

Ubid - Documentation technique Livrable 2

[Ubid, un site de vente aux enchères](#)

[Les choix technologiques](#)

[Flask, le micro-framework en Python pour designer notre API](#)

[1- API RESTFul](#)

[2- Le nombre de fonctionnalités](#)

[1. CRUD Utilisateur](#)

[2. CRUD des produits](#)

[3. Gestion des tokens](#)

[4. Simulations des protections basiques contre les attaques CSRF et les mass-assignments](#)

[5. Séparation du code grâce aux BluePrints](#)

[6. Création de helpers](#)

[7. La recherche de produits](#)

[*](#)

[-](#)

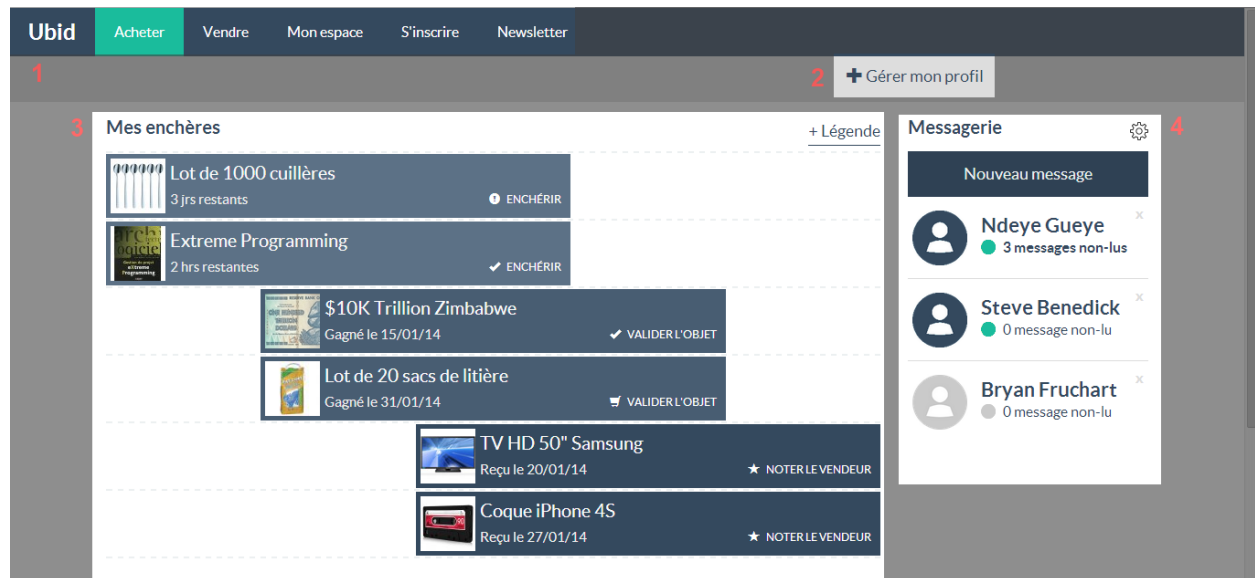
[Comment exécuter notre projet](#)

Ubid, un site de vente aux enchères

Ubid est avant tout un site de vente aux enchères, comme Ebay.

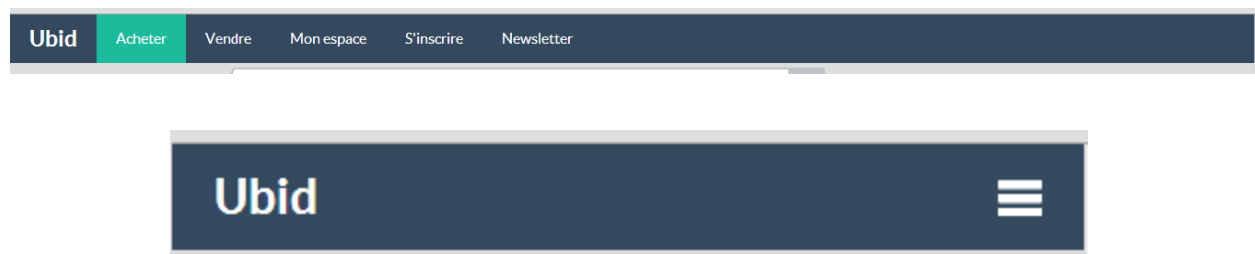
Nous avons souhaité dès le départ du projet nous démarquer des interfaces classiques, et proposer une expérience fluide et dynamique, permettant à l'utilisateur d'apprécier sa navigation.
[Partie Design]

