



ÉCOLE
D'INGÉNIEURS
PARIS-LA DÉFENSE



PROJET CLASSICMODELS

Développement d'application pour le Cloud

Cours	Développement d'applications Cloud
Professeur	TRAVERS Nicolas
Majeure - Année	IBO - 5A
Membres de l'équipe	LEVRET Alexandre TAZI Maxime RAMOS Miguel

TABLE DES MATIÈRES

<u>RESUME DU PROJET</u>	<u>2</u>
<u>ELABORATION DU PROJET</u>	<u>3</u>
SPECIFICATION DES BESOINS	3
DENORMALISATION	4
DEVELOPPEMENT DE L'APPLICATION	9
LES VUES ET LES REQUETES ASSOCIEES	11
<u>BONUS : DEPLOIEMENT DE L'APPLICATION DANS AWS</u>	<u>16</u>
<u>CONCLUSION</u>	<u>17</u>

RESUME DU PROJET

Notre projet consistait à créer une application, sur le langage de notre choix, sur une base de données NoSQL étudiée pendant les cours, à savoir MongoDB.

Tout d'abord, une étape de dénormalisation des données SQL a été effectuée. En effet, nous sommes partis d'une base de données relationnelle qui représente les données d'un magasin de voitures miniaturisées, et que nous avons transformé en JSON pour ensuite l'intégrer de manière dénormalisé dans MongoDB sous différentes collections.

Nous avons donc créé une application web codée en full JS avec la stack MERN (Mongoose, Express, ReactJS, NodeJS). Cette application sert d'interface de visualisation de données pour les requêtes qui seront décrites dans la suite du rapport.

L'application comprend 3 vues :

- Une vue utilisateur standard : il s'agit de 4 requêtes les plus demandées sur la base de données sans intervention de l'utilisateur.
- Une vue Analyste/Décisionnaire : il s'agit de 2 requêtes complexes paramétrables par l'utilisateur.
- Une vue Administrateur : Il fournit des statistiques sur les données.

SPECIFICATION DES BESOINS

La base de données SQL a été choisie à partir de ce lien :

<https://relational.fit.cvut.cz/dataset/ClassicModels>

Elle contient les données de magasins de voitures miniaturisées commercialisées internationalement dans les années 2003, 2004 et 2005

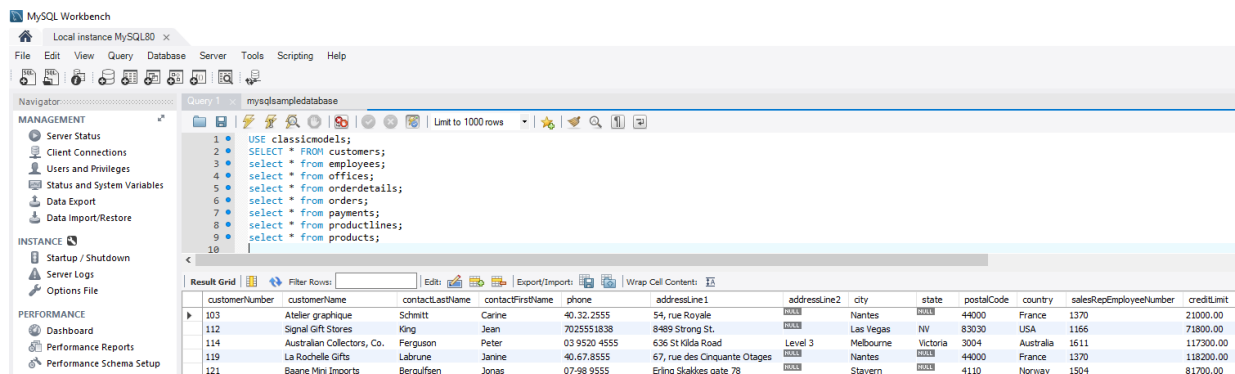
Nous avons défini les besoins suivants pour l'interrogation de cette base de données :

- Pour les 4 interrogations les plus demandées nous avons :
 - La liste des clients : pour avoir une vue globale sur les différentes informations d'un client (noms, coordonnées, montant des achats)
 - Les 10 voitures les plus commandées : pour savoir la tendance des produits les plus achetés et ainsi mieux gérer les stocks.
 - Les 5 clients les plus fidèles : pour savoir avec quel client on génère le plus de revenus
 - Chiffres d'affaires mensuel pour l'année 2004 : pour comprendre le taux des ventes au cours de l'année
- Pour les 2 interrogations complexes :
 - Les profits mensuels par année et par pays des magasins : pour connaître les régions du monde où on a généré le plus de profit et ainsi faire de la publicité ciblée.
 - Les profits mensuels moyens par année et par pays des clients :

DENORMALISATION

Les étapes de la dénormalisation ont été les suivantes :

