Projet Conception, Projet Synthèse

The Auction Zone

Par Nathaelle Fournier et Quoc Huan Tran

420-C61-IN PROJET SYNTHÈSE

Présenté à Jean-Christophe Demers

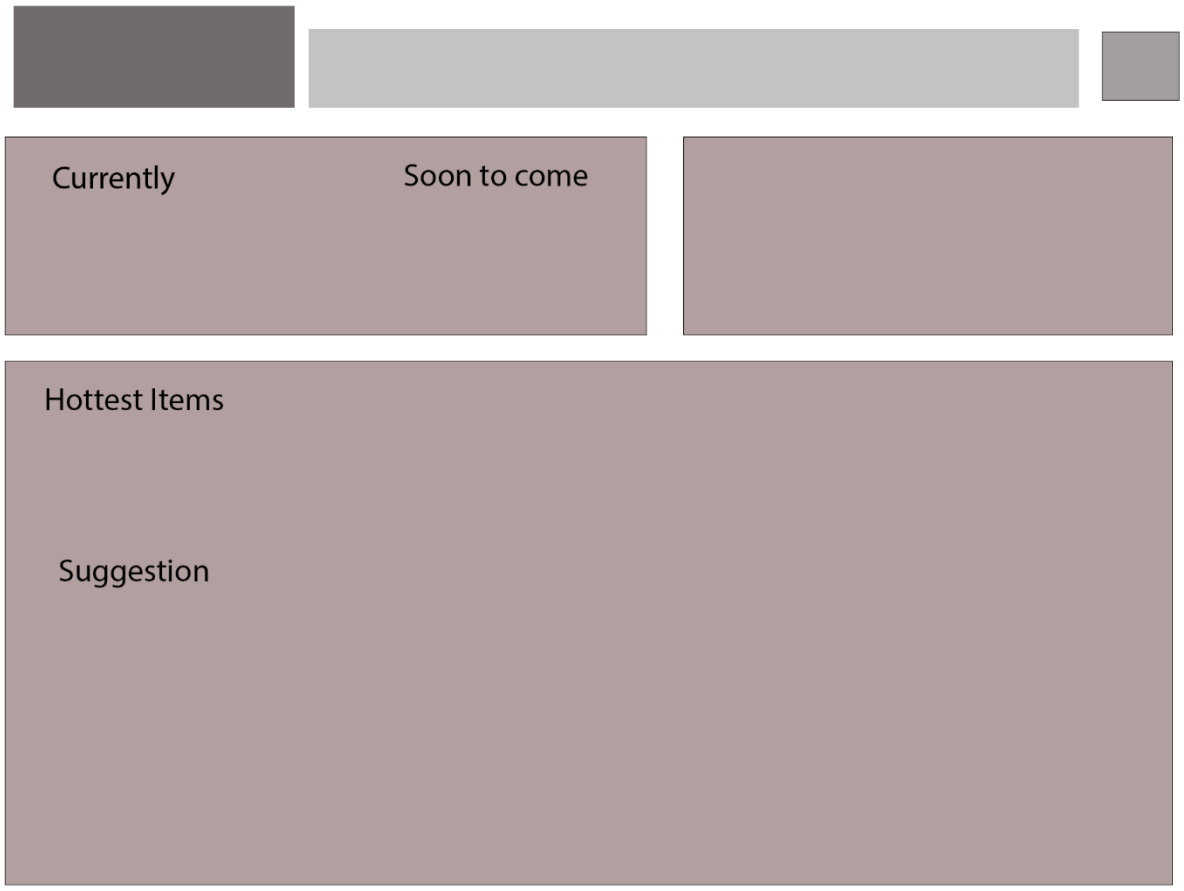
Technique de l'informatique

Cégep du Vieux Montréal

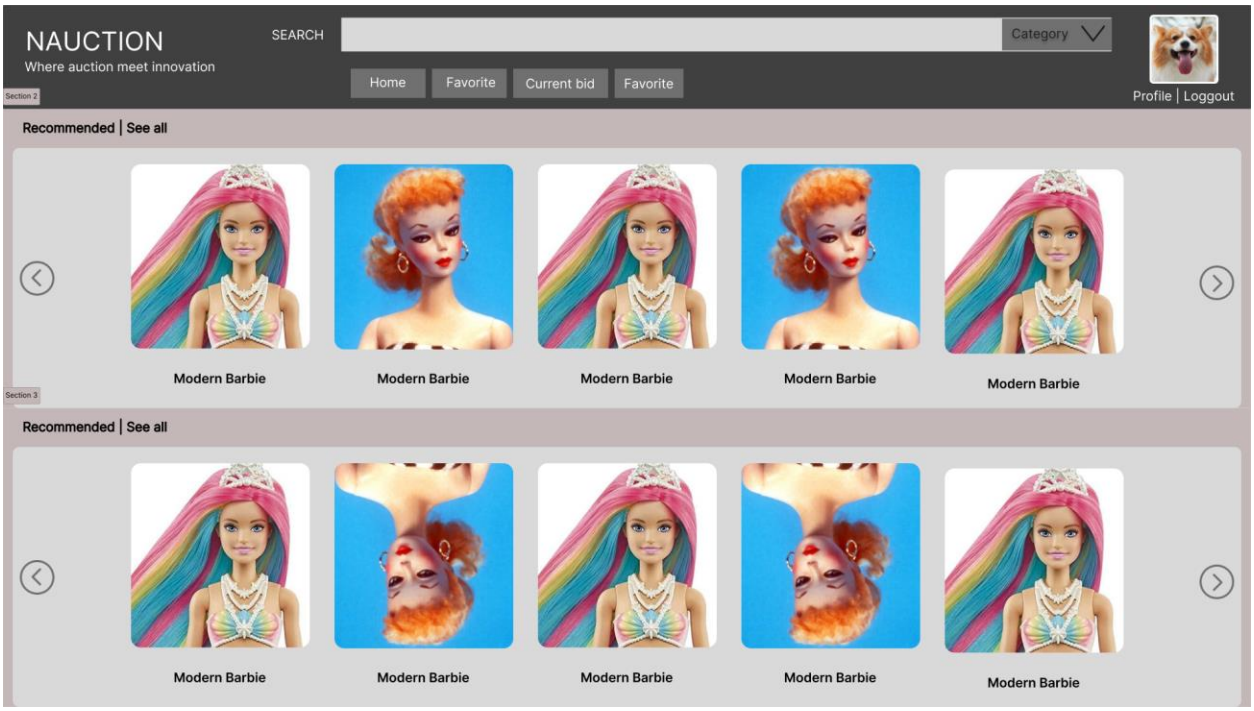
2 mars 2023

Table des matières

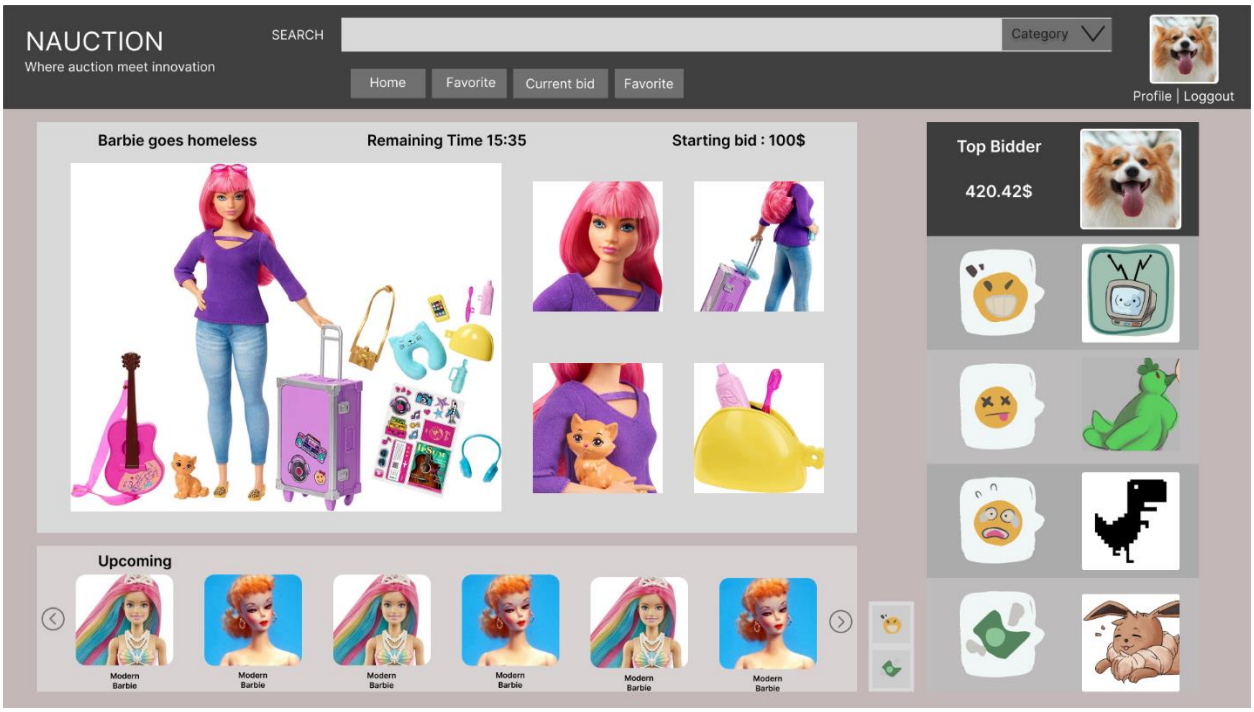
Maquettes



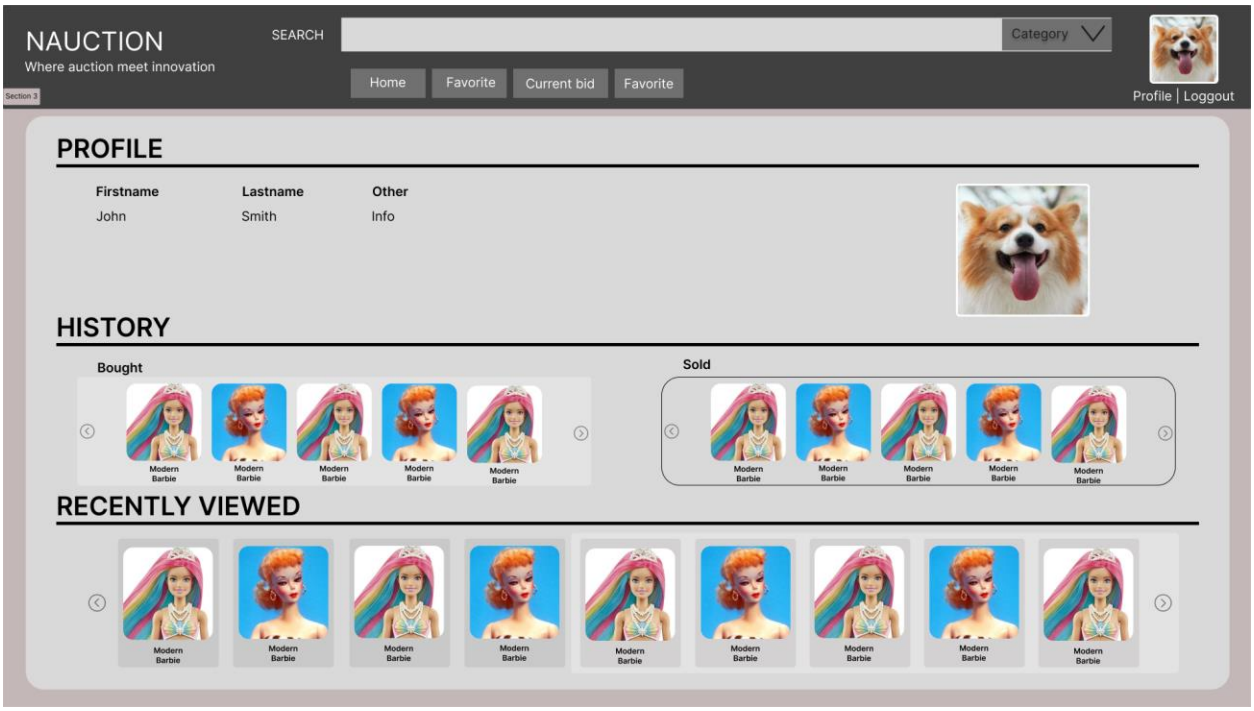
Maquette prototype



Maquette de la page principale

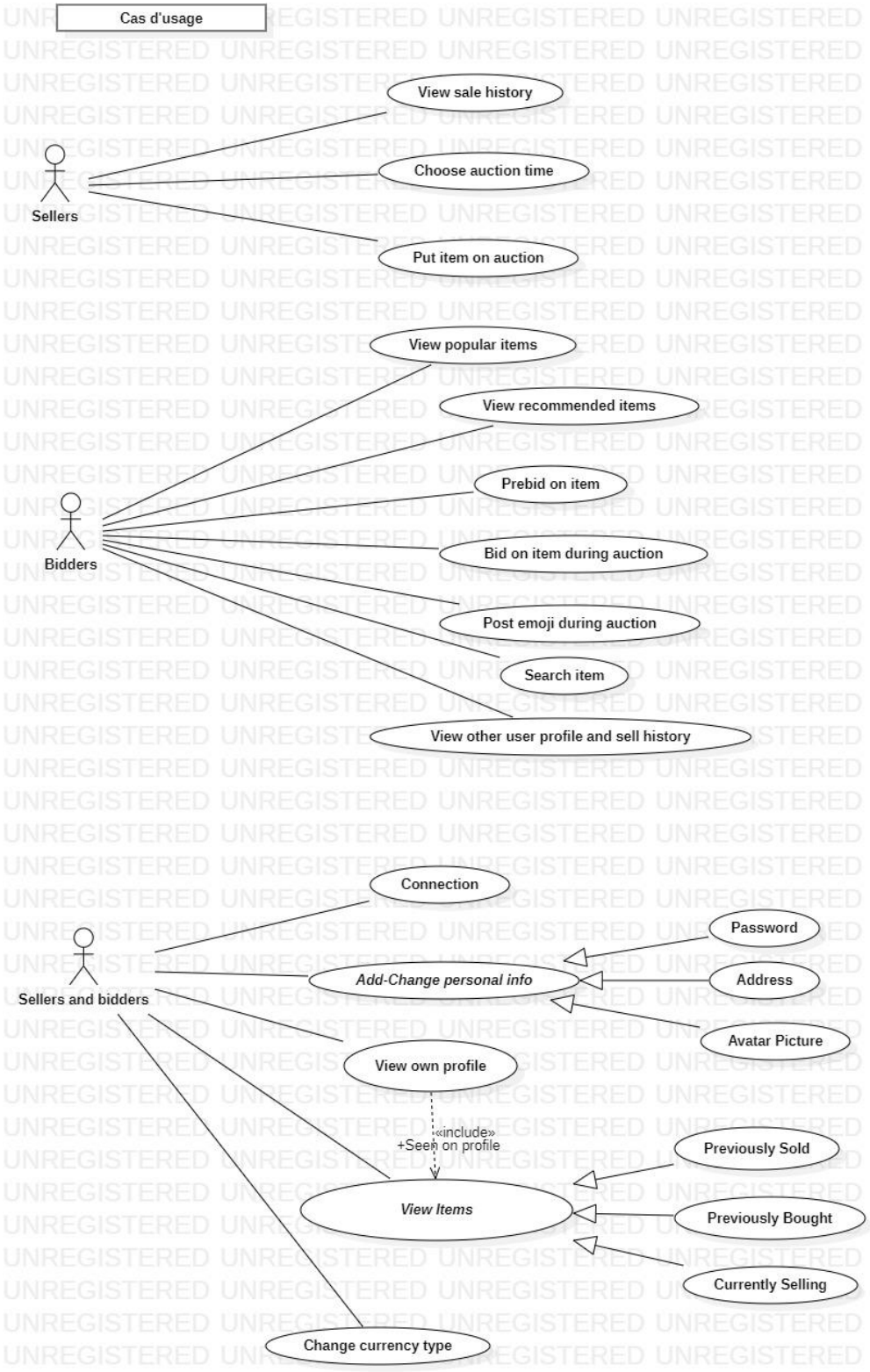


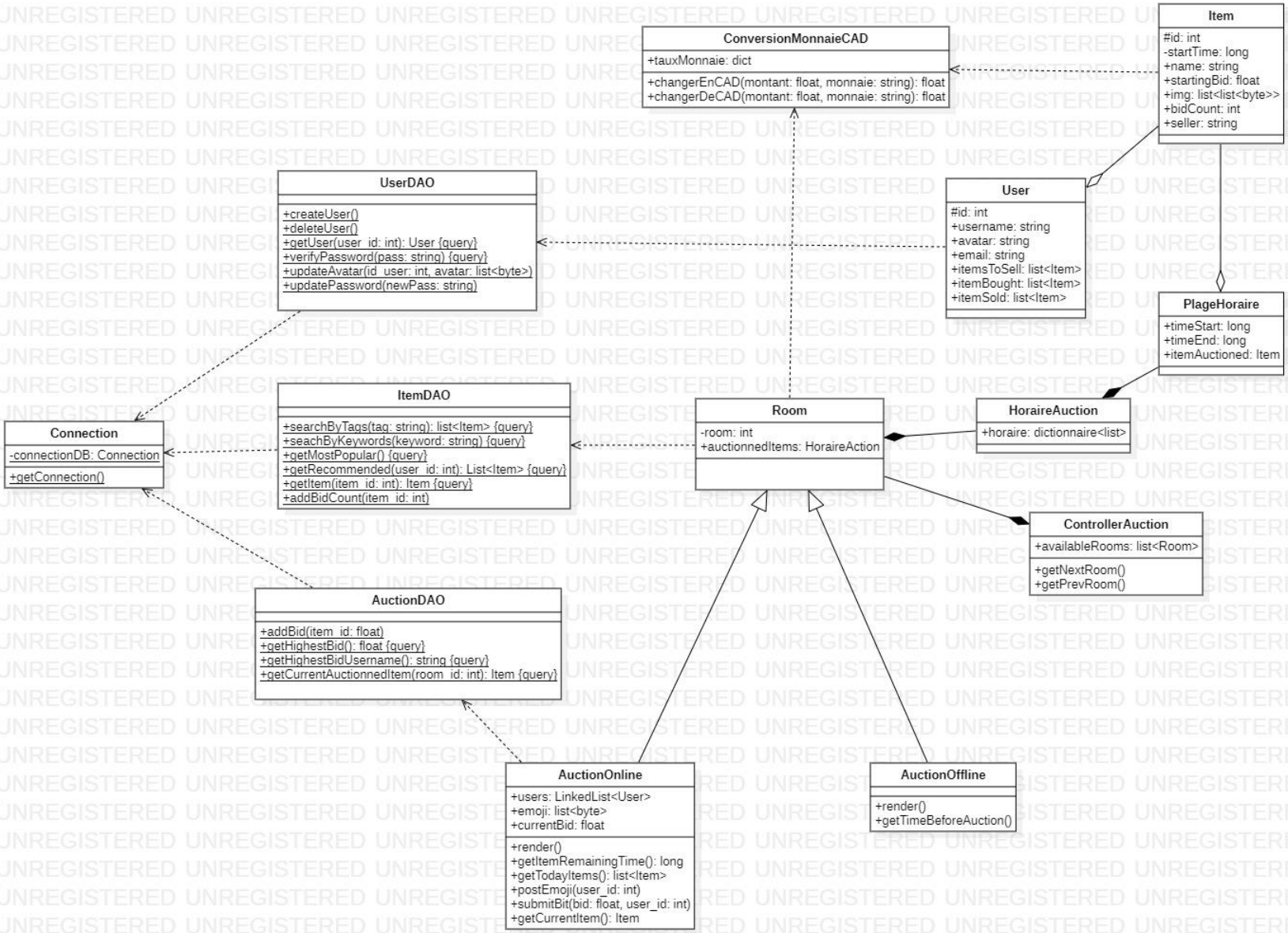
