ART.SPACE

DOCUMENT TECHNIQUE ART.SPACE

JULIEN MARCILIAC

Table des matières

Préambule :

- Connexion à l'espace administrateur : user=admin password = TEaS6T1fZGUCSHoCkb5T

- Connexion avec un client déjà inscrit : user=julienM password= 456258

Méthodologie de travail :

- J'ai tout d'abord cherché les exigences fonctionnelles générales du site web.
- J'ai pu créer les diagrammes des cas d'utilisation afin d'identifier les acteurs et les fonctionnalités attendues.
- J'ai énuméré les fonctionnalités attendues et les pages demandées dans un tableau.
- En détaillant le scénario de chaque fonctionnalité, j'ai déterminé les objets manipulés, les méthodes et attributs des objets, les associations entre objet.
- Avec ces informations, j'ai alors construit le diagramme de classes de mon programme et le schéma de la base de données.

J'ai utilisé cette méthodologie de manière itérative en programmant le site web.

Exigences fonctionnelles générales :

Exigences fonctionnelles pour l'internaute :

Recherche :
L'internaute pourra trouver un produit dans l'ensemble du catalogue.

Découverte :
Chaque produit sera classé par rubrique.
Chaque produit sera présenté en détail sur sa propre page.
On y trouvera :

- Son prix
- Ses détails

Sélection :
Lorsque l'internaute est intéressé par un produit, il peut l'enregistrer dans un panier virtuel.
Il doit pouvoir à tout moment ajouter ou supprimer un produit.

Commande :
L'internaute est invité à se connecter ou s'inscrire pour valider sa commande.
Le client devra ensuite pouvoir consulter l'historique de ses commandes.

Mise à jour des données :
L'internaute pourra mettre à jour ses coordonnées.

Exigences fonctionnelles pour l'administrateur :

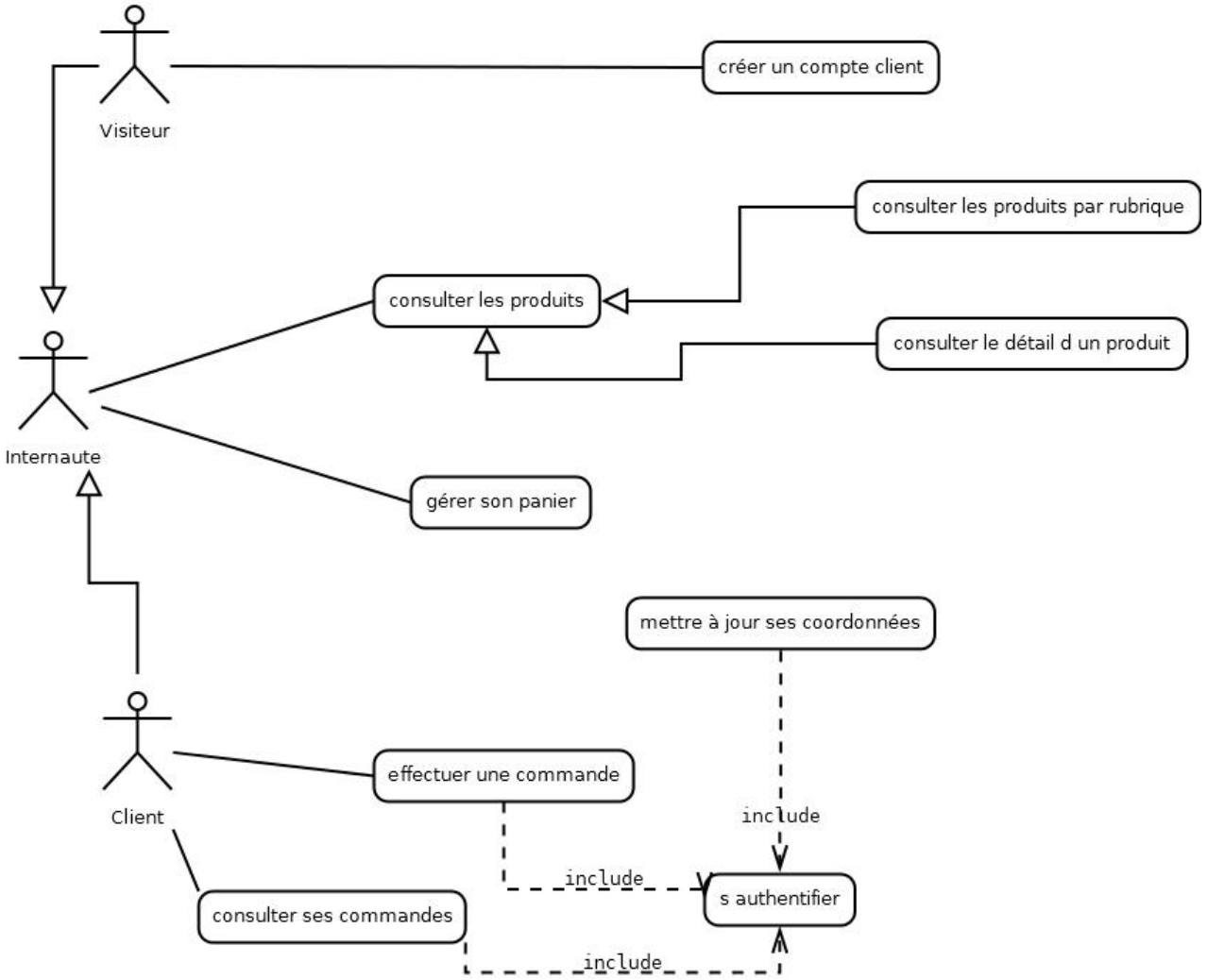
Mise à jour des données :
L'administrateur pourra ajouter des catégories ou des produits.
L'administrateur pourra mettre à jour les données des produits et des catégories.
L'administrateur pourra décider si un produit doit être affiché ou non.

