

Ro basicstyle

UNIVERSITÉ LIBRE DE BRUXELLES

Rapport : HORECA

Thomas Perale (408160)

INFO-H-303 Base de données

Esteban Zimányi, Michaël Waumans

Table des matières

1	Diagramme entité association	3
1.1	Diagramme	3
1.2	Contraintes	3
2	Modèle relationnel	3
2.1	Modèle	3
2.2	Contraintes	4
3	Hypothèses	4
4	Justification	4
5	Script SQL DDL	5
6	Requêtes	8
6.1	Tous les Utilisateurs qui apprécient au moins 3 établissements que l'utilisateur <i>Brenda</i> apprécie.	8
6.1.1	Algèbre relationnelle	8
6.1.2	Calcul relationnel	8
6.2	Tous les établissements qu'apprécie au moins un utilisateur qui apprécie tous les établissements que <i>Brenda</i> apprécie.	8
6.3	Tous les établissements pour lesquels il y a au plus un commentaire.	9
6.3.1	Algèbre relationnelle	9
6.3.2	Calcul relationnel	10
6.4	La liste des administrateurs n'ayant pas commenté tous les établissements qu'ils ont créés.	10
6.4.1	Algèbre relationnelle	10
6.4.2	Calcul relationnel	10
6.5	La liste des établissements ayant au minimum trois commentaires, classée selon la moyenne des scores attribués.	11
6.5.1	Algèbre relationnelle	11
6.5.2	Calcul relationnel	11
6.6	La liste des labels étant appliqués à au moins 5 établissements, classée selon la moyenne des scores des établissements ayant ce label.	11
6.6.1	Algèbre relationnelle	11
6.6.2	Calcul relationnel	12
7	Implémentation	12
7.1	Language et libraires	12
7.2	Script d'insertion de données	12
7.3	Instruction d'installation de l'application	17
7.4	Liaison avec la base de donnée.	17
8	Application	17
8.1	Page principale.	17
8.2	Établissements	17
8.3	Utilisateur	17
9	Apports personnels	17
9.1	Site adapté aux mobiles	17
9.2	Utilisateur administrateur	18
9.3	Localisation des restaurants sur une carte	18
9.4	Géolocalisation sur la carte	19

9.5	Gestion des photos	19
9.6	Recherche de label similaire.	19
9.7	Vote dans les commentaires	20

1 Diagramme entité association

1.1 Diagramme

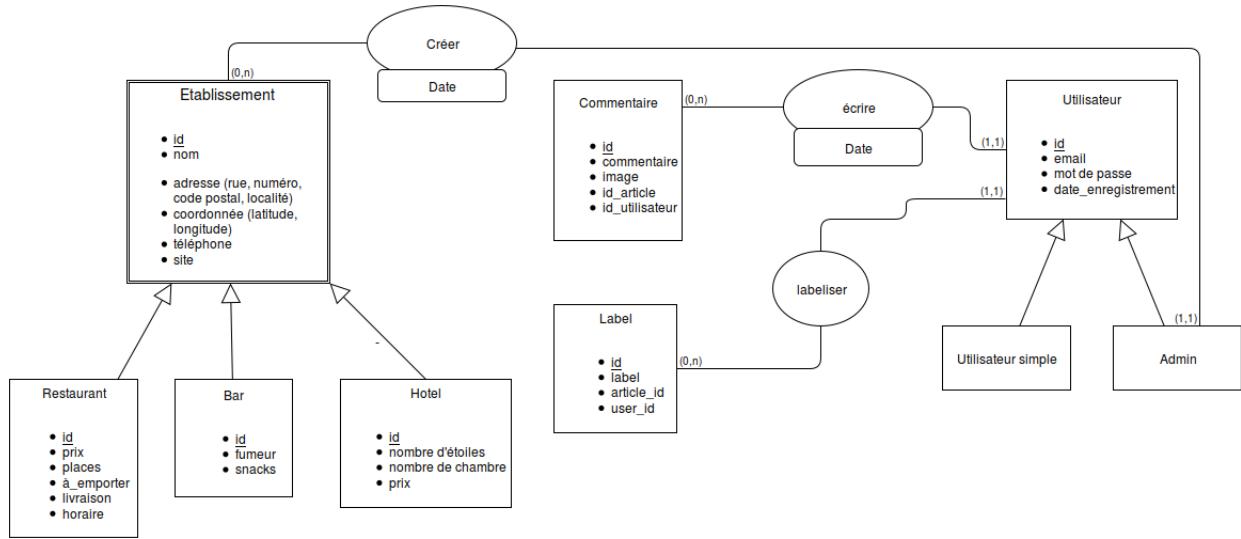


FIGURE 1 – Diagramme entité association

1.2 Contraintes

Les contraintes sont les suivantes :

- Le couple (Longitude, Latitude) est unique.
- La longitude et la latitude doivent être compris entre -180 et 180.
- Le nom d'utilisateur est unique.
- L'email d'utilisateur est unique.
- Pour les commentaires, le couple (IdUtilisateur, date) est unique. Donc un même utilisateur ne peut pas écrire deux commentaires en même temps.
- Pour les labels, le triplet : (Label, IdArticle, IdUtilisateur) est unique, ça veut dire qu'une personne ne peut pas ajouter deux fois le même label.
- Le nombre d'étoile (commentaire et hotel), doit être compris entre 1 et 5.
- Les ID des hotels/bars/restaurants référencent ceux de la table établissement, chaque établissement est soit un hotel, soit un bar, soit un restaurant, il ne peut pas être deux type d'établissement en même temps.
- La Date de création d'un établissement doit précéder celle de ses commentaires.
- La Date d'enregistrement d'un admin doit précéder celle de création d'un établissement par celui-ci.
- La Date d'enregistrement d'un utilisateur doit être antérieur à celle où il a écrit un commentaire.