L'accent est donc mis sur la simplicité et l'ergonomie, pour ce faire nous avons divisé les éléments du design en quatre parties :



1. La barre de navigation.
2. Le volet d'actions.
3. Le contenu principale.
4. Le contenu secondaire.

1) La barre de navigation.



La barre de navigation est une liste qui contient des liens vers les différentes pages du site. En position fixe en haut de l'écran, elle est facilement visible grâce à sa couleur sombre qui la

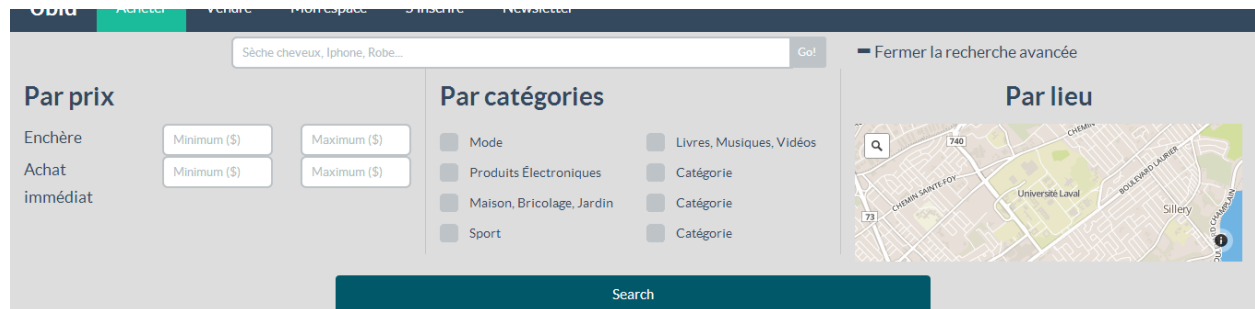
met en contraste par rapport au reste du site.

L'utilisateur peut s'en servir à tout moment pour changer de page où se repérer dans le site. Afin de ne pas alourdir le visuel du site lors de sa consultation via un appareil mobile, cette barre de navigation va prendre une autre forme pour les petits écrans.

Choix des coloris:

Bleu marine pour offrir un contraste fort avec la trame que nous avons choisi, blanc cassé pour ne pas fatiguer l'oeil de l'utilisateur tout en augmentant la clarté de l'ensemble de l'application. La section correspondante à la page actuellement visitée est quand à elle bleu turquoise, une couleur plus vive qui permet à l'utilisateur de se repérer dans l'architecture du site.

2) Le volet d'action:

The image shows a screenshot of a web application's 'Action Panel' (volet d'action) which is a grey sidebar on the left side of the page. At the top of the sidebar is a search bar with the text 'Sèche cheveux, Iphone, Robe...' and a 'Go!' button. Below the search bar are three main filter sections: 'Par prix' (By price) with buttons for 'Enchère' (Auction) and 'Achat immédiat' (Instant purchase), each with 'Minimum (\$)' and 'Maximum (\$)' input fields; 'Par catégories' (By categories) with a grid of category buttons including 'Mode', 'Produits Électroniques', 'Maison, Bricolage, Jardin', 'Sport', 'Livres, Musiques, Vidéos', and 'Catégorie'; and 'Par lieu' (By location) which features a map of a city area. At the bottom of the sidebar is a large 'Search' button. The main content area to the right of the sidebar has a dark blue header with navigation links like 'Accueil', 'Vendre', 'Mon espace', 'Ajouter', and 'Mon panier'. The main content area has a light grey background and a dark blue footer.

