CLI | heroku pg:psql postgresql-cubic-42287 --app powerful-woodland-66424 |

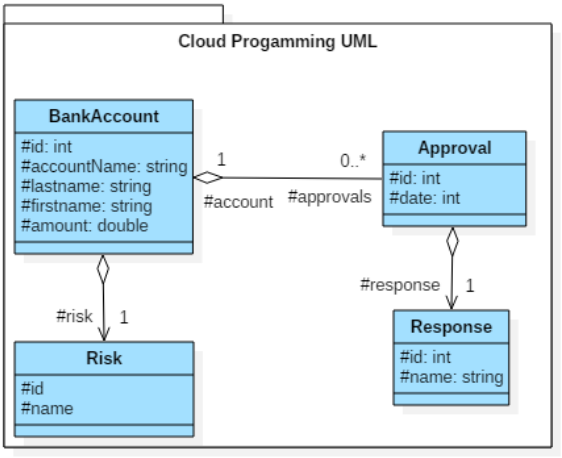



***Figure 1*** - *Diagramme UML de nos données*

- ***Les données de l'application Cloud se stocke à travers 4 tables ayant chacune un rôle spécifique.***

BankAccount : son rôle est de stocker les différents compte banquaire du service web.
Risk : Définie le type de risque d'un compte bancaire ( HIGH , LOW )
Approval : son rôle est de stocker les différentes demande de crédit faite par un client.
Response : Définie la réponse à une demande de crédit ( ACCEPTED , REFUSED )

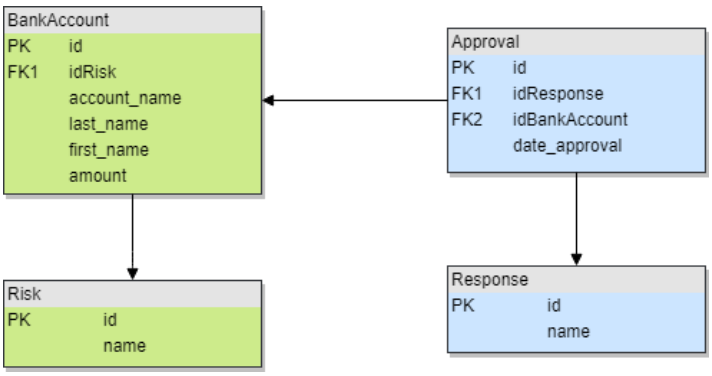



***Figure 2*** - *Schéma relationnel de nos données*

- ***Commandes PostGreSQL d'ajout de nos tables :***

```
CREATE TABLE public."Approval"
(
    id integer NOT NULL DEFAULT nextval('seq_id_approval'::regclass),
    date_approval timestamp with time zone NOT NULL,
    "idResponse" integer NOT NULL,
    CONSTRAINT "Approval_pkey" PRIMARY KEY (id),
    CONSTRAINT "fk_Response" FOREIGN KEY ("idResponse")
        REFERENCES public."Response" (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

```
CREATE TABLE public."BankAccount"
(
    id integer NOT NULL DEFAULT nextval('seq_id_bankaccount'::regclass),
    account_name text COLLATE pg_catalog."default" NOT NULL,
    last_name text COLLATE pg_catalog."default",
    first_name text COLLATE pg_catalog."default",
    amount double precision,
    "idRisk" integer NOT NULL,
    CONSTRAINT "BankAccount_pkey" PRIMARY KEY (id),
    CONSTRAINT "fk_Risk" FOREIGN KEY ("idRisk")
        REFERENCES public."Risk" (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

```
CREATE TABLE public."Risk"
(
    id integer NOT NULL DEFAULT nextval('seq_id_risk'::regclass),
    name text COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Risk_pkey" PRIMARY KEY (id)
)
```

```
CREATE TABLE public."Response"
(
    id integer NOT NULL DEFAULT nextval('seq_id_response'::regclass),
    name text COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Response_pkey" PRIMARY KEY (id)
)
```

L'objectif est de faire communiquer les services suivant avec la base de données :

- ***AccManager***

Permet d'ajouter, de supprimer et de lister des comptes.

- ***AppManager***

Permet d'ajouter, de supprimer et de lister les demandes de crédit. Ce service appel implicitement les services check_account et loanApproval.