Maquette de la salle d’encan et du chat



Maquette de la page de profile et quatre prototypes de section d’items

Conception UML




ConversionMonnaieCAD
+tauxMonnaie: dict
+changerEnCAD(montant: float, monnaie: string): float
+changerDeCAD(montant: float, monnaie: string): float
Item
#id: int
-startTime: long
+name: string
+startingBid: float
+img: list<list<byte>>
+bidCount: int
+seller: string
UserDAO
+createUser()
+deleteUser()
+getUser(user_id: int): User {query}
+verifyPassword(pass: string) {query}
+updateAvatar(id_user: int, avatar: list<byte>)
+updatePassword(newPass: string)
User
#id: int
+username: string
+avatar: string
+email: string
+itemsToSell: list<Item>
+itemBought: list<Item>
+itemSold: list<Item>
PlageHoraire
+timeStart: long
+timeEnd: long
+itemAuctioned: Item
ItemDAO
+searchByTags(tag: string): list<Item> {query}
+seachByKeywords(keyword: string) {query}
+getMostPopular() {query}
+getRecommended(user_id: int): List<Item> {query}
+getItem(item_id: int): Item {query}
+addBidCount(item_id: int)
Room
-room: int
+auctionnedItems: HoraireAction
HoraireAuction
+horaire: dictionnaire<list>
Connection
-connectionDB: Connection
+getConnection()
ControllerAuction
+availableRooms: list<Room>
+getNextRoom()
+getPrevRoom()
AuctionDAO
+addBid(item_id: float)
+getHighestBid(): float {query}
+getHighestBidUsername(): string {query}
+getCurrentAuctionnedItem(room_id: int): Item {query}
AuctionOnline
+users: LinkedList<User>
+emoji: list<byte>
+currentBid: float
+render()
+getItemRemainingTime(): long
+getTodayItems(): list<Item>
+postEmoji(user_id: int)
+submitBit(bid: float, user_id: int)
+getCurrentItem(): Item
AuctionOffline
+render()
+getTimeBeforeAuction()