2 Modèle relationnel

2.1 Modèle

Utilisateur(Id, Email, MotDePasse, DateEnregistrement, Admin(0, 1))

Utilisateur.Admin indique si l'utilisateur est un admin ou non.

Etablissement(Id, Nom, Adresse.Rue, Adresse.Numéro, Adresse.CodePostal, Adresse.Localité, Coordonnée.Latitude, Coordonnée.Longitude, Téléphone, Site, DateCreation, Creator)

Etablissement.Creator référence Utilisateur.Id.

Restaurant(Id, Prix, Places, AEmporter, Livraison, horaire)

Restaurant.Id référence Etablissement.Id.

Bar(Id, Fumeur, Snacks)

Bar.Id référence Etablissement.Id.

Hotel(Id, NombreEtoile, NombreChambre, Prix)

Hotel.Id référence Etablissement.Id.

Commentaire(Id, Commentaire, NombreEtoile, Date, Image, IdArticle, IdUtilisateur)

Commentaire.IdArticle référence Etablissement.Id.

Commentaire.IdUtilisateur référence Utilisateur.Id.

Label(Id, Label, IdArticle, IdUtilisateur)

Commentaire.IdArticle référence Etablissement.Id.

Commentaire.IdUtilisateur référence Utilisateur.Id.

2.2 Contraintes

- On doit d'abord créer l'établissement avant de créer sa spécialisation (restaurant, bar, hotel) de manière à pouvoir référencer cet établissement.

3 Hypothèses

Les utilisateur "admin" sont encodé directement dans la base de donnée.

Il est marqué dans l'énoncé :

Les utilisateurs peuvent commenter plusieurs fois le même établissement à des dates différentes.

Les dates seront stocké dans ma base de donnée sous forme de *timestamp*, dès lors un utilisateur ne pourra pas envoyer deux message lors de la même seconde.

Un administrateur peut modifier un établissement, mais j'ai décidé de ne pas stocker les dates de modification mais seulement celle de création.

Les adresses mails ainsi que les mot de passes des *Utilisateur* qui nous sont fournis dans le fichier *Data.zip* peuvent être choisis.

4 Justification

J'ai décidé pour la généralisation des établissements d'hériter les clés principales qui sont dans établissement. De cette manière j'évite de devoir redéclarer à chaque fois dans chaque table spécialisée (hotel, restaurant ou bar) les mêmes données. De plus la recherche de d'établissement par nom est rendue plus simple.

Pour le généralistion des utilisateurs j'ai décidé d'ajouter une colonne "Admin" dans la table utilisateur pour permettre de savoir si un utilisateur est admin ou non.

En ce qui concerne les *labels* j'ai décidé de mettre l'*ID* de l'utilisateur dans une des colonnes, de cette façon on sait facilement savoir si une personne a déjà ajouté un certain label ou non.

5 Script SQL DDL

..../db/create.sql

```
1 PRAGMA foreign_keys = ON;
2 PRAGMA busy_timeout=30000;
3
4 CREATE TABLE IF NOT EXISTS establishment (
5     id INTEGER PRIMARY KEY AUTOINCREMENT,
6
7     latitude REAL
8         CHECK(-90<=latitude AND latitude <=90) ,
9     longitude REAL
10        CHECK(-180<=longitude AND longitude <=180) ,
11
12     name TEXT NOT NULL,
13
14     address_street TEXT NOT NULL,
15     address_town TEXT NOT NULL,
16     address_number INTEGER NOT NULL,
17     address_zip INTEGER NOT NULL,
18
19     phone_number INTEGER(13) NOT NULL,
20
21     website TEXT,
22         — CHECK (website LIKE '%.%.%'),
23
24     picture BLOB,
25
26     creation_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
27
28     created_by TEXT,
29     FOREIGN KEY (created_by) REFERENCES account (username)
30         ON UPDATE CASCADE
31         ON DELETE CASCADE,
32
33     UNIQUE(latitude , longitude)
34 );
35
36
37 CREATE TABLE IF NOT EXISTS restaurant (
38     id INTEGER PRIMARY KEY NOT NULL REFERENCES establishment(id)
39         ON UPDATE CASCADE
40         ON DELETE CASCADE,
41
42     price INTEGER NOT NULL,
43
44     seat_number INTEGER NOT NULL,
45
46     takeaway BOOLEAN NOT NULL,
47
48     delivery BOOLEAN NOT NULL,
```

```

49
50     timetable INTEGER(7)
51         CHECK(0<=timetable AND timetable<=9)
52 );
53
54 CREATE TABLE IF NOT EXISTS bar (
55     id INTEGER PRIMARY KEY NOT NULL REFERENCES establishment ( id )
56         ON UPDATE CASCADE
57         ON DELETE CASCADE,
58
59     smokers BOOLEAN,
60
61     snacks BOOLEAN
62 );
63
64 CREATE TABLE IF NOT EXISTS hotel (
65     id INTEGER PRIMARY KEY NOT NULL REFERENCES establishment ( id )
66         ON UPDATE CASCADE
67         ON DELETE CASCADE,
68
69     stars INTEGER NOT NULL
70         CHECK(0<=stars AND stars<=5),
71
72     room_number INTEGER NOT NULL
73         CHECK(0<room_number) ,
74
75     price INTEGER NOT NULL
76 );
77
78 CREATE TABLE IF NOT EXISTS account (
79     username TEXT PRIMARY KEY NOT NULL,
80
81     email TEXT NOT NULL
82         CHECK ( email LIKE '%@%.%' ) ,
83
84     password TEXT NOT NULL,
85
86     admin BOOL DEFAULT 0 ,
87
88     creation_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
89
90     UNIQUE(email)
91 );
92
93 CREATE TABLE IF NOT EXISTS comments (
94     id INTEGER PRIMARY KEY AUTOINCREMENT,
95
96     username TEXT NOT NULL REFERENCES account ( username )
97         ON UPDATE CASCADE
98         ON DELETE CASCADE,
99
100    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