- 1- Connecter la base de données sous MySQL vers Python à l'aide de MySQLConnector



```
mydb = mysql.connector.connect(host='127.0.0.1',user='root', password='root', database='classicmodels', auth_plugin='mysql_native_password')
```

- 2- Exécuter les requêtes en python et les transformer en JSON

```
tables = ["customers", "employees", "offices", "orderdetails", "orders", "payments", "productlines", "products"]
print(len(tables))

for i in range(len(tables)):

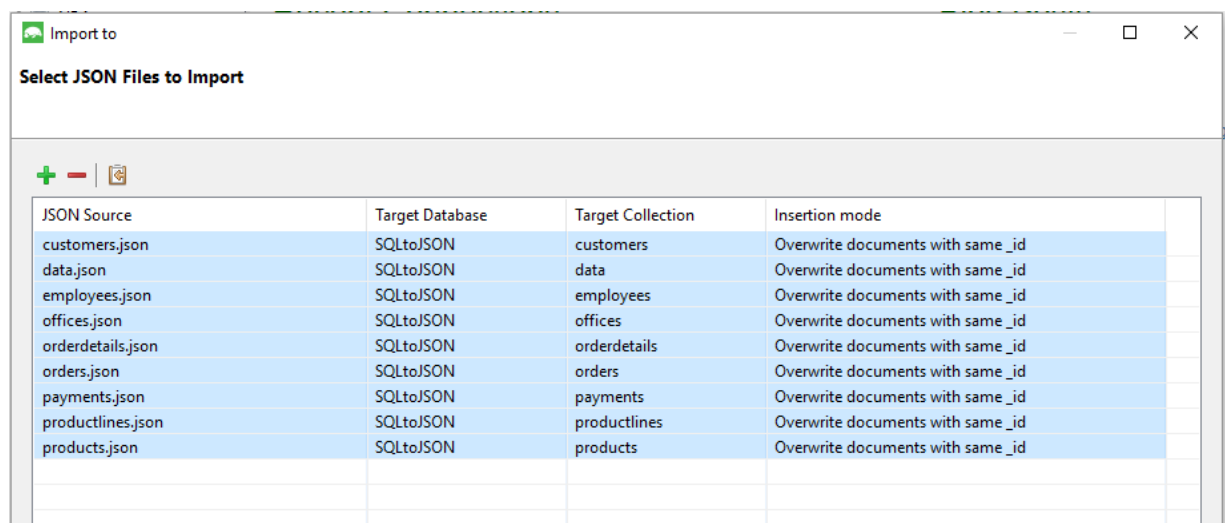
    mycursor.execute("SELECT * FROM " + tables[i])
    row_headers=[x[0] for x in mycursor.description] #this will extract row headers
    myresult = mycursor.fetchall()

    json_data=[]
    for result in myresult:
        json_data.append(dict(zip(row_headers,result)))
    print(json_data)
    print(json.dumps(json_data, default=decimal_datetime_default))

    with open(tables[i] + '.json', 'w') as outfile:
        json.dump(json_data, outfile, default=decimal_datetime_default)
```

customers.json	28/11/2018 19:39	Fichier JSON	42 Ko
data.json	27/11/2018 23:13	Fichier JSON	42 Ko
employees.json	28/11/2018 19:39	Fichier JSON	5 Ko
offices.json	28/11/2018 19:39	Fichier JSON	2 Ko
orderdetails.json	28/11/2018 19:39	Fichier JSON	342 Ko
orders.json	28/11/2018 19:39	Fichier JSON	61 Ko
payments.json	28/11/2018 19:39	Fichier JSON	27 Ko
productlines.json	28/11/2018 19:39	Fichier JSON	4 Ko
products.json	28/11/2018 19:39	Fichier JSON	46 Ko
PC SQLtoJSONconverter.py	28/11/2018 19:39	JetBrains PyChar...	2 Ko

3- Importer les JSON dans MongoDB sous différentes collections



The screenshot shows the 'Import to' window in MongoDB, specifically the 'Select JSON Files to Import' step. It displays a table with the following columns: JSON Source, Target Database, Target Collection, and Insertion mode. The data is as follows:

JSON Source	Target Database	Target Collection	Insertion mode
customers.json	SQLtoJSON	customers	Overwrite documents with same _id
data.json	SQLtoJSON	data	Overwrite documents with same _id
employees.json	SQLtoJSON	employees	Overwrite documents with same _id
offices.json	SQLtoJSON	offices	Overwrite documents with same _id
orderdetails.json	SQLtoJSON	orderdetails	Overwrite documents with same _id
orders.json	SQLtoJSON	orders	Overwrite documents with same _id
payments.json	SQLtoJSON	payments	Overwrite documents with same _id
productlines.json	SQLtoJSON	productlines	Overwrite documents with same _id
products.json	SQLtoJSON	products	Overwrite documents with same _id

4- Créer les collections dénormalisées avec des requêtes en utilisant des \$Lookup :