Commande :
L'administrateur pourra consulter les commandes effectuées par les clients.

Statistique :
L'administrateur pourra consulter le nombre de commandes effectuées et le nombre de clients enregistrés.

Cas d’utilisations :

Cas d’utilisations des internautes :



Cas d'utilisations de l'administrateur :

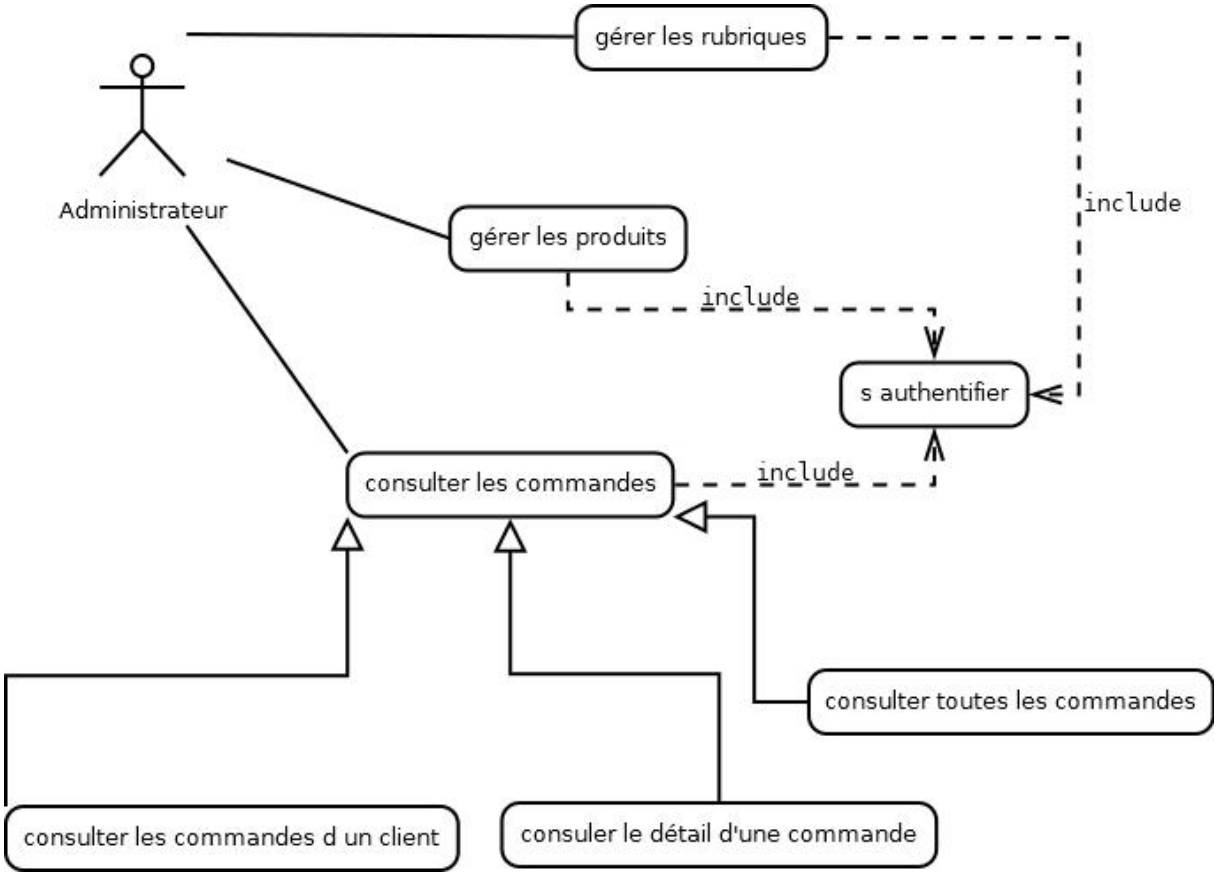


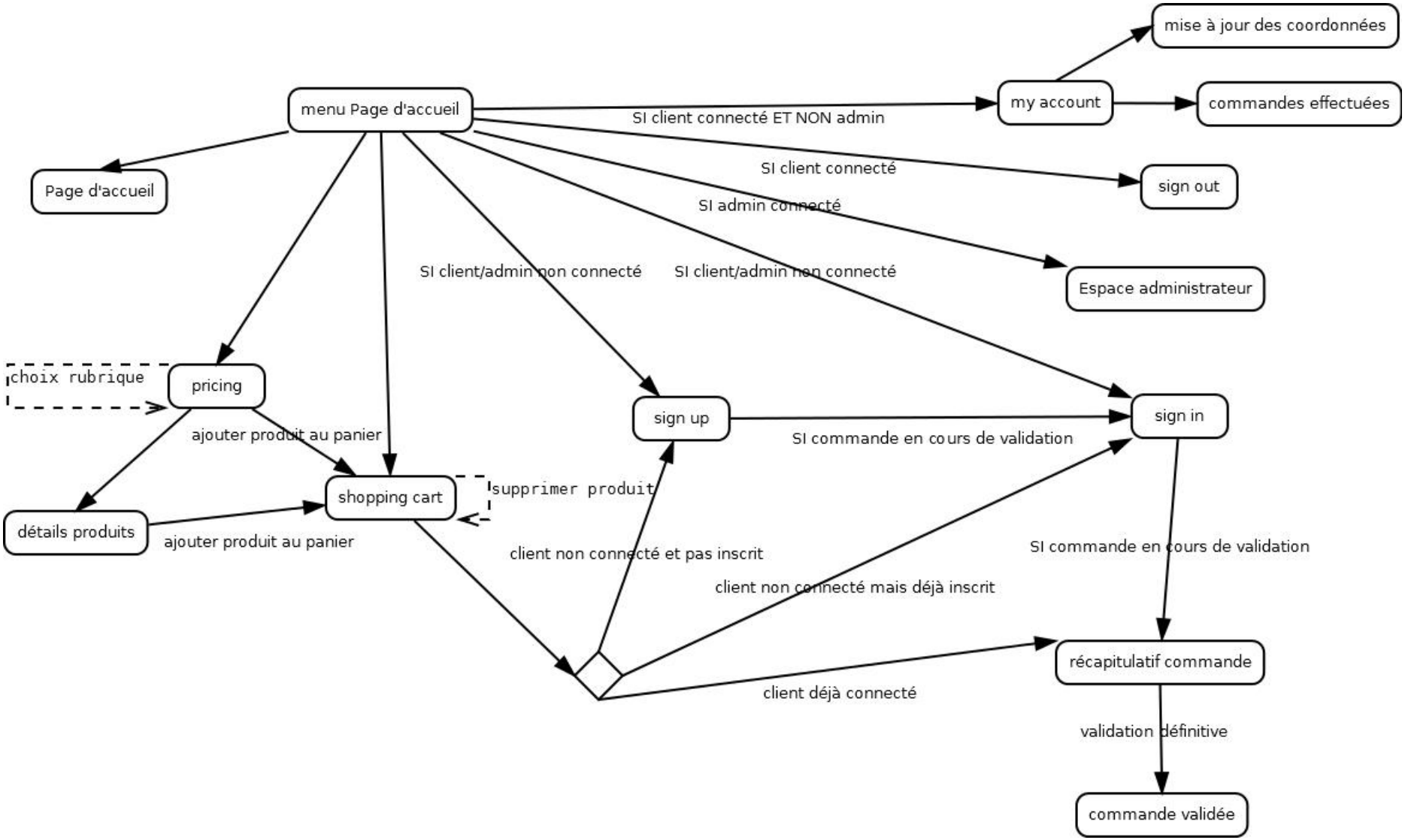
Tableau récapitulatif des besoins et état

