Projet Conception, Projet Synthèse

*NAuction*

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 3](#_Toc127454720)

[Conception UML 4](#_Toc127454721)

[Structure de données externes 6](#_Toc127454722)

[Contraintes & Éléments de Conception 10](#_Toc127454723)

[Interface utilisateur 10](#_Toc127454724)

[Structure de données 10](#_Toc127454725)

[Patrons de conception 11](#_Toc127454726)

[Expressions régulières 12](#_Toc127454727)

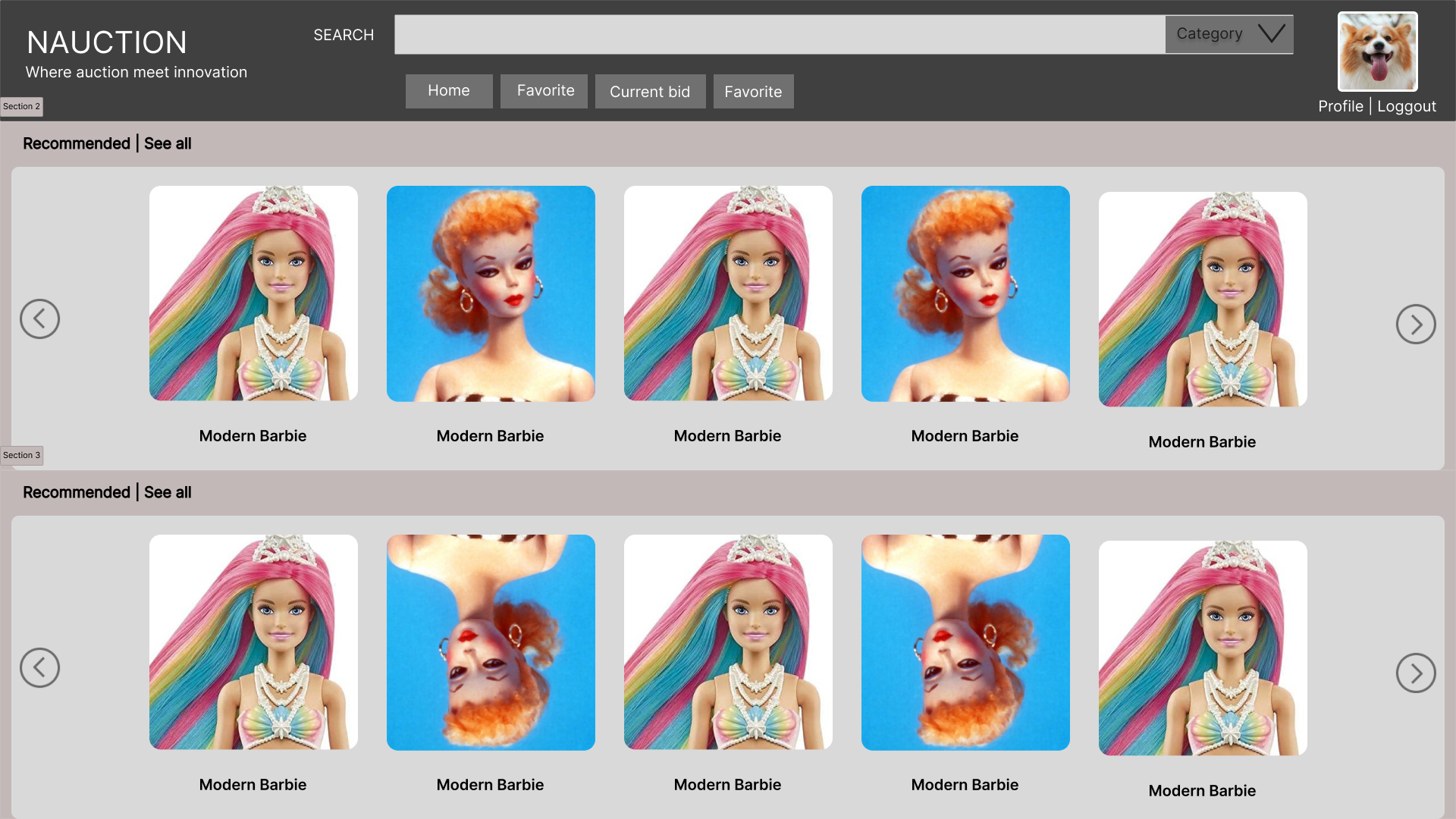
[Algorithme 12](#_Toc127454728)