```

```

101
102 establishment_id INTEGER NOT NULL REFERENCES establishment ( id )
103     ON UPDATE CASCADE
104     ON DELETE CASCADE,
105
106 rating INTEGER NOT NULL
107     CHECK(0<rating AND rating <=5),
108
109 comment_text TEXT NOT NULL,
110
111 picture_attached BLOB,
112
113     UNIQUE(username , timestamp , establishment_id )
114 );
115
116 CREATE TABLE IF NOT EXISTS comment_rating (
117     id INTEGER PRIMARY KEY AUTOINCREMENT,
118
119     username TEXT NOT NULL REFERENCES account ( username )
120         ON UPDATE CASCADE
121         ON DELETE CASCADE,
122
123     comments INTEGER NOT NULL REFERENCES comments( id )
124         ON UPDATE CASCADE
125         ON DELETE CASCADE,
126
127     UP BOOLEAN DEFAULT 0 ,
128     DOWN BOOLEAN DEFAULT 0 ,
129
130     timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
131
132     UNIQUE(username , comments )
133 );
134
135 CREATE TABLE IF NOT EXISTS label (
136     id INTEGER PRIMARY KEY AUTOINCREMENT,
137
138     name TEXT NOT NULL,
139
140     username TEXT NOT NULL REFERENCES account ( username )
141         ON UPDATE CASCADE
142         ON DELETE CASCADE,
143
144     establishment_id INTEGER NOT NULL REFERENCES establishment ( id )
145         ON UPDATE CASCADE
146         ON DELETE CASCADE,
147
148     UNIQUE (name , establishment_id , username )
149 );

```

6 Requêtes

6.1 Tous les Utilisateurs qui apprécient au moins 3 établissements que l'utilisateur *Brenda* apprécie.

Les résultats de cette requête sont visibles dans l'application à la page [/user/Brenda](#).

..../db/1.sql

```
1 --- Tout les utilisateurs qui aiment trois établissement que brenda
2 apprécie .
3 --- Mais qui ne tient pas comptes des personnes qui commentent sur le même
4 --- établissement
5 SELECT DISTINCT c1.username
6 FROM comments AS c1
7     INNER JOIN comments AS c2
8         ON "Brenda"= c2.username
9             AND c1.establishment_id=c2.establishment_id
10            AND c2.rating >=4 AND c1.rating >=4
11 WHERE c1.username!="Brenda"
12 GROUP BY c1.username HAVING COUNT(c1.username) >=3;
```

6.1.1 Algèbre relationnelle

$$\Pi_{c1.username} comments \sigma_{count(c1.username) \geq 3} \quad (1)$$

$\bowtie' Brenda' = c2.username \text{ AND } c1.establishment_id = c2.establishment_id \text{ AND } c2.rating >= 4 \text{ AND } c1.rating >= 4$ (2)

6.1.2 Calcul relationnel

$$\{c1.username | comments(c1) \wedge \dots\} \quad (3)$$

$$\exists c2(\text{comments}(c2)) \quad (4)$$

$\wedge c2.username = 'Brenda'$ (5)

$$\wedge c1.establishment_id = c2.establishment_id \quad (6)$$

$$\wedge c1.rating \geq 4 \wedge c2.rating \geq 4 \quad (7)$$

$$\wedge c1.username != 'Brenda') \} \quad (8)$$

6.2 Tous les établissements qu’apprécie au moins un utilisateur qui apprécie tous les établissements que *Brenda* apprécie.

..../db/2.sql

```
1 -- Tout les établissement qu 'apprecie au moins un utilisateurs qui apprécie
2 -- tout les établissement que Brenda apprécie .
3
4 --SELECT DISTINCT c . establishment_id
5 --FROM comments AS c , account AS a
6 --WHERE c . username=a . username
7 --      AND c . rating >= 4
```

```

8   --      AND a.username NOT IN (
9   --          SELECT DISTINCT c1.username
10  --         FROM comments AS c1
11  --             INNER JOIN comments AS c2
12  --                 ON "Brenda"= c2.username
13  --                     AND c1.establishment_id=c2.establishment_id
14  --                     AND c2.rating>=4 AND c1.rating>=4
15  --         WHERE c1.username=c2.username
16  --     )
17  --;
18
19
20 SELECT DISTINCT e.id , e.name
21 FROM establishment AS e , comments AS c
22 WHERE c.rating>=4
23     AND c.username IN (
24     — Tout les utilisateur qui aime tout les même truc que brenda .
25     SELECT DISTINCT c1.username
26     FROM comments c1
27     INNER JOIN comments AS c2
28     ON c2.username="Brenda"
29     AND c1.establishment_id=c2.establishment_id
30     WHERE c1.rating>=4 AND c2.rating>=4
31     )
32 ;
33
34 SELECT c1.username
35 FROM comments c1
36 INNER JOIN comments AS c2
37     ON c2.username="Brenda"
38     AND c1.establishment_id=c2.establishment_id
39 WHERE c1.rating>=4 AND c2.rating>=4
40 ;