Pour la 1^{ère} requête, la liste des clients, il n'y avait pas de dénormalisation à effectuer, il suffisait d'afficher le contenu de la collection customers.

```

1 db.getCollection("customers").find({})
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

```

Pour la 2^{ème} requête, les 10 voitures les plus commandées, on devait dénormaliser les tables orders, orders_details et products pour créer une collection nommée orders.

```

1 // 2nd query: Denormalize orderdetails + products
2 db.orderdetails.aggregate([
3   {
4     $lookup:
5     {
6       from: "products",
7       localField: "productCode",
8       foreignField: "productCode",
9       as: "product"
10    }
11  },
12  {
13    $unwind:
14    {
15      path: "$product",
16      preserveNullAndEmptyArrays: true
17    }
18  }
19 ])
20
21 // Denormalize orders + ordersdetails + products
22 db.orders.aggregate([
23   {
24     $lookup:
25     {
26       from: "orderdetails_products_denormalized",
27       localField: "orderNumber",
28       foreignField: "orderNumber",
29       as: "orderDetails"
30     }
31   }
32 ])
33

```

```

1 db.getCollection("orders").find({})
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40

```

Pour la 3^{ème} requête et 6^{ème} requête, les 5 clients les plus fidèles et les profits mensuels par année et par pays des clients, on devait dénormaliser les tables customers et payments pour créer une collection nommée customerspayments.

```
87 // 3rd query : Denormalize customers / employees / offices / payments / orders_denormalized
88 db.customers.aggregate([
89   {
90     $lookup:
91     {
92       from: "payments",
93       localField: "customerNumber",
94       foreignField: "customerNumber",
95       as: "payments"
96     }
97   }
98 ])
```

```
1 db.getCollection("customerspayments").find({})
2
Find
50 Documents 1 to 50
1 {
2   "_id" : ObjectId("5bfc1c571bebbf50444deab6"),
3   "customerNumber" : 103,
4   "customerName" : "Atelier graphique",
5   "contactLastName" : "Schmitt",
6   "contactFirstName" : "Carine ",
7   "phone" : "40.32.2555",
8   "addressLine1" : "54, rue Royale",
9   "addressLine2" : null,
10  "city" : "Nantes",
11  "state" : null,
12  "postalCode" : "44000",
13  "country" : "France",
14  "salesRepEmployeeNumber" : 1370,
15  "creditLimit" : 21000.0,
16  "payments" : [
17    {
18      "_id" : ObjectId("5bfee26e1bebbf50444dec57"),
19      "customerNumber" : 103,
20      "checkNumber" : "HQ336336",
21      "paymentDate" : "2004-10-19",
22      "amount" : 6066.78
23    },
24    {
25      "_id" : ObjectId("5bfee26e1bebbf50444dec59"),
26      "customerNumber" : 103,
27      "checkNumber" : "JM555205",
28      "paymentDate" : "2003-06-05",
29      "amount" : 14571.44
30    },
31    {
32      "_id" : ObjectId("5bfee26e1bebbf50444dec5b"),
33      "customerNumber" : 103,
34      "checkNumber" : "OM314933",
35      "paymentDate" : "2004-12-18",
36      "amount" : 1676.14
37    }
38  ]
39 }
40 {
41   "_id" : ObjectId("5bfc1c571bebbf50444deab7"),
```


Pour la 4^{ème} requête, chiffres d'affaires mensuel pour l'année 2004, il n'y avait pas de dénormalisation à effectuer, il suffisait d'exploiter les données de la collection payments.

```
1 db.getCollection("payments").find({})
2
Find 50 Documents 1 to 50
1 {
2   "_id" : ObjectId("5bfee26e1bebbf50444dec57"),
3   "customerNumber" : 103,
4   "checkNumber" : "HQ336336",
5   "paymentDate" : "2004-10-19",
6   "amount" : 6066.78
7 }
8 {
9   "_id" : ObjectId("5bfee26e1bebbf50444dec59"),
10  "customerNumber" : 103,
11  "checkNumber" : "JM555205",
12  "paymentDate" : "2003-06-05",
13  "amount" : 14571.44
14 }
15 {
16  "_id" : ObjectId("5bfee26e1bebbf50444dec5b"),
17  "customerNumber" : 103,
18  "checkNumber" : "QM314933",
19  "paymentDate" : "2004-12-18",
20  "amount" : 1676.14
21 }
22 {
23  "_id" : ObjectId("5bfee26e1bebbf50444dec5c"),
```