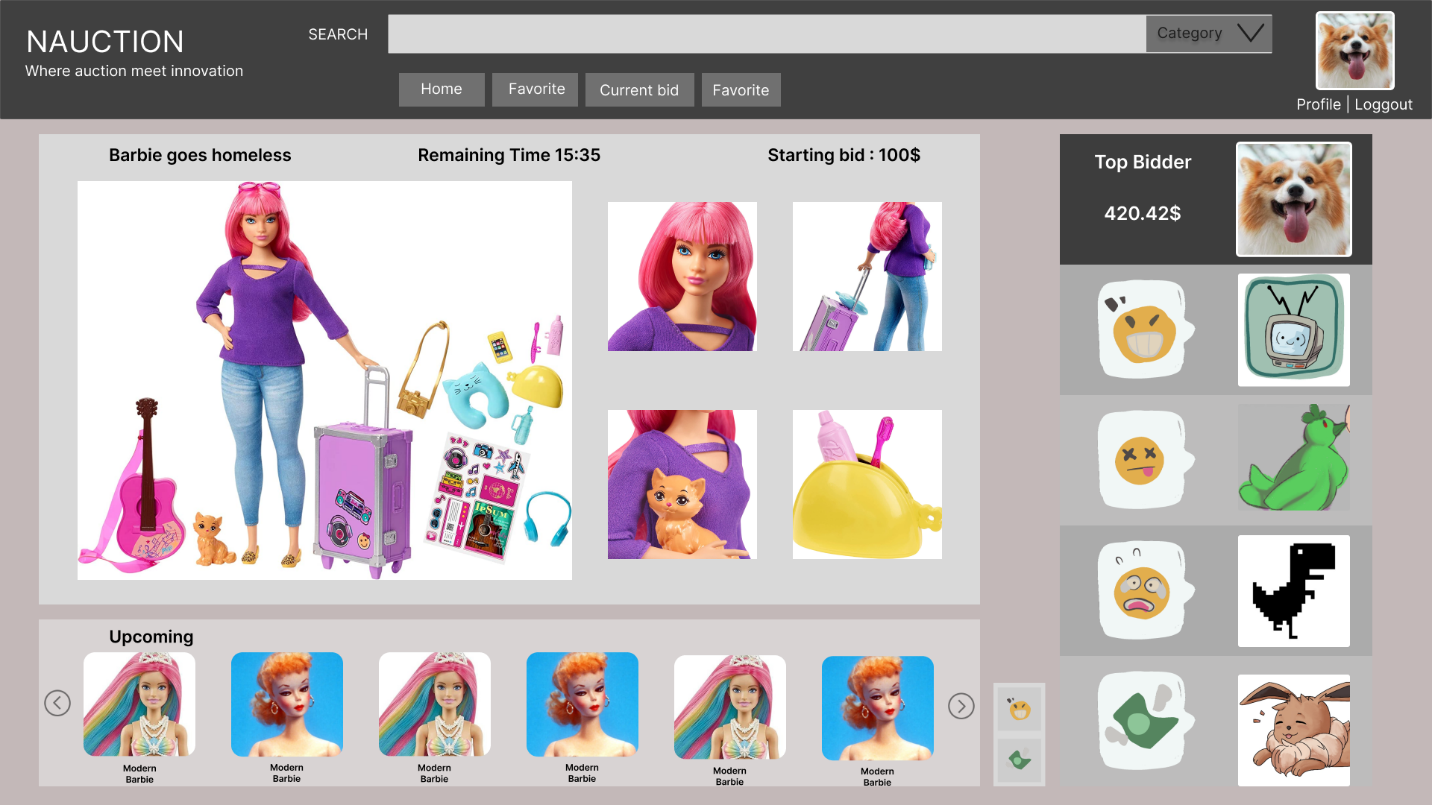
[Mathématique 12](#_Toc127454729)