```

6.3 Tous les établissements pour lesquels il y a au plus un commentaire.

Les résultats de cette requête sont visible dans l'application à la page */establishments/discover*

..../db/3.sql

```

1   — Tout les établissement avec maximum un commentaire .
2
3 SELECT c.establishment_id
4 FROM comments c
5 GROUP BY c.establishment_id HAVING COUNT(c.establishment_id)<=1;

```

6.3.1 Algèbre relationnelle

$$\Pi_{c.establishment_id} \text{comments} \sigma_{\text{count}(c.establishment_id) \leq 1} \Pi_{c.establishment_id} \quad (9)$$

6.3.2 Calcul relationnel

$$\{c.establishment_id | comments(c) \wedge \quad \quad \quad (10)$$

$$\exists(c.username_1, c.username_2) \quad \quad \quad (11)$$

$$\wedge(c.username_1 \neq c.username_2)\} \quad \quad \quad (12)$$

6.4 La liste des administrateurs n'ayant pas commenté tous les établissements qu'ils ont créés.

..../db/4.sql

```

1 --- Tout les administrateur qui n'ont pas commenté
2 --- tous les établissments qu'ils ont crée
3
4 SELECT a.username
5 FROM account AS a
6 WHERE a.admin=1
7     AND a.username NOT IN (
8         SELECT e.created_by
9         FROM establishment AS e
10        INNER JOIN comments AS c
11            ON e.id = c.establishment_id
12            AND e.created_by != c.username
13        GROUP BY c.username HAVING COUNT(c.establishment_id)>0
14    )
15 ;

```

6.4.1 Algèbre relationnelle

$$\Pi_{a.username} account \quad \quad \quad (13)$$

$$\sigma_{a.admin \text{ AND}} \quad \quad \quad (14)$$

$$(\Pi_{e.created_by} establishment) \quad \quad \quad (15)$$

$$\bowtie_{e.id=c.establishment_id \text{ AND } e.created_by!=c.username} \quad \quad \quad (16)$$

$$\Pi_{c.establishment_id} \quad \quad \quad (17)$$

6.4.2 Calcul relationnel

$$\{a.username | account(a) \wedge \quad \quad \quad (18)$$

$$\exists a (a.admin = 1) \wedge \quad \quad \quad (19)$$

$$\neg(e.created_by | establishment(e) \quad \quad \quad (20)$$

$$\exists c (comments(c) \wedge \quad \quad \quad (21)$$

$$e.id = c.establishment_id \wedge e.created_by \neq c.username) \quad \quad \quad (22)$$

$$\}) \} \quad \quad \quad (23)$$

6.5 La liste des établissements ayant au minimum trois commentaires, classée selon la moyenne des scores attribués.

Les résultats de cette requête sont visible dans l'application à la page `/establishments/ratings`

`./db/5.sql`

```

1  -- La liste des établissements ayant au minimum trois commentaires ,
2  -- classée selon la moyenne des scores attribués .
3
4  SELECT AVG(c.rating) , c.establishment_id
5  FROM comments AS c
6  GROUP BY c.establishment_id HAVING COUNT(c.establishment_id)>=3
7  ORDER BY AVG(c.rating) ;

```

6.5.1 Algèbre relationnelle

$$\Gamma_{AVG(c.rating)} \Pi_{AVG(c.rating), c.establishment_id} comments \sigma_{count(c.establishment_id) >= 3} \Pi_{c.establishment_id} \quad (24)$$

6.5.2 Calcul relationnel

$$\{c.establishment_id | comments(c) \wedge \quad (25)$$

$$\exists c_1, c_2, c_3 (\quad (26)$$

$$c_1.establishment_id = c_2.establishment_id = c_3.establishment_id \quad (27)$$

$$\wedge c_1.username! = c_2.username! = c_3.username) \} \quad (28)$$

6.6 La liste des labels étant appliqués à au moins 5 établissements, classée selon la moyenne des scores des établissements ayant ce label.

Les résultats de cette requête sont visible dans l'application à la page `/establishments/label/...`

`./db/6.sql`

```

1  -- La liste des labels étant appliqués à au moins 5 établissements ,
2  -- classée selon la moyenne des scores des établissements ayant ce label .
3
4  SELECT AVG(c.rating) , l.name
5  FROM label AS l
6  JOIN comments AS c
7  ON c.establishment_id=l.establishment_id
8  GROUP BY l.name HAVING COUNT(DISTINCT l.establishment_id)>=5
9  ORDER BY AVG(c.rating)
10 ;

```

6.6.1 Algèbre relationnelle

$$\Gamma_{AVG(c.rating)} \Pi_{AVG(c.rating), l.name} label \sigma_{count(DISTINCT l.establishment_id) >= 5} \Pi_{l.name} \quad (29)$$

$$\bowtie_{c.establishment_id=l.establishment_id} \quad (30)$$

6.6.2 Calcul relationnel

$$\{l.name | \text{label}(l) \wedge \quad (31)$$

$$\exists l_1, l_2, l_3, l_4, l_5 \quad (32)$$

$$l_1.establishment_id! = l_2.establishment_id! = l_3.establishment_id! = l_4.establishment_id! = l_5.establishment_id \wedge \quad (33)$$

$$l_1.name = l_2.name = l_3.name = l_4.name = l_5.name \quad (34)$$

$$\} \quad (35)$$

7 Implémentation

7.1 Language et libraires

- *Express.js* comme framework web pour le language *node.js*.
- *Handlebars.js* comme moteur de template pour *express.js*.
- *Bootstrap* et *jQuery* comme framework de frontend.
- *SQLite3* comme moteur de base de donnée.