Structure de données externes

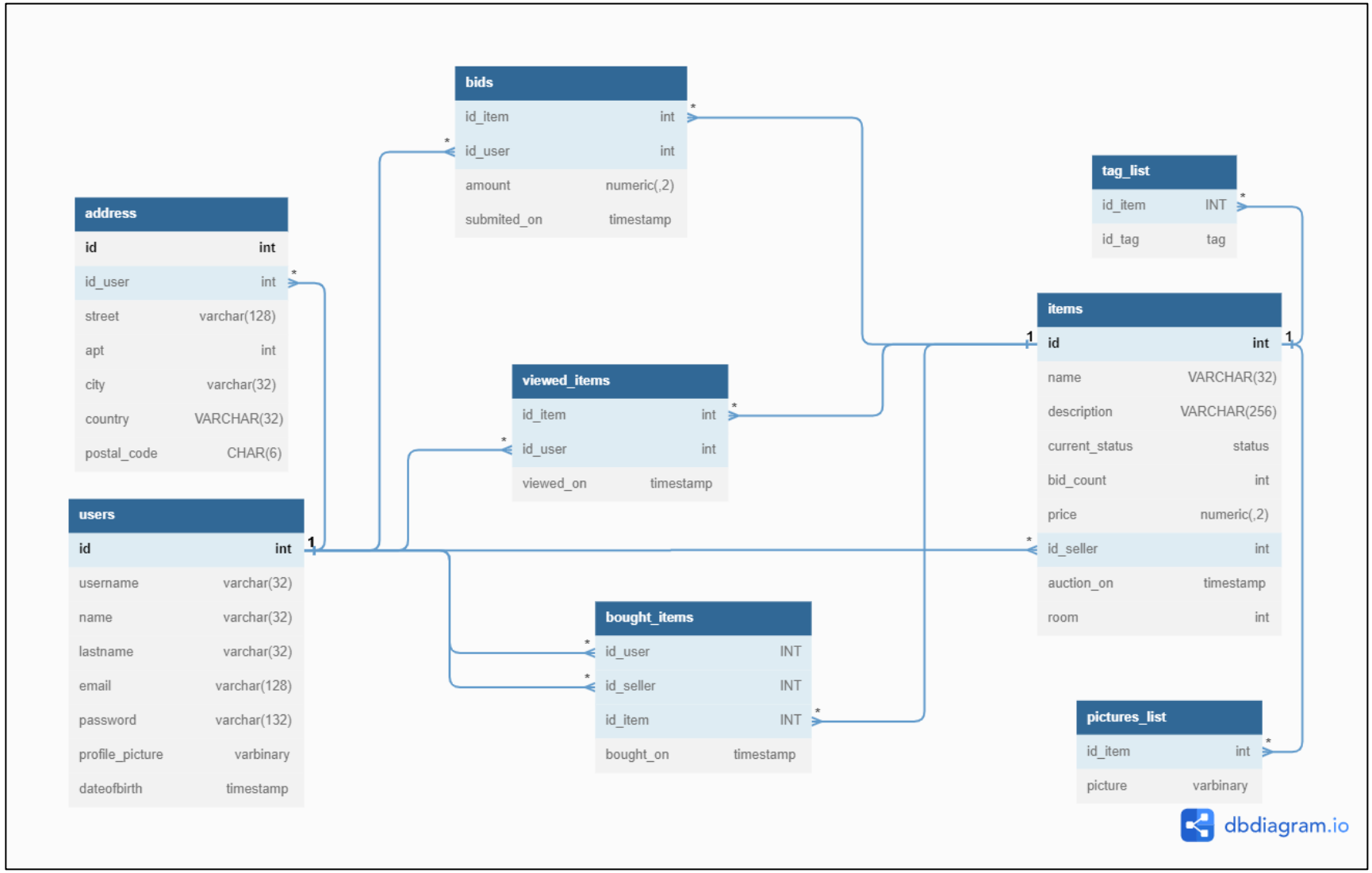
Nous utiliserons une base de données basée sur PostgreSQL. Voici la première ébauche de code pour celle-ci.

```
CREATE TYPE "status" AS ENUM (
  'waiting',
  'in_auction',
  'sold'
);

CREATE TYPE "tag" AS ENUM (
  'Antique',
  'Art & Sculture',
  'Automobile',
  'Bijoux & Accessoire',
  'Collection',
  'Livre & Manuscrit',
  'Meuble',
  'Monnaie',
  'Musique',
  'Sport',
  'Vaisselle & Coutellerie',
  'Vêtement'
);

CREATE TABLE "users" (
  "id" INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  "username" varchar(32) UNIQUE NOT NULL,
  "name" varchar(32),
  "lastname" varchar(32),
  "email" varchar(128) UNIQUE NOT NULL,
  "password" varchar(132) NOT NULL,
  "profile_picture" varbinary,
  "dateofbirth" timestamp NOT NULL
);
```

```
CREATE TABLE "address" (
  "id" INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  "id_user" int NOT NULL,
  "street" varchar(128) NOT NULL,
  "apt" int DEFAULT null,
  "city" varchar(32) NOT NULL,
  "country" VARCHAR(32) NOT NULL,
  "postal_code" CHAR(6) NOT NULL
);

CREATE TABLE "items" (
  "id" INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  "name" VARCHAR(32) NOT NULL,
  "description" VARCHAR(256) DEFAULT '',
  "current_status" status NOT NULL,
  "bid_count" int DEFAULT 0,
  "price" numeric(,2) DEFAULT 1,
  "id_seller" int NOT NULL,
  "auction_on" timestamp UNIQUE NOT NULL,
  "room" int
);

CREATE TABLE "bought_items" (