Le volet d'action est une zone située juste en dessous de la barre de navigation qui s'ouvre dynamiquement lorsque l'utilisateur en a besoin.

Cette zone du design contient les actions principales que l'utilisateur peut vouloir entreprendre. Formulaire, recherche avancée, paramètres du profil, tous sont disposés dans le volet d'action de la page correspondante.

Cette disposition permet d'augmenter l'utilisabilité du site puisque l'utilisateur garde inconsciemment en mémoire que c'est dans cette zone qu'il pourra trouver les boutons et formulaires lui permettant d'interagir avec le site.

Choix des coloris:

Le volet d'action est doté d'un fond gris le distinguant comme élément à part entière du design.

3) Le contenu principal:

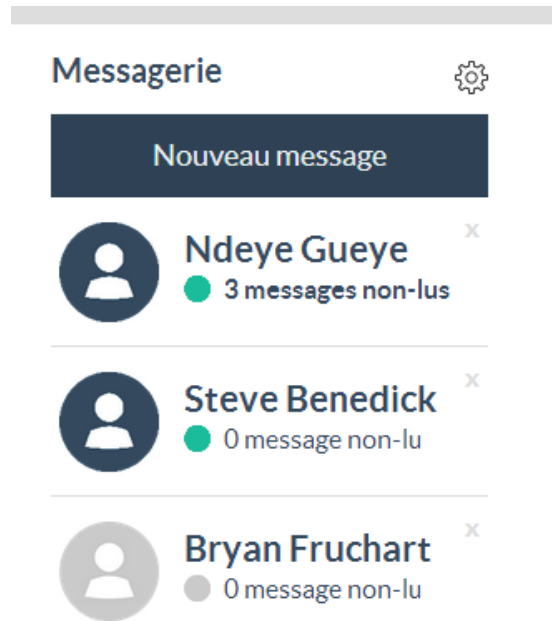


Mes ventes

Titre	Temps restant	Nombre d'enchères	Meilleure enchère	Modifier	Supprimer
Lot de 6 cuillères	2j 3h	7	22,11\$	Modifier	Supprimer
Belle mère	999j+	0	0,10\$	Modifier	Supprimer
Dépouille de Nyan Cat	4j 17h	500+	1786,33\$	Modifier	Supprimer
Développeur PHP	27j 3h	0	27,00\$	Modifier	Supprimer
Développeur Django	23j	3	600\$	Modifier	Supprimer
Développeur Ruby On Rails	2j 3h	77	900\$	Modifier	Supprimer

La zone de contenu principal, situé en dessous de la barre de navigation et du volet d'action contient, comme son nom l'indique, le contenu jugé de première importance comme la liste des produits à acheter, la liste des produits en vente de l'utilisateur etc.

3) Le contenu secondaire






La zone de contenu secondaire permet d'afficher des informations qui ne sont pas jugées comme prioritaire pour un site d'enchère.

Exemple : la boîte de messagerie d'un utilisateur.

Note particulière: le système de suivis d'une enchère.

Mes enchères

[+ Légende](#)

	Lot de 1000 cuillères 3 jrs restants	 ENCHÉRIR
	Extreme Programming 2 hrs restantes	✓ ENCHÉRIR
	\$10K Trillion Zimbabwe Gagné le 15/01/14	✓ VALIDER L'OBJET
	Lot de 20 sacs de litière Gagné le 31/01/14	✓ VALIDER L'OBJET
	TV HD 50" Samsung Reçu le 20/01/14	★ NOTER LE VENDEUR
	Coque iPhone 4S Reçu le 27/01/14	★ NOTER LE VENDEUR

Nous avons choisi une disposition particulière pour le suivi des enchères auxquelles l'utilisateur a pris part.

Les produits sur lesquelles l'utilisateur est positionné sont représentés par des rectangles dont le design et la position varient en fonction de l'avancement de l'enchère.