7.2 Script d'insértion de données

..../init.py

```

1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 import dateutil.parser
5 import sqlite3 as lite
6 import sys
7 import datetime
8 from bs4 import BeautifulSoup
9
10 DATA_PATH = "./db/datas/"
11
12 USER_DICT = dict()
13
14 con = lite.connect('./db/test.db')
15
16 def parse_date(date):
17     foo = dateutil.parser.parse(date)
18     print(foo)
19     return foo
20
21 def new_user(username, db, date=None, admin=0):
22     user = USER_DICT.get(username)
23     if user:
24         if date and date < user['date']:

```

```

25     cmd = "UPDATE_account SET creation_date=%s WHERE
26         username=%s" % (date, username)
27     print(cmd)
28     db.execute(cmd)
29     user[ 'date' ] = date
30     if admin:
31         cmd = "UPDATE_account SET admin=%i WHERE username=%s" %
32             (admin, username)
33         db.execute(cmd)
34     else:
35         if date:
36             # Si l'utilisateur n'existe pas.
37             cmd = "INSERT INTO account (username, email, password, admin,
38                 creation_date) VALUES ('%s', '%s', 'default', %i, '%s')" %
39                 (username, username + "@gmail.com", admin, date)
40             print(cmd)
41             db.execute(cmd)
42             date = datetime.datetime.now()
43
44             USER_DICT[username] = dict()
45             USER_DICT[username][ 'date' ] = date
46
47             con.commit()
48
49
50 def parse_comments(comments, _id, db):
51     for comment in comments:
52         username = comment[ 'nickname' ]
53         timestamp = parse_date(comment[ 'date' ])
54
55         new_user(username, db, timestamp)
56
57         rating = comment[ 'score' ]
58         comment_text = comment.text
59
60         db.execute("INSERT INTO comments (username, timestamp,
61             establishment_id, rating, comment_text) VALUES (?, ?, ?, ?, ?)",
62             (username, timestamp, _id, int(rating), comment_text))
63
64         con.commit()
65
66
67 def parse_label(label, _id, db):
68     for tag in label:
69         name = tag[ 'name' ]
70         for user in tag.findall('user'):
71             new_user(user[ 'nickname' ], db)

```

```

69         cmd = "INSERT INTO `label`(`name`, `username`, `establishment_id`) "
70             VALUES ('%s', '%s', %i)" % (name, user['nickname'], _id)
71     print(cmd)
72     db.execute(cmd)
73
74 def parse_bar(s, conn):
75     """
76     Parse bar file .
77     """
78     for cafe in s.findall("cafe"):
79         created_by = cafe['nickname']
80         creation_date = parse_date(cafe['creationdate'])
81
82         db = con.cursor()
83         new_user(created_by, db, creation_date, 1)
84         db.close()
85
86         obj = {}
87         info = cafe.find('informations')
88         name = info.find('name').text # Bar name.
89         address_street = info.find('address').find('street').text
90         address_number = info.find('address').find('num').text
91         address_zip = info.find('address').find('zip').text
92         address_town = info.find('address').find('city').text
93         longitude = info.find('address').find('longitude').text
94         latitude = info.find('address').find('latitude').text
95
96         phone_number = info.find('tel').text
97         website = info.find('website')
98         if website:
99             website = website.text
100
101         cmd = "insert_into_establishment_"
102             (latitude, longitude, name, address_street, address_town, address_number, address_z
103             .....values_(%f,%f,'%s','%s','%s','%s,%i','%s','%s','%s','%s')" %
104             \
105             (float(latitude), float(longitude), name, address_street,
106             address_town, address_number, int(address_zip),
107             phone_number, website, created_by, creation_date)
108         print(cmd)
109
110         db = con.cursor()
111         db.execute("insert_into_establishment_"
112             (latitude, longitude, name, address_street, address_town, address_number, address_z
113             .....values_(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)" ,
114             (float(latitude), float(longitude), name, address_street,
115             address_town, address_number,
116             int(address_zip), phone_number, website,
117             creation_date, created_by))
118         con.commit()
119         db.close()

```

```

113     _id = db.lastrowid
114
115
116     smokers = bool(info.find('smoking'))
117     snacks = bool(info.find('snack'))
118
119     cmd = "INSERT INTO bar_(id ,smokers ,snacks ) VALUES(%i,%i,%i)" %
120         (_id ,smokers , snacks)
121     db = con.cursor()
122     db.execute(cmd)
123     con.commit()
124     db.close()
125
126     db = con.cursor()
127     parse_comments(cafe.findall('comment') , _id , db)
128     db.close()
129
130     db = con.cursor()
131     parse_label(cafe.findall('tag') , _id , db)
132     db.close()
133
134 def parse_restaurant(s , conn):
135     """
136     Parse bar file .
137     """
138     for cafe in s.findall("restaurant"):
139         created_by = cafe['nickname']
140         creation_date = parse_date(cafe['creationdate'])
141
142         db = con.cursor()
143         new_user(created_by , db , creation_date , 1)
144         db.close()
145
146         obj = {}
147         info = cafe.find('informations')
148         name = info.find('name').text # Bar name.
149         address_street = info.find('address').find('street').text
150         address_number = info.find('address').find('num').text
151         address_zip = info.find('address').find('zip').text
152         address_town = info.find('address').find('city').text
153         longitude = info.find('address').find('longitude').text
154         latitude = info.find('address').find('latitude').text
155
156         phone_number = info.find('tel').text
157         website = info.find('website')
158         if website:
159             website = website.text
160
161         cmd = "insert into establishment_
162             (latitude ,longitude ,name ,address_street ,address_town ,address_number ,address_z
163             values(%f,%f,'%s','%s','%s','%s,%i','%s','%s','%s','%s','%s')" %
164             \