Pour la 5^{ème} requête, les profits mensuels par année et par pays des magasins, on devait dénormaliser les tables customers, employees, offices et payments pour créer une collection customersofficespayments.

```
101 // 5th query : Denormalize customers / employees / offices / payments
102 db.customers.aggregate([
103   {
104     $lookup:
105     {
106       from: "employees",
107       localField: "salesRepEmployeeNumber",
108       foreignField: "employeeNumber",
109       as: "employee"
110     },
111   },
112   {
113     $unwind:
114     {
115       path: "$employee",
116       preserveNullAndEmptyArrays: true //dans le cas où le customer n'as pas d'employee, on garde quand même le customer
117     },
118   },
119   {
120     $lookup:
121     {
122       from: "offices",
123       localField: "employee.officeCode",
124       foreignField: "officeCode",
125       as: "employee.office"
126     },
127   },
128   {
129     $lookup:
130     {
131       from: "payments",
132       localField: "customerNumber",
133       foreignField: "customerNumber",
134       as: "payments"
135     },
136   },
137 ])
```

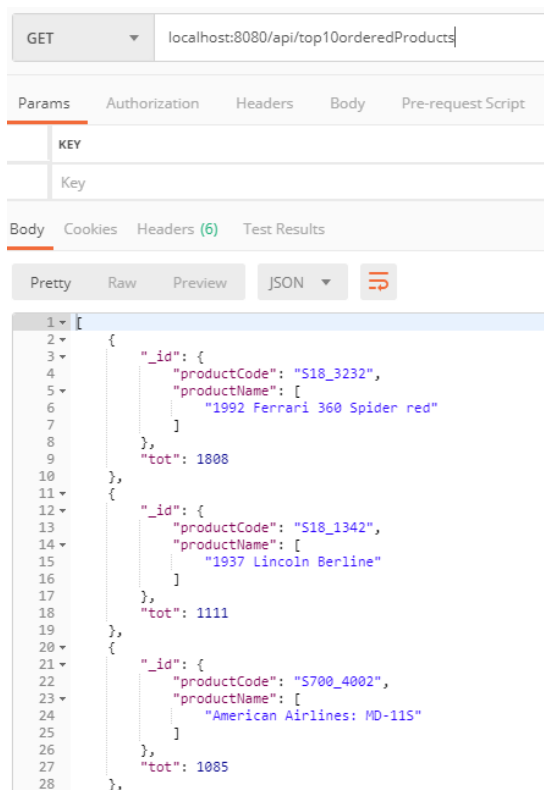
```
1 db.getCollection("customersofficespayments").find({})
2
Find 50 Documents 1 to 50
1 {
2   "_id" : ObjectId("5bfee571bebbf50444deab6"),
3   "customerNumber" : 103,
4   "customerName" : "Atelier graphique",
5   "contactLastName" : "Schmitt",
6   "contactFirstName" : "Carine",
7   "phone" : "40.32.2555",
8   "addressline1" : "54 rue Royale",
9   "addressline2" : null,
10  "city" : "Nantes",
11  "state" : null,
12  "postalCode" : "44000",
13  "country" : "France",
14  "salesRepEmployeeNumber" : 1370,
15  "creditLimit" : 21000.0,
16  "employee" : {
17    "_id" : ObjectId("5bfee26d1bebbf50444deb61"),
18    "employeeNumber" : 1370,
19    "lastName" : "Hernandez",
20    "firstName" : "Gerard",
21    "extension" : "x2028",
22    "email" : "ghernandez@classicmodelcars.com",
23    "officeCode" : "4",
24    "reportTo" : 1102,
25    "jobTitle" : "Sales Rep",
26    "office" : {
27      "_id" : ObjectId("5bfee26d1bebbf50444deb61"),
28      "officeCode" : "4",
29      "city" : "Paris",
30      "phone" : "+33 14 723 4404",
31      "addressline1" : "43 Rue Joffroy D'abbans",
32      "addressline2" : null,
33      "state" : null,
34      "country" : "France",
35      "postalCode" : "75017",
36      "territory" : "EHEA"
37    }
38  },
39  "payments" : [
40    {
41      "_id" : ObjectId("5bfee26e1bebbf50444dec57"),
42      "customerNumber" : 103,
43      "checkNumber" : "HQ336336",
44      "paymentDate" : "2004-10-19",
45      "amount" : 6066.78
46    },
47  ]
48 }
```