  "id_user" INT NOT NULL,
  "id_seller" INT NOT NULL,
  "id_item" INT UNIQUE NOT NULL,
  "bought_on" timestamp
);

CREATE TABLE "tag_list" (
  "id_item" INT,
  "id_tag" tag
);

CREATE TABLE "pictures_list" (
  "id_item" int,
  "picture" varbinary
);
```

```
CREATE TABLE "bids" (
  "id_item" int NOT NULL,
  "id_user" int NOT NULL,
  "amount" numeric(,2) NOT NULL,
  "submited_on" timestamp NOT NULL
);

CREATE TABLE "viewed_items" (
  "id_item" int NOT NULL,
  "id_user" int NOT NULL,
  "viewed_on" timestamp
);
```

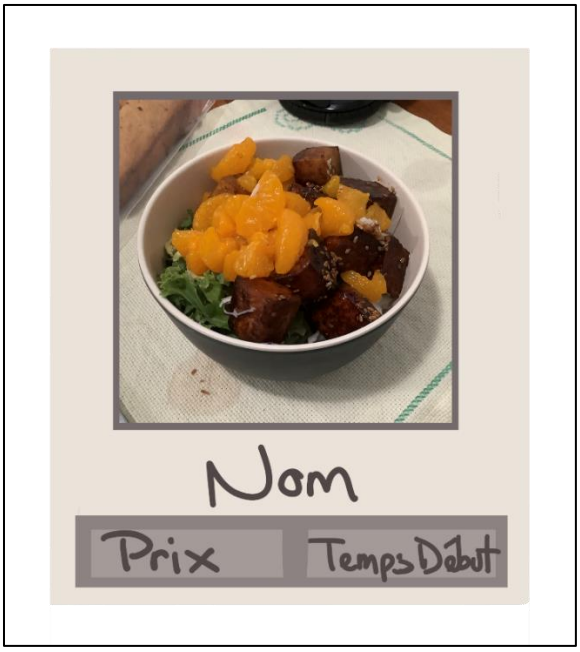
```
ALTER TABLE "address" ADD FOREIGN KEY ("id_user") REFERENCES "users" ("id");

ALTER TABLE "tag_list" ADD FOREIGN KEY ("id_item") REFERENCES "items" ("id");

ALTER TABLE "bids" ADD FOREIGN KEY ("id_item") REFERENCES "items" ("id");

ALTER TABLE "bids" ADD FOREIGN KEY ("id_user") REFERENCES "users" ("id");

ALTER TABLE "items" ADD FOREIGN KEY ("id_seller") REFERENCES "users" ("id");

ALTER TABLE "pictures_list" ADD FOREIGN KEY ("id_item") REFERENCES "items"
("id");

ALTER TABLE "bought_items" ADD FOREIGN KEY ("id_seller") REFERENCES "users"
("id");

ALTER TABLE "bought_items" ADD FOREIGN KEY ("id_user") REFERENCES "users"
("id");

ALTER TABLE "bought_items" ADD FOREIGN KEY ("id_item") REFERENCES "items"
("id");

ALTER TABLE "viewed_items" ADD FOREIGN KEY ("id_item") REFERENCES "items"
("id");

ALTER TABLE "viewed_items" ADD FOREIGN KEY ("id_user") REFERENCES "users"
("id");
```