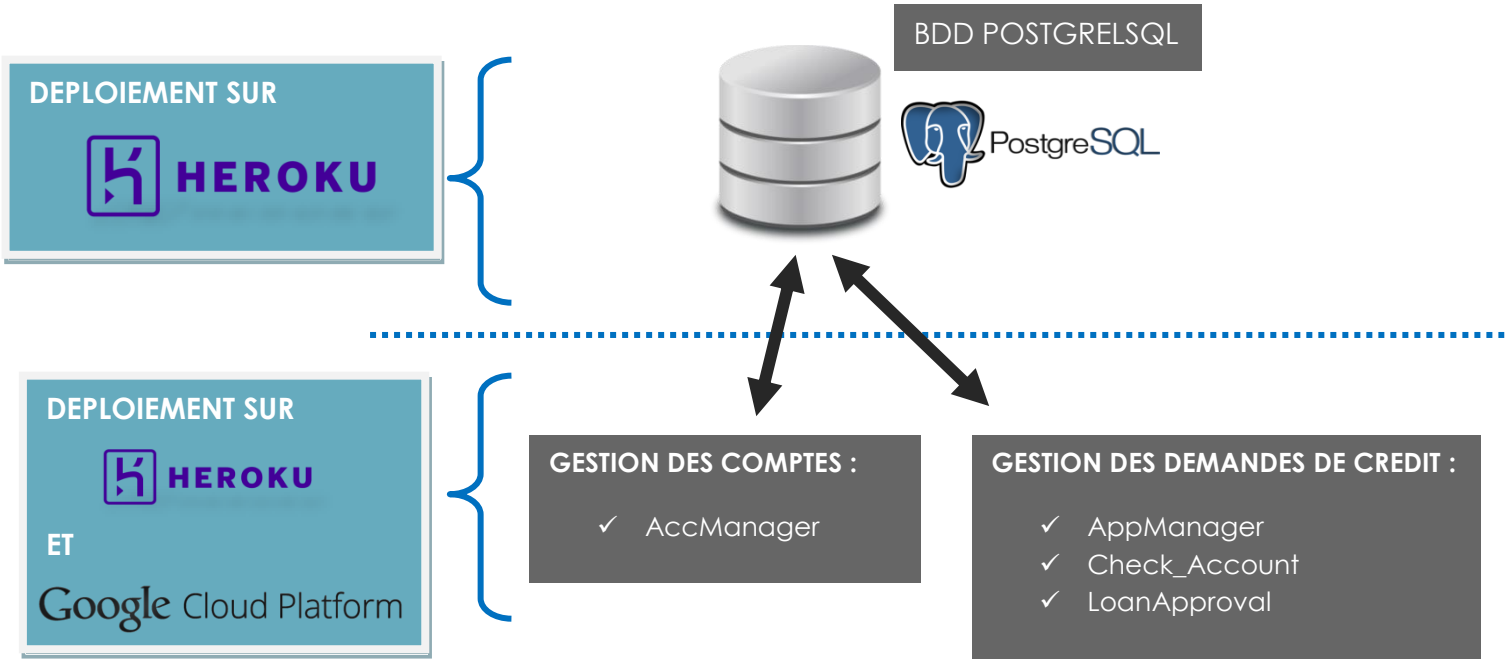
- ***Check_Account***

Service qui vérifie le RISK pour un utilisateur donné.

- ***LoanApproval***

Service de demande de crédit.

***Les services sont partagés en deux catégories, l'une pour la gestion des comptes, l'autre pour la gestion des demandes de crédit.***




**Suite à de nombreux soucis vis-à-vis du déploiement des services sur GAE, nous avons déployé l'intégralité des services de l'application sur la plateforme Heroku.**

***La gestion des comptes est déployée sur Heroku, la gestion des demandes de crédit est elle aussi déployée sur Heroku mais aurais du être déployée sur Google App Engine.***

**Name**
*arcane-lake-71736*
**Heroku Git URL**
*https://git.heroku.com/arcane-lake-71736.git*
**Domain**
*https://arcane-lake-71736.herokuapp.com/*

HEROKU

# Définition des rôles des Web services


- **Account List :**
  **Method :** GET
  **Response Code :** 200 (OK)
  **Url :** {{hostname}}/api/accounts

- **Add Account :**
  **Method :** POST
  **Response Code :** 201 (Created) **OR** 500 (Internal Server Error)
  **Url :** {{hostname}}/api/accounts

- **Delete Account :**
  **Method :** DELETE
  **Response Code :** 204 (No Content) **OR** 404 (Not Found)
  **Url :** {{hostname}}/api/accounts

---



- **Approval List :**
  **Method :** GET
  **Response Code :** 200 (OK)
  **Url :** {{hostname}}/api/approvals

- **Delete Approval :**
  **Method :** DELETE
  **Response Code :** 204 (No Content) **OR** 404 (Not Found)
  **Url :** {{hostname}}/api/approvals

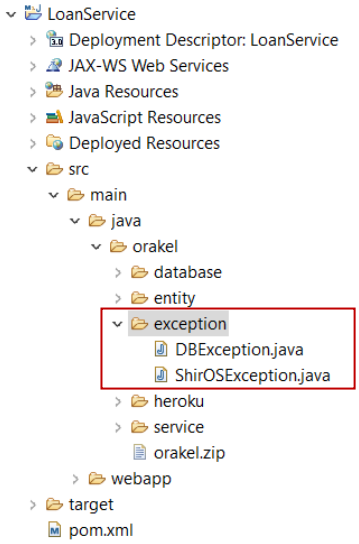
- **Risk Account :**
  **Method :** GET
  **Response Code :** 200 (OK) **OR** 404 (Not Found)
  **Url :** {{hostname}}/api/risk/:id

- **Loan Request :**
  **Method :** POST
  **Response Code :** 201 (Created) **OR** 500 (Internal Server Error)
  **Url :** {{hostname}}/api/loan