DEVELOPPEMENT DE L'APPLICATION

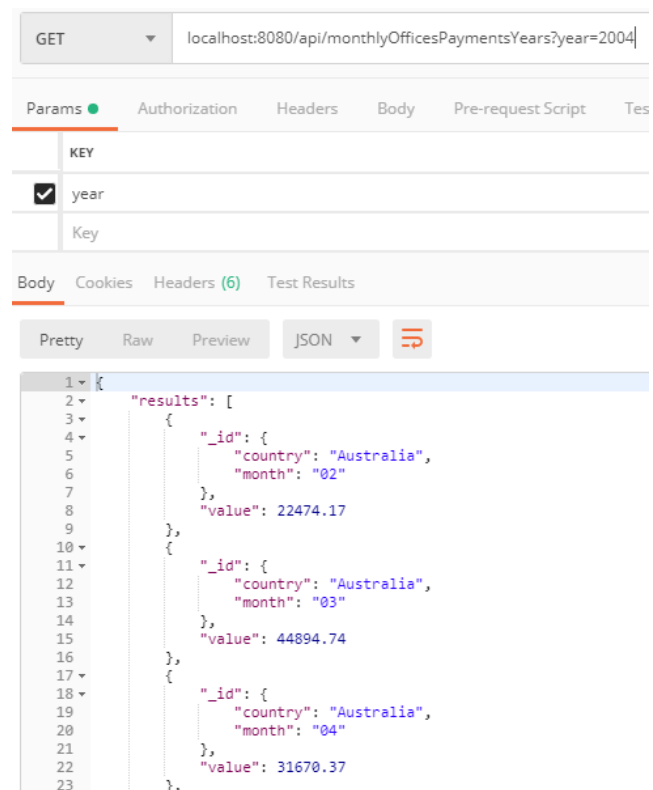
Au niveau du back-end :

Pour créer le serveur web, où l'application pourrait se lancer, nous avons utilisé le module Express sous NodeJS. Avec ce module nous avons pu créer une API que le client pourrait requêter pour avoir les données voulues à partir de différents endpoints (routing).

Voici quelques exemples des résultats des requêtes http sous Postman :



```
1 [
2   {
3     "_id": {
4       "productCode": "S18_3232",
5       "productName": [
6         "1992 Ferrari 360 Spider red"
7       ]
8     },
9     "tot": 1808
10  },
11  {
12    "_id": {
13      "productCode": "S18_1342",
14      "productName": [
15        "1937 Lincoln Berline"
16      ]
17    },
18    "tot": 1111
19  },
20  {
21    "_id": {
22      "productCode": "S700_4002",
23      "productName": [
24        "American Airlines: MD-11S"
25      ]
26    },
27    "tot": 1085
28  },
29 ]
```



```
1 {
2   "results": [
3     {
4       "_id": {
5         "country": "Australia",
6         "month": "02"
7       },
8       "value": 22474.17
9     },
10    {
11      "_id": {
12        "country": "Australia",
13        "month": "03"
14      },
15      "value": 44894.74
16    },
17    {
18      "_id": {
19        "country": "Australia",
20        "month": "04"
21      },
22      "value": 31670.37
23    }
24  ]
25 }
```

Certains endpoint contenaient des paramètres.

Le principe était que l'on puisse récupérer les paramètres par l'utilisateur et de traiter ces paramètres avec le module mongoose qui permet d'exécuter des commandes de Mongo DB à partir du code JS.

```
9   app.get('/api/customers', (req, res, next) => {
10     var query = "{}"
11     var object_query = JSON.parse(query);
12     customers.find(object_query)
13       .exec()
14       .then((query_result) => res.json(query_result))
15       .catch((err) => next(err));
16   });
17
18   app.get('/api/top10orderedProducts', (req, res, next) => {
19     opUnwind = { $unwind: "$orderDetails" };
20     opGroup = { $group: { _id: { productCode: "$orderDetails.productCode", productName:
21     opSort = { $sort: { tot: -1 } };
22     opLimit = { $limit: 10 }
23     orders.aggregate([opUnwind, opGroup, opSort, opLimit])
24       .exec()
25       .then((query_result) => res.json(query_result))
26       .catch((err) => next(err));
27   });
28 }
```

Au niveau client :

Pour afficher les résultats des requêtes sous une forme de visualisation de données, nous avons utilisé le framework ReactJS avec les librairies react-table et react-charts.

ReactJS est basé sur des états de données qui changent en fonction d'évènements. Nous avons donc créé des états de données correspondant à chaque requête, et au lancement de l'application (évènement du chargement de DOM) les requêtes sont exécutées.

Les requêtes sont exécutées par l'utilisation de fonctions de « fetch » qui font des requêtes HTTP sur les endpoints que nous avons créé au niveau du serveur.