En gris les besoins de l'internaute/client.
En blanc les besoins de l'administrateur.

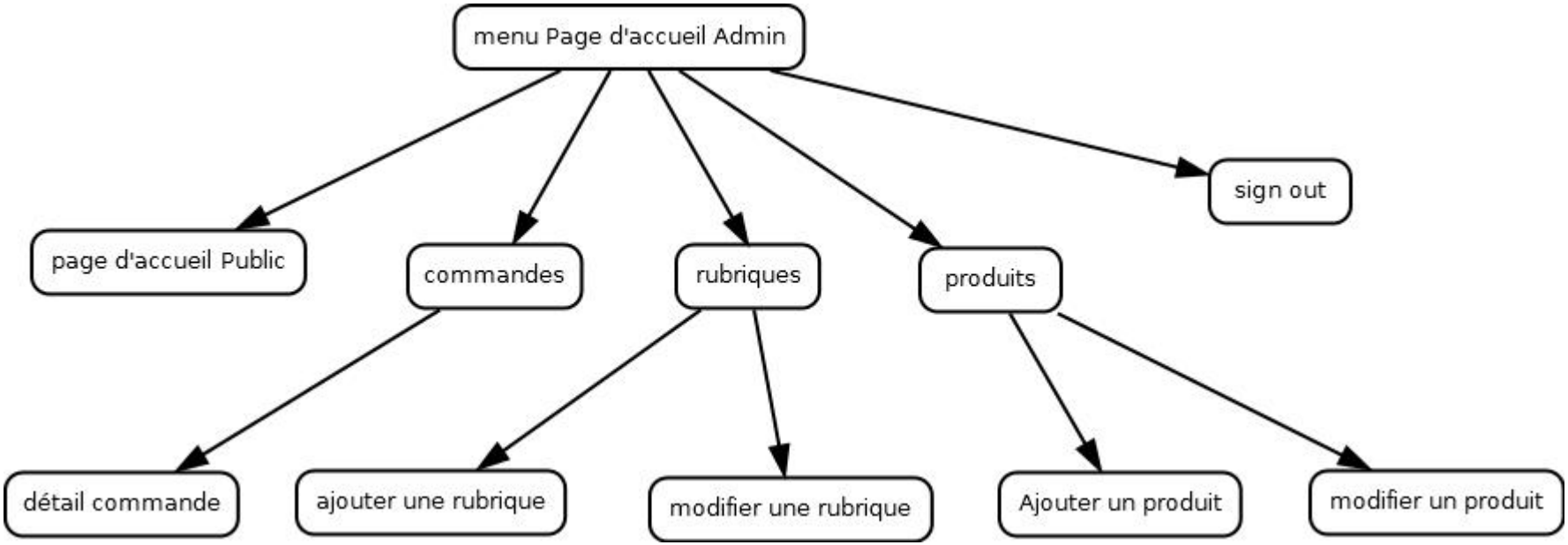
LIBELLE	ETAT	Page réalisée
Créer un compte client	OK	Page signUp
S'authentifier	OK	Page signIn
Modifier ses coordonnées	OK	Page signUpdate
Consulter les produits par rubrique	OK	Page pricing
Consulter les détails d'un produit	OK	Page pricingDetail
Ajouter un produit au panier	OK	Page pricing/ Page pricingDetail
Supprimer un produit du panier	OK	Page panier
Valider sa commande (devis)	OK	Page panier
Valider définitivement sa commande	OK	Page panierValidate
Consulter ses commandes	OK	Page myAccount
Consulter le détail de ses commandes	OK	Page myAccountDetailCommande
Se déconnecter	OK	Page signOut
Consulter la liste des rubriques existantes	OK	Page adminCategorie
Ajouter une rubrique	OK	Page adminCategorieAjout