bids
id_item int
id_user int
amount numeric(,2)
submited_on timestamp
address
id int
id_user int
street varchar(128)
apt int
city varchar(32)
country VARCHAR(32)
postal_code CHAR(6)
tag_list
id_item INT
id_tag tag
items
id int
name VARCHAR(32)
description VARCHAR(256)
current_status status
bid_count int
price numeric(,2)
id_seller int
auction_on timestamp
room int
viewed_items
id_item int
id_user int
viewed_on timestamp
users
id int
username varchar(32)
name varchar(32)
lastname varchar(32)
email varchar(128)
password varchar(132)
profile_picture varbinary
dateofbirth timestamp
bought_items
id_user INT
id_seller INT
id_item INT
bought_on timestamp
pictures_list
id_item int
picture varbinary
dbdiagram.io

Contraintes & Éléments de Conception

Interface utilisateur

Les contraintes pour l'interface utilisateur seront toutes remplies avec les étiquettes des objets en vente. Ceux-ci auront leur titre, au moins une image et leur prix de départ. Le prix devra être formaté selon le choix de monnaie de l'utilisateur.



Exemple d'étiquette

Sinon, on retrouve du texte dans la page de profile contenant les informations personnelles de l'usager ainsi qu'une image pour avatar. Dans l'encan même, une sélection d'émojis sous forme d'images seront mis à la disposition des enchérisseurs pour maintenir l'intérêt de ceux-ci et rendre l'expérience plus amusant.

Structure de données

Nous allons utiliser des **tableaux** pour stocker les données qui n'ont pas besoin d'être modifier régulièrement. Ils seront utilisés pour regrouper les données qui n'ont pas restriction de sélection, par exemple, parcourir la liste des plages horaires d'une journée. Ils seront aussi utilisés pour les regroupements de données qui ont la possibilité d'être modifier comme la mise soumise par un tel usager.

Nous pensons utiliser un **dictionnaire** pour identifier les différents taux de change ainsi que le symbole monétaire correspondant à chacun. Les usagers pourront donc utiliser la monnaie de leur choix dans notre site web. Comme nous voulons enregistrer les prix en montant d'argent canadien dans la BD, nous devons convertir le prix lors de l'affichage à l'écran et y mettre son symbole monétaire correspondant. Par ailleurs, un utilisateur qui n'utilise pas la monnaie par défaut pourra tout de même entrer un montant avec sa devise de préférence et ce montant sera convertit avant d'être insérer dans la base de données comme un montant canadien.

Pour la structure de données que nous pensions implémenter en entier, nous avons décidé d'aller avec une **liste chainée** pour stocker les références des usagers présents dans une salle d'encan. Nous avons choisi cette structure de données, car les usagers pourront interchanger de place à tout moment et ce indépendamment d'où il se retrouve dans l'ordre de la liste. C'est cette structure qui contiendra les listes d'usagers et des emojis qu'ils ont envoyés.

Patrons de conception

Les patrons de conception que nous avons choisi de spécifiquement implémenter dans notre projet sont *Observer*, *Strategy* et *State*.

Observer

Nous avons choisi *Observer*, car il s'agit d'un patron évident pour tous projets web qui demande quelconques interactions de l'utilisateur. Il sera implémenté, dans la majorité des cas, pour appeler une fonction à l'appui d'un bouton ou de générer une nouvelle composante *React* à l'écran. Par exemple, appuyer sur un bouton pour appeler la fonction de recherche ou appuyer sur un autre bouton pour ouvrir la fenêtre pour modifier son mot de passe. (Shvets A. , Observer, 2023)

Strategy

Nous pensions inclure le patron de stratégie, puisqu'il s'implémente naturellement dès que l'on utilise le polymorphisme. Nous pensions inclure deux méthodes de recherches qui retourneraient une liste d'objet à vendre qui correspond aux paramètres envoyés. Les usagers pourront soit faire une recherche par mots clés ou par tags. (Shvets A. , Strategy, 2023)