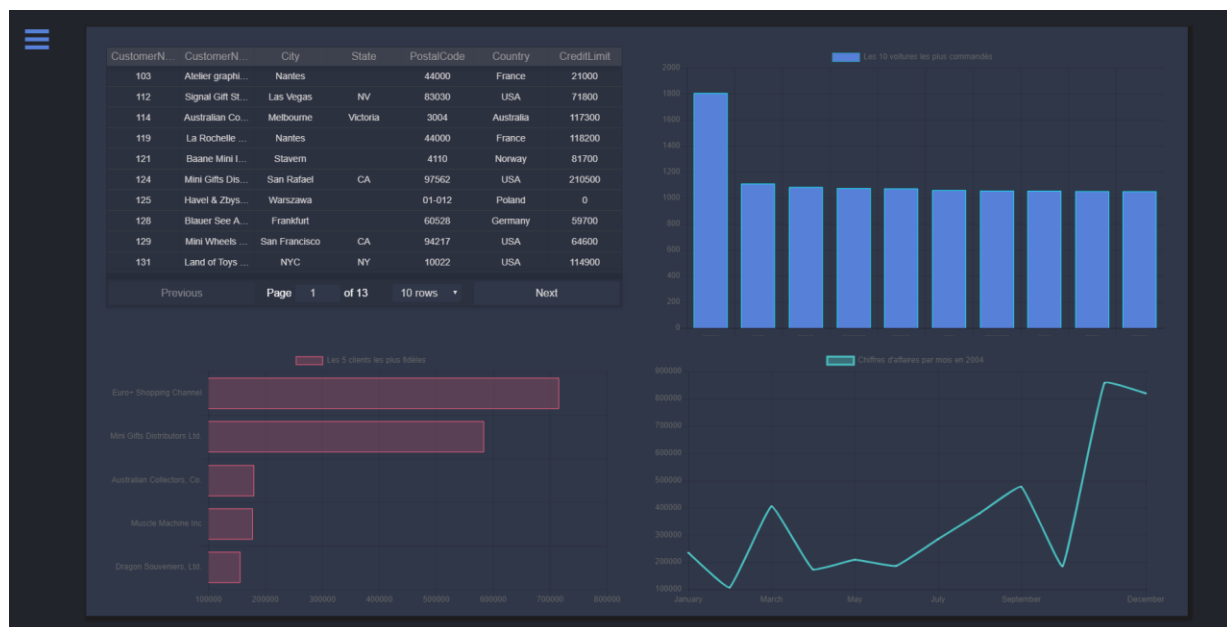
Voici par exemple la fonction qui fetch les données de l'endpoint customers et qui met à jour l'état des données.

```

fetchCustomers() {
  fetch("/api/customers")
    .then(response => response.json())
    .then(parsedJSON => {
      this.setState({
        jsondata: parsedJSON
      })
    })
  }
)
.catch(error => console.log("parsing failed", error))
}

```


LES VUES ET LES REQUETES ASSOCIES





La première vue, montre les 4 premières requêtes avec find et aggregates.

La liste des clients :

```
1 // 1 - La liste des clients
2 db.customers.find({})
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```


Find 


 50 Documents 1 to 50 

```
1 {
2   "_id" : ObjectId("5bfc1c571bebbf50444deab6"),
3   "customerNumber" : 103,
4   "customerName" : "Atelier graphique",
5   "contactLastName" : "Schmitt",
6   "contactFirstName" : "Carine ",
7   "phone" : "40.32.2555",
8   "addressLine1" : "54, rue Royale",
9   "addressLine2" : null,
10  "city" : "Nantes",
11  "state" : null,
12  "postalCode" : "44000",
13  "country" : "France",
14  "salesRepEmployeeNumber" : 1370,
15  "creditLimit" : 21000.0
16 }
```

Les 10 voitures les plus commandées :

```
4 // 2 - Les 10 voitures les plus commandées
5 opUnwind = {$unwind : "$orderDetails"};
6 opGroup = {$group : {_id:{productCode: "$orderDetails.productCode", productName:"$orderDetails.product.productName"}, "tot":{$sum:"$orderDetails.quantityOrdered"}}};
7 opSort = {$sort : {tot:-1}};
8 opLimit = {$limit: 10};
9 db.orders.aggregate([opUnwind, opGroup, opSort, opLimit]);
10
11
12
13
14
15
16
17
18
```

Find Document Document Document Document Aggregate 

 50 Documents 1 to 10

```
1 {
2   "_id" : {
3     "productCode" : "S18_3232",
4     "productName" : [
5       "1992 Ferrari 360 Spider red"
6     ]
7   },
8   "tot" : NumberInt(1808)
9 }
10 {
11   "_id" : {
12     "productCode" : "S18_1342",
13     "productName" : [
14       "1937 Lincoln Berline"
15     ]
16   },
17   "tot" : NumberInt(1111)
18 }
```

Les 5 clients les plus fidèles :

```
11 // 3 - Les 5 clients les plus fidèles
12 opProject = { $project: { paymentTotal: { $sum: "$payments.amount"}, "customerNumber": 1, "customerName":1, "employee":1}}
13 opSort = {$sort:{"paymentTotal":-1}}
14 opLimit = {$limit: 5}
15 db.customerspayments.aggregate([opProject, opSort, opLimit])
16
```

Find Document Document Document Document Aggregate Document Document Document Aggregate ⌕

50 Documents 1 to 5