LIBELLE	ETAT	Page réalisée
Modifier une rubrique	OK	Page adminCategorieModifier
Consulter la liste des produits existants	OK	Page adminProduit
Ajouter un produit	OK	Page adminProduitAjout
Modifier un produit	OK	Page adminProduitModifier
Consulter l'historique des commandes de tous les clients	OK	Page adminCommande
Consulter le détail d'une commande	OK	Page adminCommandeDetail
Consulter toutes les commandes d'un client	OK	Page adminClientDetail
Consulter le nombre total de commandes effectuées	OK	Page adminIndex
Consulter le nombre de clients enregistrés	OK	Page adminIndex
Se déconnecter	OK	Page signOut

Arborescence Art.space :



Arborescence Back Office :



Explication du fonctionnement général du programme et diagramme de classes :

- J'ai utilisé la programmation orienté objet et le patron Modèle Vue Contrôleur.
- J'utilise le patron Modèle Vue Contrôleur afin d'avoir une séparation claire entre les données et la partie graphique affichant les données et pour faciliter la maintenance et l'évolution du site (cf. page suivante).
- Chaque besoin exprimé dans le tableau récapitulatif des besoins possède un contrôleur.
- Un objet instanciant une *classe* ***client, produit, catégorie, commande*** *et* ***ligneCommande*** a pour rôle de représenter une ligne présente en BDD.
- Chacune de ces classes possèdent un ***manager*** qui s'occupe d'enregistrer ou de récupérer les informations en base de données.

Exemple de l'enregistrement d'un client dans la base de données :

- L'internaute saisit les informations dans le formulaire.
- Un objet client est créé et hydraté avec ces informations.
- Des méthodes de l'objet client contrôlent les informations contenues dans l'objet.
- Des méthodes du manager client contrôlent si par exemple un email ou un identifiant de l'objet existent déjà en base de données.
- Si les contrôles sont OK, le manager va insérer en base de données le nouveau client.

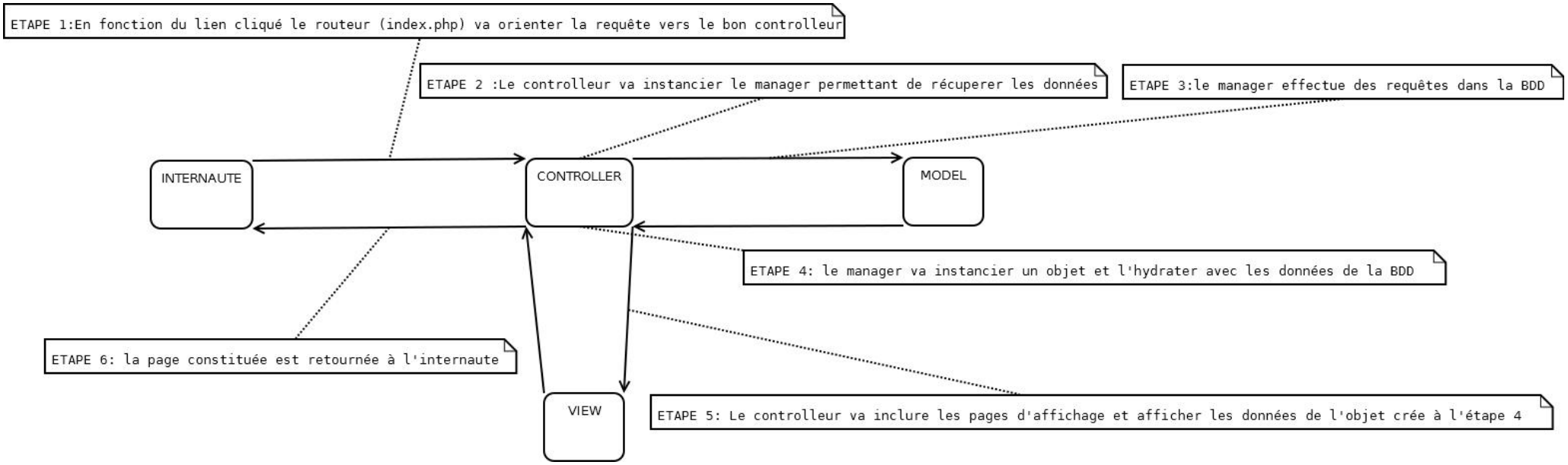
Exemple lors de la récupération des informations d'un client dans la base de données :