---

# Gestion des erreurs dans les services

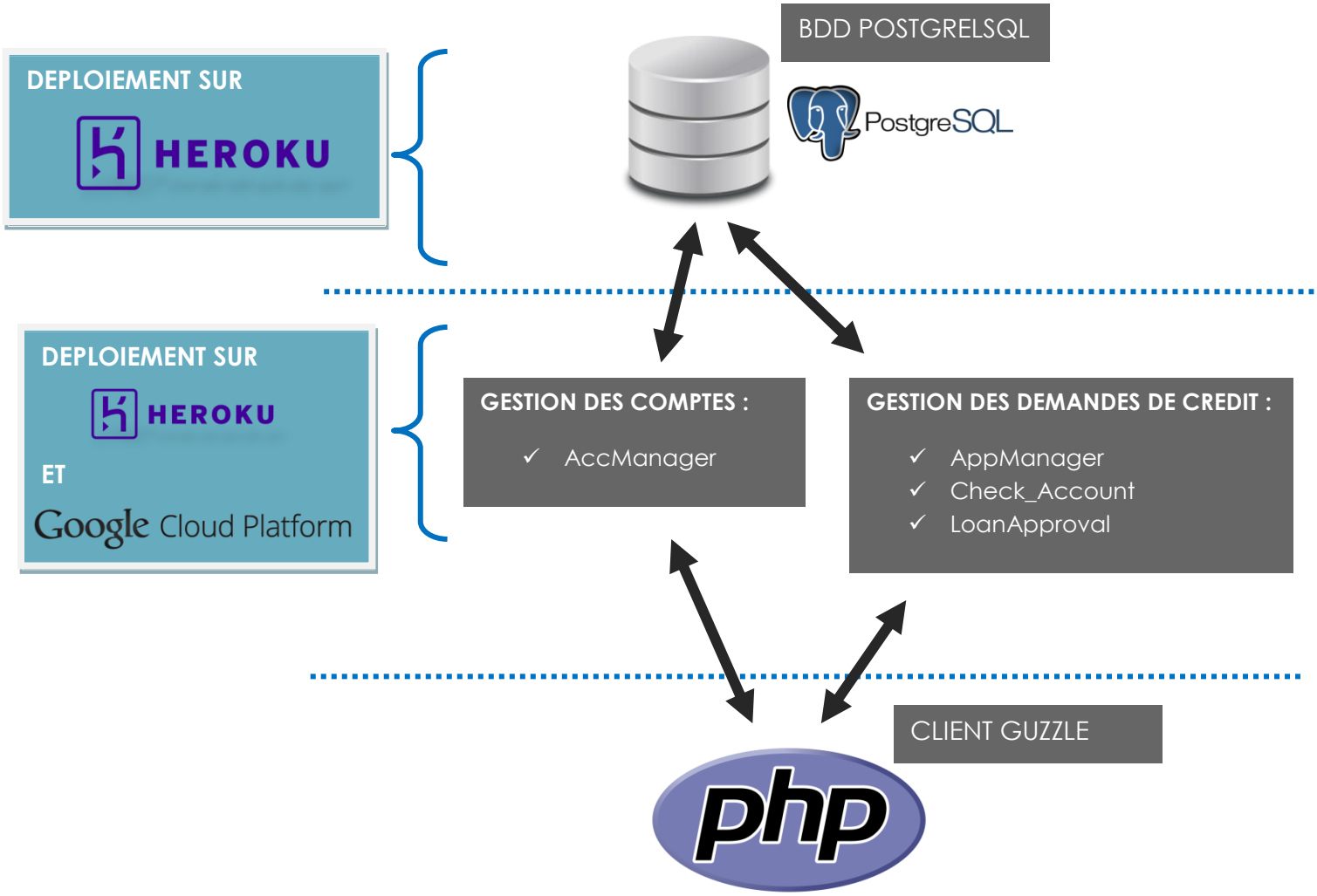
La gestion des erreurs permettent un affichage en JSON de celles-ci depuis la page client. Ceci permet de savoir le type d'erreur, son message ainsi que le code de retour. Cette gestion permet également mais également de visionner la trace complète des erreurs.


C'est le dossier exception et ses classe DBException.java et ShirOSException.java qui permette cette gestion des erreurs en retournant ces erreurs en type string sous le format de données textuelles JSON.

## ▪ Client Guzzle

Le client Guzzle est codé dans le langage PHP dans sa version 7.1. Il client Guzzle fait appel aux différents services web pour interagir ou afficher les résultats retournés.




## ▪ Répartirions des tâches :

- **Florian Hebert :**
  - ✓ Réalisation des service en rapport avec la gestion des demandes de crédit (Approval List,Delete Approval, Loan Request)
  - ✓ Réalisation de la base de données
  - ✓ ***Déploiement vers la plateforme GAE > NON FONCTIONNEL***
  - ✓ Réalisation de la documentation
- **Alexandre Caillot :**
  - ✓ Réalisation des service en rapport avec la gestion des demandes de crédit ( Account List, Add Account, Risk Account, Delete Account )
  - ✓ Réalisation du client Guzzle
  - ✓ Déploiement vers la plateforme Heroku
  - ✓ Debug du projet