State

Nous pensons utiliser le patron du *State* pour la page où l'on retrouverait les encans. Puisque ceux-ci ne seraient qu'accessible que lorsqu'il y en a un cours. Advenant le cas où il n'y en aurait pas, seul un horaire de ceux à venir serait afficher. Ainsi, dépendamment de l'état de la page, nous allons générer différentes composantes *React*. Ainsi, nous aurons juste à stocker les composantes appropriées dans un objet représentant l'état de la page souhaitée. Ainsi, lorsque l'on vérifie la condition pour connaitre l'état de la page, nous avons qu'à appeler l'objet qui convient au scénario. De cette manière, il sera facile d'implémenter un nouvel état et de l'associer à une nouvelle condition. (Shvets, 2023) (Wikipedia, 2023)

Expressions régulières

Nous pensons utiliser les expressions pour la recherche d'article en particulier. Sois l'usager entre quelque chose dans la barre de recherche et on retourne une liste des articles dont le nom correspond aux données reçues. Donc si l'on entre ‘’école‘’, nous allons retourner tous les items où le nom convient à ‘’/%école%/’’ sans qu'il soit sensible à la case et aux accents. (Albe, 2023) De plus, il faudra aussi vérifier si un mot de passe correspond aux critères exigés, soit la longueur de celui-ci et si l'on retrouve un caractère spécial dans celui-ci. Le *régex* qui conviendrait à ces critères serait similaires à ‘’`/\w{6,}/&&/\W+/` ‘’

Algorithme

L'algorithme de recommandation qui se tient sur les tags des items visités, ainsi que ceux acheté précédemment par cet usager. Donc lorsque les calculs de score sont effectué, nous allons implanter notre propre algorithme de trie de données. Nous avons décidé d'implanter dans notre projet le quick-sort. Cet algorithme rapide et en moyenne devrait nous retourner le résultat en ordre de O(n log n). L'algorithme utilise principalement des récursion avec un principe de pivot sur le début et la fin de la liste envoyée. (Tri rapide, 2023) (QuickSort, 2023)

Mathématique

Pour calculer la mise minimale permise sur un article au courant de l'encan

$$mise\ minimal = \ plus\ haut\ bid\ présent * \ pourcentage\ minimal$$

Pour calculer les montant d'argent avec les taux de change

- Pour l'affichage de l'utilisateur avec une monnaie étrangère sélectionnée

$$montant_{étrangé} = \ montant_{canadian} * \ taux\ conversion_{étrangé}$$

- Pour l'enregistrement dans la base de données provenant d'une monnaie étrangère

$$montant_{canadian} = \frac{montant_{étrangé}}{taux\ conversion_{étrangé}}$$

Pour calculer l'importance attribué à **un tag** d'un usagé, on prend en considération les items que l'utilisateur a précédemment visités et achetés.

- Calcule par rapport aux pages visités
  On ne prend que l'historique des pages visités jusqu'à une certaine date

$$score_{visite} = \frac{nbr\ pages\ visités\ avec\ tag}{nbr\ pages\ visités}$$

- Calcul par rapport aux articles achetés
  On considère toutefois que plus qu'il y a de temps qui a passé depuis l'achat de l'article, moins qu'il est pertinent

$$score_{achat} = constante * \sum_{item\_bought} 1 - \left(\frac{time_{current} - \ time_{bought}}{time_{current}}\right)$$

Selon l'action que l'utilisateur a fait par rapport à l'item, on y attribue un **facteur d'importance**. On considère qu'un item que l'utilisateur aurait acheté correspond mieux aux champs d'intérêt de celui-ci qu'un item qu'il aurait seulement visité la page.

$$facteur_{achat} \in [0.5,\ 1]$$

$$score_{final} = score_{achat} * facteur_{achat} + score_{visite} * (1 - facteur_{achat})$$

Veille Technologique

Nous avons choisi d'effectuer nos recherches sur *React* dans le cadre du cours de *Veille Technologique.* C'est pour la réutilisation de composantes graphiques partout à travers les différentes pages du projet, ainsi que pouvoir les mettre à jour à tous moments grâce au *Virtual DOM*.

Références

Albe, L. (2023, 02 24). *ICU Collations against GLIBC 2.28 Data Corruption*. Récupéré sur CYBERTEC: https://www.cybertec-postgresql.com/en/icu-collations-against-glibc-2-28-data-corruption/

QuickSort. (2023, 03 01). Récupéré sur GeekForGeek: https://www.geeksforgeeks.org/quick-sort/

Shvets. (2023, 02 22). *State*. Récupéré sur Refactoring Guru: https://refactoring.guru/design-patterns/state

Shvets, A. (2023, 02 22). *Observer*. Récupéré sur Refactoring Guru: https://refactoring.guru/design-patterns/observer

Shvets, A. (2023, 02 22). *Strategy*. Récupéré sur Refactoring Guru: https://refactoring.guru/design-patterns/strategy

Tri rapide. (2023, 03 01). Récupéré sur Wikipedia: https://fr.wikipedia.org/wiki/Tri_rapide

Wikipedia. (2023, 02 22). *Finite-state Machine*. Récupéré sur Wikipedia: https://en.wikipedia.org/wiki/Finite-state_machine