[Veille Technologique 12](#_Toc127454730)

# Maquettes



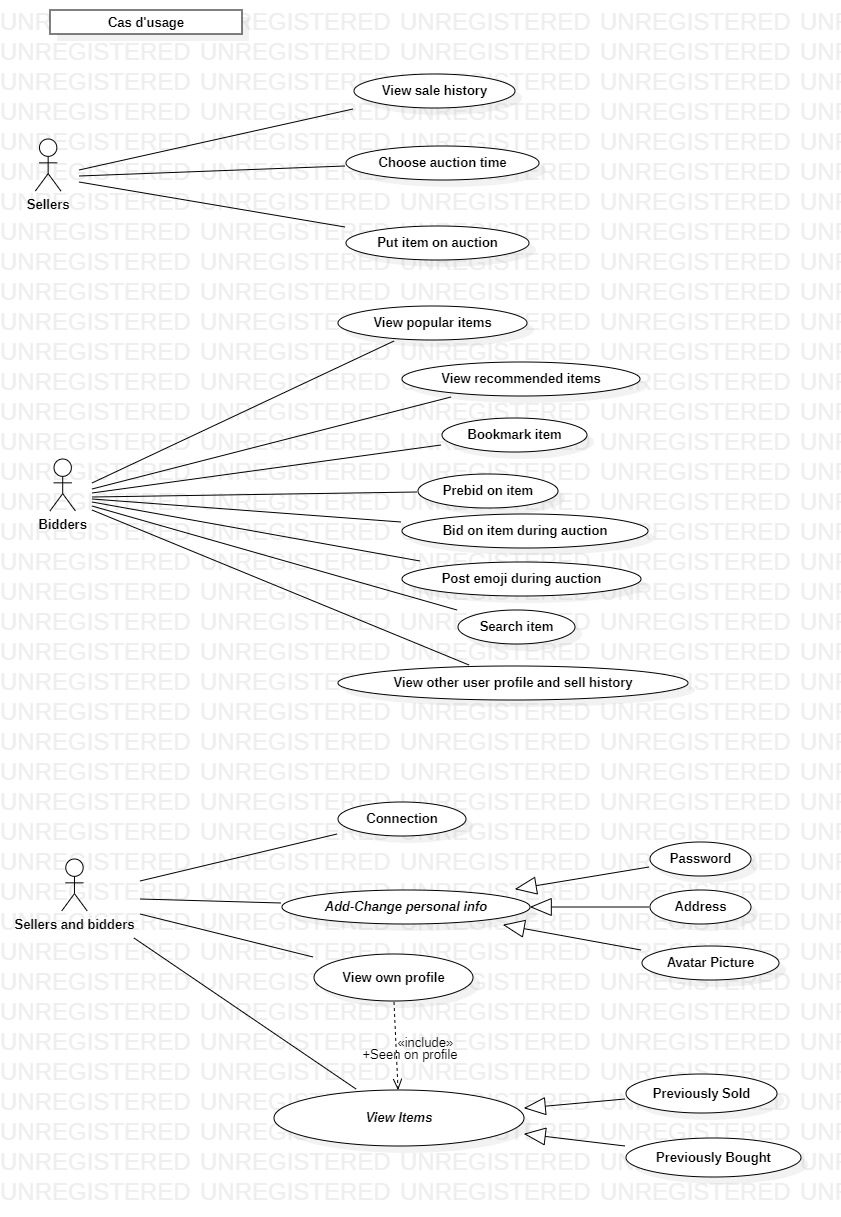




Une image contenant texte

Description générée automatiquement

# Conception UML



# 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,

"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,

"min\_increase" numeric(5,2) 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 "bids" (

"id\_item" int NOT NULL,

"id\_user" int NOT NULL,

"amount" numeric(,2) NOT NULL,

"submited\_on" timestamp NOT NULL

);