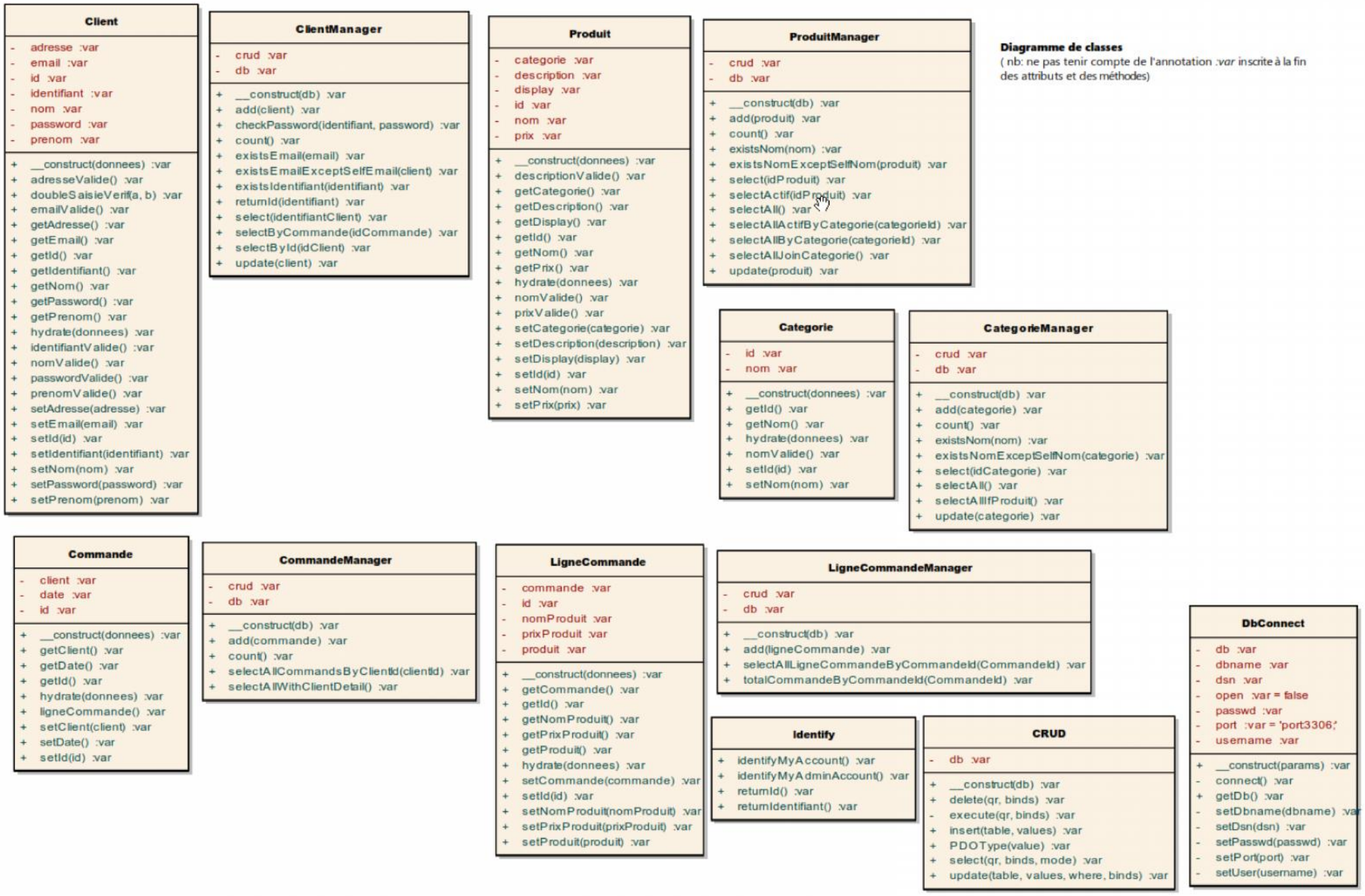
-Le manager va récupérer les informations d'un client dans la base de données.
-Le manager va créer un objet Client en l'hydratant avec ces informations.
- La vue affichera les informations contenues dans l'objet en utilisant les getters (ex : $client->getNom())