```

```

162         (float(latitude), float(longitude), name, address_street,
163          address_town, address_number, int(address_zip),
164          phone_number, website, created_by, creation_date)
165      print(cmd)
166
167      db = con.cursor()
168      db.execute("insert into establishment_
169      (latitude,longitude,name,address_street,address_town,address_number,address_z
170      _id,values(?,?,?,?,?,?,?,?,?,?)",
171      (float(latitude), float(longitude), name, address_street,
172        address_town, address_number,
173        int(address_zip), phone_number, website,
174        creation_date, created_by))
175      con.commit()
176      db.close()
177
178      _id = db.lastrowid
179
180      takeaway = bool(info.find('takeaway'))
181      delivery = bool(info.find('delivery'))
182      price = int(info.find('pricerange').text)
183      seat_number = int(info.find('banquet')['capacity'])
184
185      cmd = "INSERT INTO restaurant_
186      (id,takeaway,delivery,price,seat_number) VALUES_
187      (%i,%i,%i,%i,%i)" % (_id,takeaway,delivery,price,seat_number)
188      db = con.cursor()
189      db.execute(cmd)
190      con.commit()
191      db.close()
192
193
194
195
196
197  if __name__ == "__main__":
198      parse_bar(BeautifulSoup(open(DATA_PATH + "Cafes.xml").read(),
199          'html.parser'), con)
200      parse_restaurant(BeautifulSoup(open(DATA_PATH +
201          "Restaurants.xml").read(), 'html.parser'), con)

```

7.3 Instruction d'installation de l'application

À la racine du projet.

```
1 make  
2 npm install  
3 node index.js
```

7.4 Liaison avec la base de donnée.

L'accès par l'application à la base de donnée se fait par l'ensemble des fonctions stockées dans le dossier `./db/`.

`database_utils.js` Ensemble de fonctions qui interagissent avec tout les établissements.

`restaurant_db_utils.js` Ensemble de fonctions qui interagissent avec les restaurants.

`bar_db_utils.js` Ensemble de fonctions qui interagissent avec les bars.

`hotel_db_utils.js` Ensemble de fonctions qui interagissent avec les hotels.

`label_db_utils.js` Ensemble de fonctions qui interagissent avec les labels.

`comment_utils.js` Ensemble de fonctions qui interagissent avec les commentaires.

8 Application

8.1 Page principale.

La page principale est la première page de l'application que l'utilisateur va voir. Dans cette page, plusieurs chose sont déjà possible :

- S'inscrire
- Se connecter
- Choisir parmis une selection d'établissement pris au hasard.
- Accéder à une selection d'établissement plus spécifique.
- Une carte des établissements qui sont sur le site.

8.2 Établissements

La page des établissements donne à l'utilisateur un coup d'œil sur ses caractéristiques propres. Mais c'est aussi un lieu pour les utilisateurs de donner leur avis sur l'établissement et de le classer dans des catégories. Chaque commentaire qu'un utilisateur laisse peut être mis en avant par un système de vote individuel au commentaire, de cette façon les commentaires qui sont malhonnêtes selon les utilisateurs sont visibles.

8.3 Utilisateur

9 Apports personnels

9.1 Site adapté aux mobiles

Pour que l'application soit adaptée au type de consommation actuelle, elle a été créée dans l'idée de pouvoir aussi bien être facile d'utilisation sur ordinateur que sur mobile.

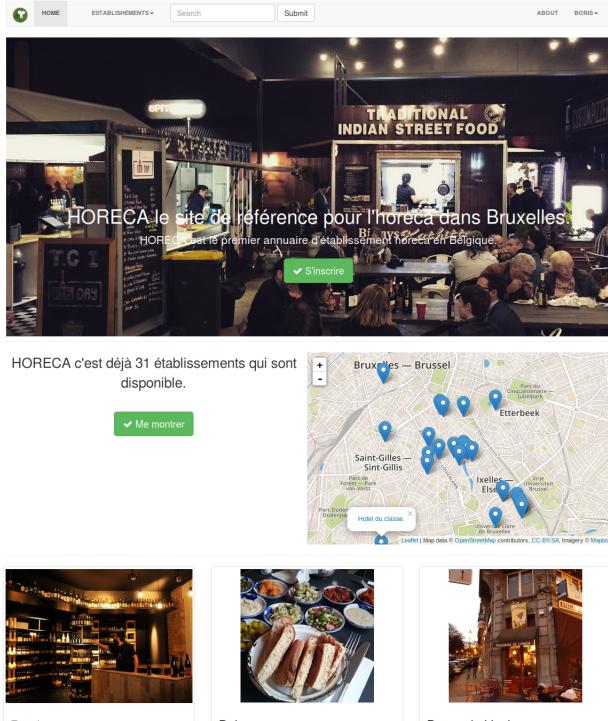


FIGURE 2 – Page principale.