```
1 {
2   "_id" : ObjectId("5bfc1c571bebbf50444deac0"),
3   "customerNumber" : NumberInt(141),
4   "customerName" : "Euro+ Shopping Channel",
5   "paymentTotal" : 715738.98
6 }
7 {
8   "_id" : ObjectId("5bfc1c571bebbf50444deabb"),
9   "customerNumber" : NumberInt(124),
10  "customerName" : "Mini Gifts Distributors Ltd.",
11  "paymentTotal" : 584188.24
12 }
13 {
14   "_id" : ObjectId("5bfc1c571bebbf50444deab8"),
15   "customerNumber" : NumberInt(114),
16   "customerName" : "Australian Collectors, Co.",
17   "paymentTotal" : 180585.07
18 }
19
```

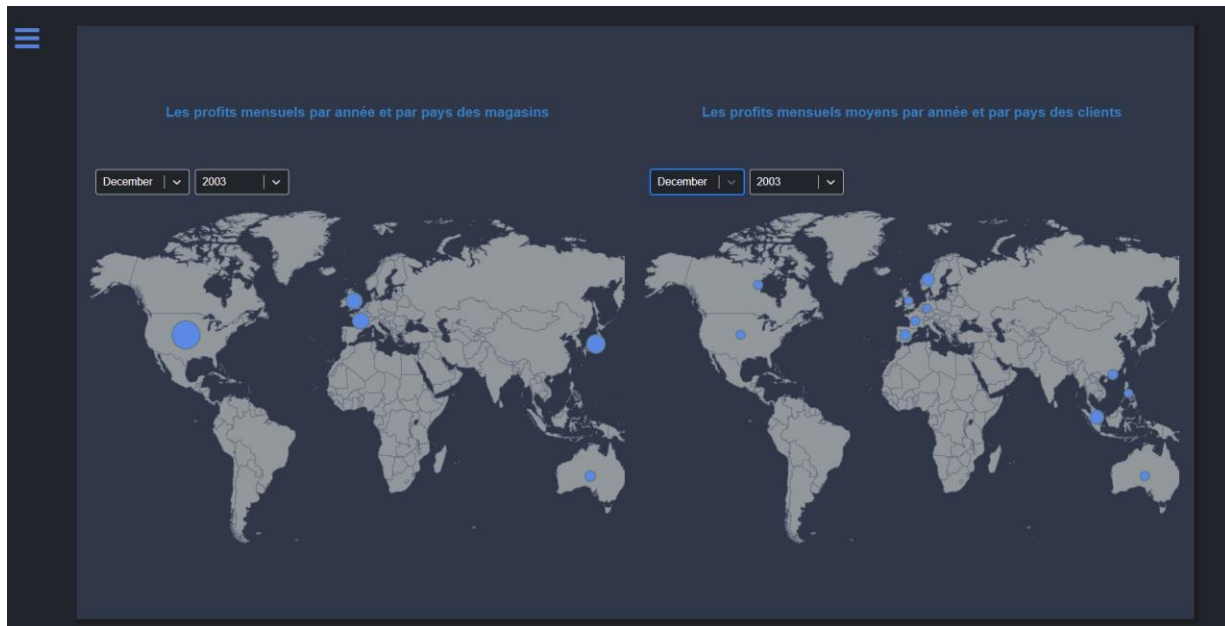
Chiffres d'affaires mensuel pour l'année 2004 :

```
17 // 4 - Chiffres d'affaires mensuel pour l'année 2004
18 opMatch = { $match: { "paymentDate": { $gt : "2004-01-01", $lt : "2005-01-01" }}}
19 opGroup = {$group : {_id: {$substr: ['$paymentDate', 5, 2]}, paymentsMonthTotal: {$sum: "$amount"}}};
20 opSort = {$sort: {"_id":1}}
21 db.payments.aggregate([opMatch, opGroup, opSort])
22
```

Find Document Document Document Document Aggregate Document Document Document

50 Documents 1 to 12

```
1 {
2   "_id" : "01",
3   "paymentsMonthTotal" : 234152.13
4 }
5 {
6   "_id" : "02",
7   "paymentsMonthTotal" : 106652.01000000001
8 }
9 {
10  "_id" : "03",
11  "paymentsMonthTotal" : 404603.21
12 }
13 {
14   "_id" : "04",
15   "paymentsMonthTotal" : 173245.96
16 }
17
```



La deuxième vue, montre les 2 dernières requêtes complexes avec map/Reduce et où l'utilisateur peut choisir les années.

Les profits mensuels par année et par pays des magasins :

```

24 // 5 - Mensual profit in 2004 by offices location
25 mapFunction = function () {
26   if(this.employee.office[0] && this.payments){
27     for(var i=0; i< this.payments.length; i++){
28       if(this.payments[i].paymentDate.substr(0,4) == "2004"){
29         emit({ "country" : this.employee.office[0].country, "month": this.payments[i].paymentDate.substr(5,2)},this.payments[i].amount)
30       }
31     }
32   }
33 };
34
35 reduceFunction = function (key, values) {
36   return Array.sum(values)
37 };
38 queryParam = {"query":{}};
39 db.customersofficespayments.mapReduce(mapFunction, reduceFunction, queryParam);

```

Find	Document	Document	Document	Document	Aggregate	Document	Document	Document	Aggregate	Document	Document
Documents 1 to 1											