Explication:

- Une enchère qui n'est pas encore terminée se situe sur la gauche de l'écran et le fond de la boxe et d'un bleu peu saturé. Dans cette position, un clic sur l'enchère permet de surenchérir ou d'annuler l'enchère.
- Lorsque l'enchère est gagnée, le rectangle qui la représente passe en position central afin de signaler à l'utilisateur qu'il a remporté la mise. La couleur bleue se sature un peu plus. L'utilisateur est alors invité à valider l'achat pour que le vendeur puisse envoyer son produit.
- Enfin lorsque le vendeur indique au site qu'il a envoyé son produit, l'enchère se positionne à droite de l'écran, ce qui indique à l'utilisateur que son produit a été envoyé et qu'il ne devrait pas tarder à le recevoir. Le bleu de la box se sature encore plus pour créer une différence avec les autres états. Une fois qu'il aura reçu son produit l'utilisateur pourra alors noter le vendeur avant que l'enchère ne disparaisse totalement de cette vue.

Notez que dans le cas d'une connexion via un appareil ayant un écran de petite taille, les enchères s'alignent verticalement pour augmenter la visibilité de l'ensemble. Seul la différence

de teinte du bleu et les actions possibles pour chaque box indique à l'utilisateur l'état d'avancement d'une enchère.

Ubid

Mes enchères

— Légende

- ✓ Vous êtes la meilleur enchère
- ! Vous n'êtes pas la meilleur enchère
- 🛒 Enchère remportée, passez au paiement
- ★ Le produit a été envoyé, notez le vendeur



Lot de 1000 cuillères

3 jrs restants

! ENCHÉRIR



Extreme Programming

2 hrs restantes

✓ ENCHÉRIR



\$10K Trillion Zimbabwe

Gagné le 15/01/14 ✓ VALIDER L'OBJET



Lot de 20 sacs de litière

Gagné le 31/01/14 🛒 VALIDER L'OBJET



TV HD 50" Samsung

Reçu le 20/01/14 ★ NOTER LE VENDEUR

Les choix technologiques

Pour mener à bien Ubid, nous avons choisi de travailler avec les technologies suivantes :

- Bootstrap pour le Framework CSS
- AngularJS pour la partie cliente
- Flask (Python) pour la partie serveur
- NoSQL pour la base de donnée (Base de donnée par fichier).

La plupart de ces choix nous ont été imposés mais nous avons préféré utiliser AngularJS à Backbone. Une personne de notre équipe maîtrisant déjà cette technologie, nous pensons avoir un avantage à utiliser AngularJS. En effet qu'il s'agisse de Backbone ou d'AngularJS, les deux étaient nouveaux pour le reste de l'équipe. Nous espérons ainsi, grâce à notre choix, pouvoir aller de l'avant plus rapidement tout en apprenant AngularJS.

Bootstrap semble être la solution idéale concernant l'aspect graphique du site. En effet, ce framework fournit de nombreux style prédéfinis et également des modules additionnels présents sur internet qui peuvent facilement s'ajouter. Nous avons adapté le thème de base en ajoutant une palette de couleur flat et en nous basant sur un thème Bootstrap flat. Des ajouts ont été nécessaires pour répondre à nos besoins en matière de design et de module. Sur le long terme l'utilisation de bootstrap est une bonne chose. En effet, la structure de base permet de facilement maintenir notre code et d'apporter rapidement des modifications au thème. Pour se faire nous avons ajouté nos propres fichiers CSS. Nous pensons que cette pratique est la meilleure puisqu'elle permet de mettre à jour Bootstrap si c'est nécessaire, sans perdre les modifications que nous avons ajoutées. La prise en main de Bootstrap nous aura pris plus de temps que prévu mais reste très intuitive. De plus la documentation proposée ainsi que les nombreux exemples et mise en pratique nous donnaient un bon aperçu des possibilités et de comment arriver à nos fins.

De même que pour Bootstrap, Flask nous a été imposé par les professeurs. Étant tous novices en Python nous avons naturellement pris le temps de comprendre le fonctionnement grâce aux TP et à l'API fournie. Nous avons ensuite pu commencer à développer tout l'aspect Python que nous allons traiter en détail dans un chapitre suivant de ce rapport.