COMMENT ON COLUMN "items"."room" IS 'auction\_on & room must be unique';

COMMENT ON COLUMN "bids"."amount" IS 'must be higher then precious price with the same id\_item';

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");

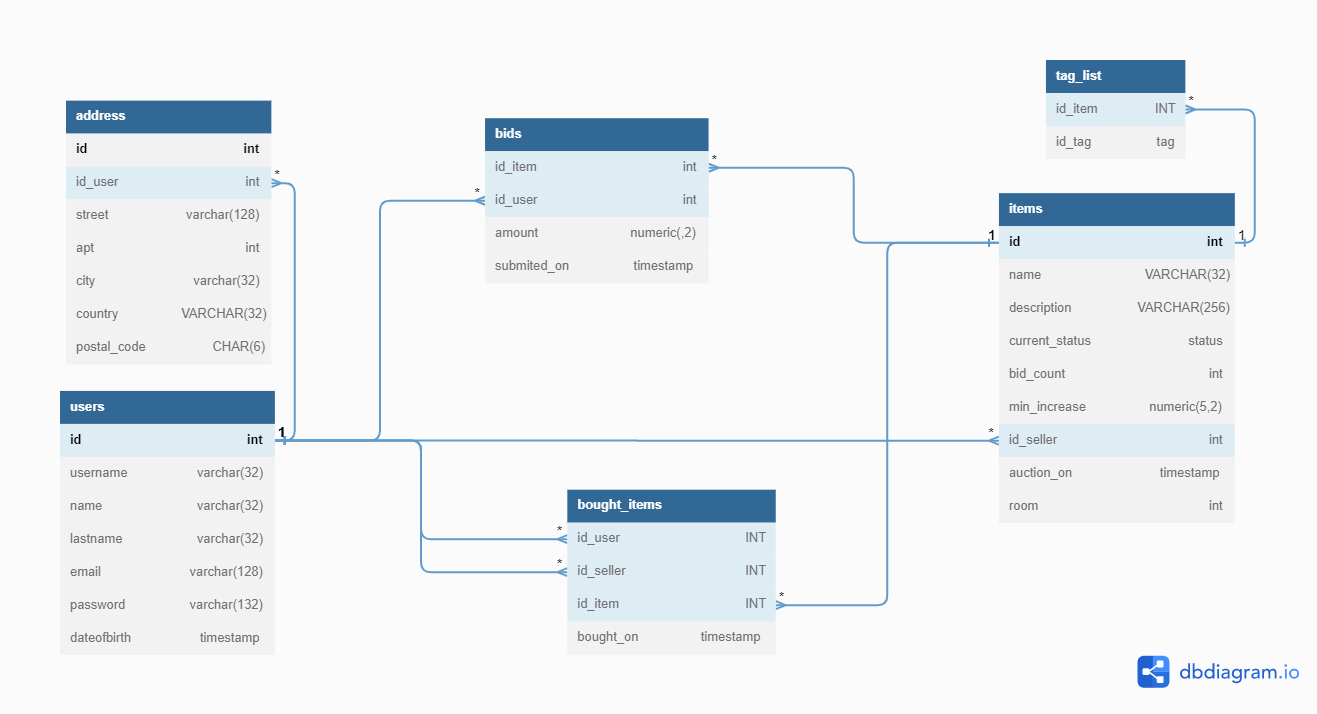
ALTER TABLE "bids" ADD FOREIGN KEY ("id\_user") REFERENCES "users" ("id");

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

ALTER TABLE "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");



# 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 standard monétaire canadien.

Une image contenant texte, alimentation, différent, plat

Description générée automatiquement

Exemple d’étiquette

## 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 que l’on doit parcourir en ordre, 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 regrouper les éléments à un utilisateur.

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)

## Algorithme

Algorithme de recommandation qui se tient sur les tags des items visités, ainsi que ceux acheté précédemment par cet usager.

## Mathématique

Pour calculer un minimum de mise automatique se basant sur le prix courant de l’item

## Veille Technologique

*React* pour la réutilisation de composantes graphiques, ainsi que pouvoir les modifier 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/

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

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