```

1  "results" : [
2    {
3      "_id" : {
4        "country" : "Australia",
5        "month" : "02"
6      },
7      "value" : 22474.17
8    },
9    {
10     "_id" : {
11       "country" : "Australia",
12       "month" : "03"
13     },
14     "value" : 44894.74
15   },
16 ]
17

```

Les profits mensuels moyens par année et par pays des clients :

```
42 // 6 - Mean mensual expenses from customers in 2014 by customers locations
43 mapFunction = function () {
44   if(this.payments){
45     for(var i=0; i< this.payments.length; i++){
46       if(this.payments[i].paymentDate.substr(0,4) == year){
47         emit({ "country" : this.country, "month": this.payments[i].paymentDate.substr(5,2)},{ "avg":this.payments[i].amount, "sum":this.payments[i].amount, "nb":1})
48       }
49     }
50   }
51 };
52
53 reduceFunction = function (key, values) {
54   sum = 0; nb = 0;
55   for (i=0; i< values.length ; i++){
56     sum += values[i].sum;
57     nb += values[i].nb;
58   }
59   return {"avg" : sum / nb, "sum" : sum, "nb" : nb};
60 };
61
62 queryParam = {"query":{},"scope":{"year":"2005"},"out":{"inline":true}};
63 db.customerspayments.mapReduce(mapFunction, reduceFunction, queryParam);
```

Document	Document	Aggregate	Document	Document	Document	Aggregate	Document	Document	Document	Aggregate	Text	Text	Document	Document
----------	----------	-----------	----------	----------	----------	-----------	----------	----------	----------	-----------	------	------	----------	----------

50 Documents 1 to 1

```
1  "results" : [
2    {
3      "_id" : {
4        "country" : "Australia",
5        "month" : "01"
6      },
7      "value" : {
8        "avg" : 31835.36,
9        "sum" : 31835.36,
10       "nb" : 1.0
11     }
12   },
13   {
14     "_id" : {
15       "country" : "Australia",
16       "month" : "02"
17     },
18     "value" : {
19       "avg" : 27083.78,
20       "sum" : 27083.78,
21       "nb" : 1.0
22     }
23   }
24 ]
```


BONUS : DEPLOIEMENT DU PROJET DANS LE CLOUD : AWS

Dans le but d'étendre le champ de nos connaissances nous nous sommes penchés sur un déploiement via le cloud. Nous avons choisi d'utiliser Amazon Web Services (AWS) qui, en étant gratuit pendant un an (en fonction des ressources utilisées), nous permettait également d'avoir accès à la communauté la plus importante dans le domaine du cloud, car AWS est leader dans son domaine.

Nous avons donc créé une simple instance via Terraform qui est un soft édité par la société Hashicorp qui permet de faire de l'Infrastructure as Code. Grâce à cela nous pouvons codifier notre infrastructure de A à Z (le réseau, les VM, le stockage block, le stockage objet, les règles de sécurité) ce qui permet de lancer, modifier ou supprimer en une commande son infrastructure cloud. Cela reste très pratique pour le versionning de code.

Voici un bout de code qui permet par exemple la création d'une instance en lui attribuant un nom, son OS, sa zone de disponibilité (datacenter dans une région) et son sous-réseau par exemple :

```
instance.tf
1 resource "aws_instance" "server" {
2   name           = "${var.name_instance}"
3   ami            = "${var.ami_id}"
4   instance_type  = "${var.instance_type}"
5   zone           = "${var.zone}"
6
7   network_interface {
8     network_interface_id = "${data.aws_subnet.subnet.id}"
9   }
10 }
```

Une fois que notre infrastructure ait été mise en place il nous restait plus qu'à avoir le code source de notre application via un git clone, d'installer les composants nécessaires, d'importer les json pour MongoDB puis de laisser tourner notre applicatif.

Vous pouvez voir le résultat de notre travail à cette adresse : <http://35.180.61.119/>

CONCLUSION

Ce projet nous a permis de nous intéresser aux systèmes de base de données NoSQL, leurs avantages comme leurs inconvénients. Nous avons pu appliquer les technologies NoSQL dans un cas d'utilisation simple comme une application web orientée expérience utilisateur.

Ce projet nous a aussi permis d'intégrer les connaissances acquises pendant les cours comme la dénormalisation des données, la manipulation des requêtes simples find ou aggregates mais aussi des plus complexes en utilisant des map/reduce.

Nous avons aussi été curieux dans la manière de déployer le projet dans le cloud à l'aide de AWS, ce qui facilite la visualisation en temps réel sans avoir à l'installer.

La suite de ce projet est de pouvoir intégrer une vue d'administration qui pourrait afficher les différentes statistiques sur les données.

Pour suivre l'évolution du projet :

Git : <https://github.com/MiguelRamosF/classicmodels>