Le pari de réaliser le projet avec une gestion de données basée sur du NoSQL est assez atypique. En effet, nous avons tous été habitués à utiliser au cours de notre scolarité diverses bases de données et systèmes de gestion de base de données. Le concept nous semblait très intéressant et nous avons rapidement pris le pli d'utiliser du json pour traiter les données via notre javascript.

Peu importe la partie codée (front end, back end), nous essayons au maximum d'appliquer les règles de bonne pratique vues en cours et trouvées sur internet pour permettre un développement d'application optimale.

Nous avons par exemple divisé au maximum les différentes parties du site dans des fichiers différents pour pouvoir plus facilement modifier et maintenir le site. Il en est de même pour les fichiers CSS et JS. Nous avons essayé au maximum d'appliquer l'adage "Un fichier une tâche". Nous avons également pris soin de bien indenter notre code pour une meilleure lisibilité. Nous avons aussi commenté des parties qui nous semblaient nécessaire.

Enfin nous nous sommes penchés sur des règles de nommage afin de retrouver plus facilement les éléments. Par exemple nous nous sommes imposés des noms se rapprochant du framework bootstrap pour nos classes CSS avec l'utilisation des préfixes "btn" par exemple.

Flask, le micro-framework en Python pour designer notre API

1- API RESTFul

Pour construire notre API, nous avons du auparavant réfléchir à l'architecture de nos modèles. Nous avons ainsi découpé chaque ressource, et définis les paramètres qui étaient obligatoires, optionnels, les types et erreurs de retours.

Pour ce faire, nous avons travaillé communément sur un tableur classique afin de déterminer notre structure de donnée.

	A	B	C	D	E	F	G	H
	Nom collection	Route de base	Ressource (route)	Type	paramètre1	à implémenter	Réponse ok	Réponse ko
1		/login	login	POST	username, password		200	401 (Pas d'username trouv
3		/logout	logout	POST			201	none
4		/	register	POST	username*, password*, email*, address1*, address2, city*, postalcode*, country*, firstname*, lastname*		201	409 (Username already tal
5		/id	update	POST	password, email, address1, address2, city, postalcode, country, firstname, lastname		201	403(forbidden)
6		/id	show	GET	id		201	403 (forbidden)
8								
9	Products	/products	En cours d'implémentation					
10		/	create	POST	(idUser), title, description, dateStart, dateLength(day), startPrice, buyoutPrice, reservePrice	picture	201	400 (paramètres manquan
11		/id	update	POST	à définir	picture	201	403(forbidden)
12		/id	show	GET	none	picture	200	404 (not found)
13		/id	delete	DELETE		picture	200	403(forbidden), 404(not fo
14								
15								
16								

2- Le nombre de fonctionnalités

Nous avons choisi de ne pas développer un grand nombre de fonctionnalité, mais de poser notre base de code et de librairies / modèles qui nous permettront par la suite d'avancer plus rapidement et plus efficacement. Aussi, nous pouvons compter deux grandes fonctionnalités visuelles sur notre application :

- le CRUD des utilisateurs
- le CRUD des produits avec dépendance de donnée aux utilisateurs.
- La recherche de produits

Nous pouvons également compter dans les fonctionnalités développées des 'non visuelles', qui ne sont pas immédiatement jugeable en naviguant sur le site, mais qui permettent de gagner énormément en robustesse et efficacité de code.

Quelques fonctionnalités non visuelles :

- La gestion des tokens utilisateurs de session du côté serveur.
- Simulation des protections CSRF en validant les champs des formulaires avec des listes d'exclusions / d'autorisations

- Séparation du code grâce aux BluePrints de Flask, permettant de créer un fichier par modèle, tout en conservant le partage de donnée entre les classes
- Création de helpers permettant la gestion CRUD générique de tout type d'objet, tout en gérant les associations d'objets.

1. CRUD Utilisateur

Il s'agit d'un CRUD classique, permettant l'enregistrement / modification / connexion et déconnexion des utilisateurs.