`./images/establishment.png`

FIGURE 3 – Page principale.

9.2 Utilisateur administrateur

Des comptes administrateur sont à disposition pour pouvoir gérer les publications, si celle-ci doivent être changer, ou pour faire un travail de modération dans les commentaires ou les labels. Un administrateur a le pouvoir de :

- Supprimer un commentaire.
- Supprimer un label.
- Supprimer un établissement.
- Supprimer un utilisateur.
- Ajouter d'autres utilisateur au poste d'administrateur.
- Créer des restaurants.
- Créer des hotels.
- Créer des bars.
- Modifier les données des établissements.

9.3 Localisation des restaurants sur une carte

Afin de permettre de pouvoir directement voir l'emplacement des établissements sur une l'application intègre des cartes pour visualiser l'emplacement de l'établissement *HORECA*. Celles-ci sont intégrées à la page web à l'aide d'une bibliothèque Javascript, Leaflet. En plus de montrer l'emplacement sur une carte celles-ci montre aussi un rapide aperçu des caractéristiques de l'établissement.

The screenshot shows a user profile page for 'Admin : Boris'. At the top, there are links for 'HOME', 'ESTABLISSEMENTS', 'Search', 'Submit', 'ABOUT', and 'SIGN IN'. Below this, it says 'Admin : Boris' and shows the email 'Boris@gmail.com' and the creation date 'Crée le 2003-04-04 00:00:00'. The main content area is divided into two columns: 'Commentaires' and 'Labels'. Under 'Commentaires', there are three entries with their respective text content. Under 'Labels', there are three entries with their respective names.

Commentaires	Labels
Commentaire du 2008-01-06 00:00:00 Très plein les soirs de match, bruyant et enflamé.	Label Fumeur
Commentaire du 2008-02-04 00:00:00 Sympa à l'extérieur en été mais très bruyant à l'intérieur.	Label Bon marché
Commentaire du 2008-02-05 00:00:00 Bonne ambiance à partir de 23h. Déco apache. Spécialité : sous marin.	Label Terrasse

FIGURE 4 – Page utilisateur.

The screenshot shows a responsive website for a restaurant named 'Au Vieux Bruxelles'. At the top, there are links for 'HOME', 'ESTABLISSEMENTS', 'Search', 'Submit', 'ABOUT', and 'SIGN IN'. The main content area features a map of Brussels with a specific location highlighted. A callout box on the map provides details about the restaurant: 'Au Vieux Bruxelles Rue Saint-Boniface, 35 Ixelles (1050)'. To the right of the map, there is a sidebar titled 'Au Vieux Bruxelles labels' which lists 'Bon (3)' and 'Bon rapport qualité/prix (3)'. At the bottom, there is a footer with some icons and text.

FIGURE 5 – Site responsive.

9.4 Géolocalisation sur la carte

En plus de montrer la localisation des restaurants, l'application permet de pouvoir se géolocaliser sur la carte, par apport aux établissements.

Cette fonctionnalité est d'autant plus la bienvenue que l'application est adaptée aux téléphones mobiles.

9.5 Gestion des photos

L'application gère le stockage d'image dans la base de donnée. Chaque utilisateur inscrit sur le site peuvent dès lors écrire un commentaire dans la section adaptée accompagné d'une photo de l'établissement visité par exemple.

Quant aux utilisateurs dit *administrateur*, ils peuvent modifier la photo de l'établissement pour que chaque visiteur ait un aperçu de l'apparence de l'établissement.

De plus la gestion de ces photos permet de les visionner de manière confortable à l'aide du module enhance.js. En effet celui-ci permet de voir une photo *en grand*, sans devoir changer de page.

9.6 Recherche de label similaire.

Pour permettre aux visiteurs de trouver des établissements avec des contraintes spécifiques, l'application lie tout les labels de chaque établissement. Un visiteur peut dès lors retrouver tout les établissements qui possède le label de son choix, comme par exemple *Fumeur*.



FIGURE 6 – Établissement indiqué sur une carte.

🍴 Chez Théo Sodexho ULB

by Boris (2008-02-10 00:00:00)

Moyenne des utilisateurs : 2.5 étoiles.

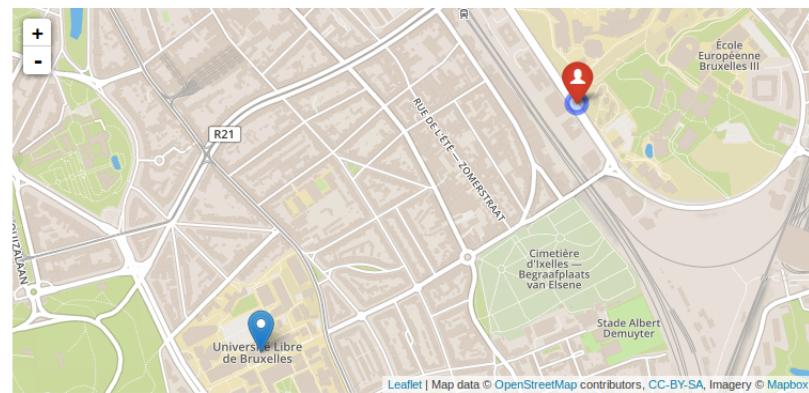


FIGURE 7 – Un marqueur indique l'endroit où l'on se trouve.