Nom collection	Route de base	Ressource (route)	Type	paramètre1	à implémenter	Réponse ok	Réponse ko
	/login	login	POST	username, password		200 401 (Pas d'username trouvé)	
	/logout	logout	POST			201 none	
	/	register	POST	username*, password*, email*, address1*, address2, city*, postalcode*, country*, firstname*, lastname*		201 409 (Username already take)	
	/id	update	POST	password, email, address1, address2, city, postalcode, country, firstname, lastname		201 403(forbidden)	
	/id	show	GET	id		201 403 (forbidden)	

Nous avons également implémenté dans les fonctions Users des helpers permettant de rapidement vérifier des droits utilisateurs sur des modèles génériques. Cela permet en appelant une seule fonction de savoir si l'utilisateur a le droit ou non à une ressource particulière. Cela nous est très pratique, car nous gagnons un temps considérable à ne pas faire du code de validation spécifique sur chaque classe.

```
# Check if the user is authenticated and if he has the right to access ressource.
# If it's not the same user, abort 400
def has_right_abort(resp, user_id=None):
    if not resp or not 'user_id' in resp or not 'token' in resp:
        abort(make_response('Missing Token or User Id',400))
    id = has_right(resp, user_id)
    if id != False:
        return id
    else:
        abort(401)

# Return true if same user and token, otherwise return false
def has_right(resp, user_id):
    if not resp or not 'user_id' in resp or not 'token' in resp:
        return 0
    token = resp['token']
    user = helpers.get_by(glob.users, user_id)
    if int(resp['user_id']) != int(user_id) or resp['token'] != user['token']:
        return False
    else:
        return True
```

2. CRUD des produits

Le CRUD des produits est lui aussi extrêmement classique, et permet de faire toute les actions sur les modèles de données de type "produit".

Products	/products	En cours d'implémentation				
	/	create	POST	(idUser), title, description, dateStart, dateLength(day), startPrice, buyoutPrice, reservePrice	picture	201 400 (paramètres manquants)
	/:id	update	POST	à définir	picture	201 403(forbidden)
	/:id	show	GET	none	picture	200 404 (not found)
	/:id	delete	DELETE		picture	200 403(forbidden), 404(not found)

Il est à noter que nous avons une relation de type *user has_many products*, de ce fait product belongs_to users, donc les products ont une foreign_key stockée permettant d'accéder à l'utilisateur ayant crée ce produit. Cela nous permet de retrouver rapidement la liste de tous les produits mis en vente par un utilisateur donné, mais également de conserver une gestion des droits fines.

3. Gestion des tokens

Nous avons rencontrés quelques difficultés à utiliser les sessions fournies par l'API Flask par défaut, aussi nous avons décidé d'implémenter notre propre système de gestion es tokens, qui sont donc conservés en base de donnée. Nous avons ainsi la possibilité de gérer finement le cryptage des données, et d'éviter un maximum le vol de session.

Cette gestion des tokens nous permet également de pouvoir dans le futur mettre des sécurités de donnée supplémentaire pour l'utilisateur, comme une vérification par habitudes de connections, des connections via des services spécialisés (OAuth, Facebook ...).

```
def generate_token(user):
    user = helpers.get_by(glob.users, user['id'])
    user['token'] = base64.b64encode(str(user['id']) + str(os.urandom(5)) + str(glob.secret_key))
    myjson.save_json(glob.users, users_path)

def destroy_token(user):
    user['token'] = ""
    myjson.save_json(glob.users, users_path)

def get_user_by_id(user_id, token=None):
    user = helpers.get_by(glob.users, user_id)
    if user is None:
        abort(make_response("User not found", 400))
```

4. Simulations des protections basiques contre les attaques CSRF et les mass-assignments

Les attaques CSRF permettent de faire de l'injection de donnée dans des cmeps qui n'existent pas, ou encore de se faire passer une personne différente. Nous avons essayé de limiter ce genre de comportement au maximum en s'inspirant des protections de Ruby On Rails "[Strong Parameters](#)".

```

# Create a new object for the given model
###
# model : the global model to modify
# save_path : The file path for the json model
# allowed_fields : The allowed fields for the model
# mandatory_fields : Check if the mandatory fields are present and not empty
# special_fields : Special behaviour for special fields
# Association : Allow association between 2 models. association = [[association_field : 'field_name', association_entry: entry, association_name: 'name']]
def new_object(model, resp, save_path, allowed_fields, mandatory_fields=None, special_fields=None, associations=None):
    new_model = {}
    if mandatory_fields:
        exclude_field = ['reservePrice', 'startPrice', 'id', 'user_id', 'user']
        null_fields = ['title', 'description', 'buyoutPrice', 'dateLength', 'dateStart']
        product = helpers.update_object(glob.products, product_id, resp, glob.products_path, null_fields=null_fields, exclude_fields=exclude_field)

```

5. Séparation du code grâce aux BluePrints

Les [blueprints](#) permettent de gagner énormément en flexibilité au niveau du code, en permettant de séparer son code proprement en modules, en enregistrant des hooks permettant de faire des callbacks très rapidement dans le code, et également en préfixant une ou plusieurs routes facilement.

```

# Blueprint allow to do "multi-file", by registering collections
app.register_blueprint(products.pdt, url_prefix="/products")
app.register_blueprint(users.usr, url_prefix="/user")

```

6. Création de helpers

Les helpers sont accessibles de partout dans le code et nous permettent de faciliter la gestion des CRUD sur les objets.

Ainsi, dans notre class Helpers, nous avons les fonctions Add / Delete / Update et Show, qui sont génériques à tout type de donnée.

```

# Update the json object and save it in database.
###
# model : the global model to modify
# save_path : The file path for the json model
# value_model : the model_entry to modify
# exclude_fields : Exclude fields, even if they are in the form
# null_fields : Don't set fields if they are empty
# mandatory_fields : Check if all fields are presents
def update_object(model, value_model, resp, save_path, exclude_fields=None, null_fields=None, mandatory_fields=None):
    model_entry = get_by(model, value_model, test="update_object")
    print model_entry
    if model_entry: # If the model_entry exist
        new_model = {}
        if mandatory_fields:
            verify_mandatory_field_form(mandatory_fields, resp)

```

7. La recherche de produits

Cette fonctionnalité n'a pas été implémentée sur l'interface graphique, mais il est cependant possible de chercher un produit avec un ou des mots clés, grâce à la méthode HTTP POST /products/search et en envoyant en paramètre {'query':'my awesome query'}.

The screenshot displays a REST client interface with the following components:

- Environment:** Normal, Basic Auth, Digest Auth, OAuth 1.0, No environment.
- URL:** localhost:5000/products/search
- Method:** POST
- Form:** x-www-form-urlencoded (selected). Key: query, Value: seche cheveux.
- Buttons:** Send, Preview, Add to collection, Reset.
- Response:** STATUS 200 OK, TIME 47 ms.
- Body:** JSON response showing a list of products.

```
1 {
2   "products": [
3     {
4       "buyoutPrice": "",
5       "dateLength": "1",
6       "dateStart": "20141112",
7       "description": "super seche cheveux",
8       "id": 2,
9       "imageUrl": "",
10      "reservePrice": "",
11      "startPrice": "123",
12      "title": "seche cheveux",
13      "user_id": 1
14    },
15    {
16      "buyoutPrice": "",
17      "dateLength": "1",
18      "dateStart": "20141112",
19      "description": "super seche cheveux",
20      "id": 3,
21      "imageUrl": "",
22      "reservePrice": "1",
23      "startPrice": "1",
24      "title": "seche cheveux",
25      "user_id": "1"
26    }
27  ]
28 }
```

Comment exécuter notre projet

Pour exécuter notre projet, il suffit simplement de taper dans un terminal

```
$> python flask/app.py
```

```
$> node ./scripts/web-server.js
```

Une fois le serveur lancé, tout est prêt pour consommer l'API et utiliser notre interface graphique.