9.7 Vote dans les commentaires

Un système de vote individuel à chaque commentaire est mis en place pour que chaque utilisateur inscrive le droit de juger des commentaires qui selon lui ne sont pas raisonnable ou inversément.

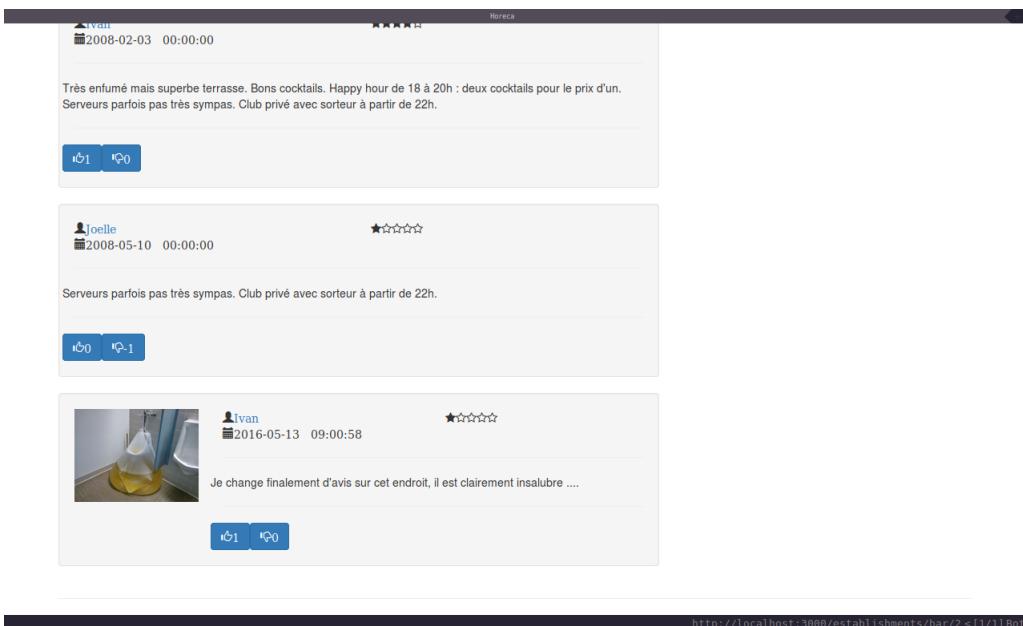


FIGURE 8 – Ici un commentaire accompagné d'une photo.

The screenshot shows a web interface for a platform called "Horeca". The top navigation bar includes links for "HOME", "ESTABLISSEMENTS", "Search", "Submit", "ABOUT", and "IVAN".

A search query "Bar à vin" has been entered in the search bar, resulting in the following message:

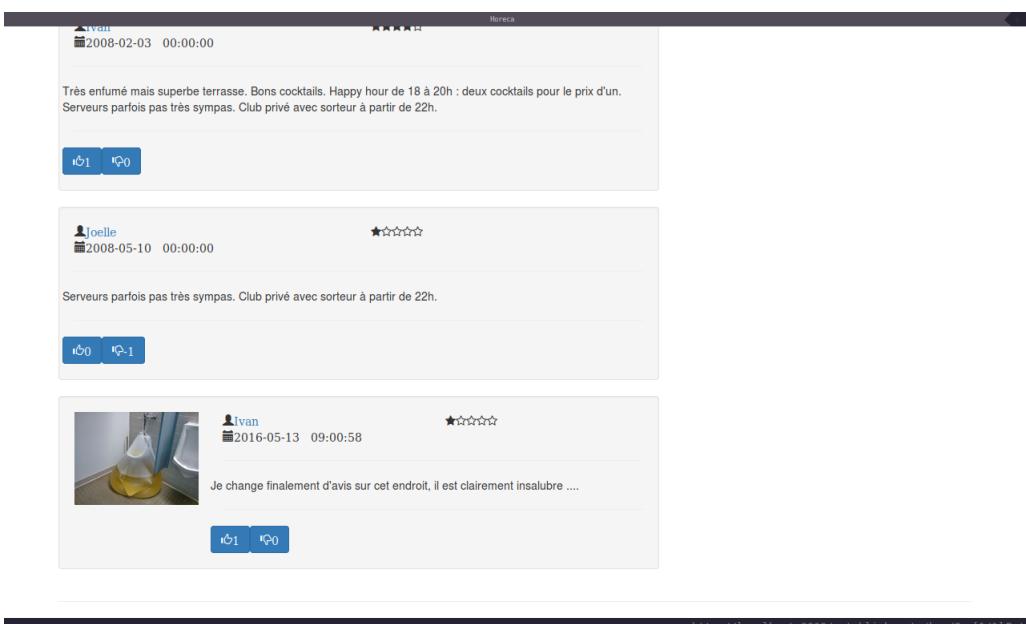
Il y a 4 établissements qui utilisent le label "Bar à vin"

The results are displayed in three cards:

- Marché au vin**
Rue du Belvédère, 14 (Ixelles).
02/640 56 10
[Me montrer](#)
- Delecta**
Rue Lannoy, 2 (Ixelles).
02/644 19 49
[Me montrer](#)
- Dico**


The URL at the bottom of the page is <http://localhost:3000/label/Bar à vin<1/1>.Top>.

FIGURE 9 – Page regroupant les établissements sous le label ”Bar à vin”.



[http://localhost:3000/establishments/bar/2-\[1/1\].box](http://localhost:3000/establishments/bar/2-[1/1].box)

FIGURE 10 – Des votes en dessous de chaque commentaire.