- Tous les managers utilisent la *classe* ***CRUD*** pour enregistrer, modifier ou sélectionner une entité.
- *La classe* ***dbConnect*** permet de se connecter à la base de données et contient une instance de PDO.
- *La classe* ***identify*** permet de contrôler si un internaute a les droits pour consulter une page. Par exemple, les pages d'administration ne peuvent être consultées que par l'administrateur.
- *La classe* ***panier*** est une classe qui permet d'enregistrer les différents achats du client avant l'enregistrement dans la base de données.

- Chaque objet est instancié grâce à un autoload.

Explication du patron Model View Controller appliqué au site ART.space




Client
- adresse :var
- email :var
- id :var
- identifiant :var
- nom :var
- password :var
- prenom :var
+ __construct(donnees) :var
+ adresseValide() :var
+ doubleSaisieVerif(a, b) :var
+ emailValide() :var
+ getAdresse() :var
+ getEmail() :var
+ getId() :var
+ getIdentifiant() :var
+ getNom() :var
+ getPassword() :var
+ getPrenom() :var
+ hydrate(donnees) :var
+ identifiantValide() :var
+ nomValide() :var
+ passwordValide() :var
+ prenomValide() :var
+ setAdresse(adresse) :var
+ setEmail(email) :var
+ setId(id) :var
+ setIdentifiant(identifiant) :var
+ setNom(nom) :var
+ setPassword(password) :var
+ setPrenom(prenom) :var
ClientManager
- crud :var
- db :var
+ __construct(db) :var
+ add(client) :var
+ checkPassword(identifiant, password) :var
+ count() :var
+ existsEmail(email) :var
+ existsEmailExceptSelfEmail(client) :var
+ existsIdentifiant(identifiant) :var
+ returnId(identifiant) :var
+ select(identifiantClient) :var
+ selectByCommande(idCommande) :var
+ selectById(idClient) :var
+ update(client) :var
Produit
- categorie :var
- description :var
- display :var
- id :var
- nom :var
- prix :var
+ __construct(donnees) :var
+ descriptionValide() :var
+ getCategorie() :var
+ getDescription() :var
+ getDisplay() :var
+ getId() :var
+ getNom() :var
+ getPrix() :var
+ hydrate(donnees) :var
+ nomValide() :var
+ prixValide() :var
+ setCategorie(categorie) :var
+ setDescription(description) :var
+ setDisplay(display) :var
+ setId(id) :var
+ setNom(nom) :var
+ setPrix(prix) :var
ProduitManager
- crud :var
- db :var
+ __construct(db) :var
+ add(produit) :var
+ count() :var
+ existsNom(nom) :var
+ existsNomExceptSelfNom(produit) :var
+ select(idProduit) :var
+ selectActif(idProduit) :var
+ selectAll() :var
+ selectAllActifByCategorie(categorieId) :var
+ selectAllByCategorie(categorieId) :var
+ selectAllJoinCategorie() :var
+ update(produit) :var
Diagramme de classes
(nb: ne pas tenir compte de l'annotation :var inscrite à la fin des attributs et des méthodes)
Categorie
- id :var
- nom :var
+ __construct(donnees) :var
+ getId() :var
+ getNom() :var
+ hydrate(donnees) :var
+ nomValide() :var
+ setId(id) :var
+ setNom(nom) :var
CategorieManager
- crud :var
- db :var
+ __construct(db) :var
+ add(categorie) :var
+ count() :var
+ existsNom(nom) :var
+ existsNomExceptSelfNom(categorie) :var
+ select(idCategorie) :var
+ selectAll() :var
+ selectAllIfProduit() :var
+ update(categorie) :var
Commande
- client :var
- date :var
- id :var
+ __construct(donnees) :var
+ getClient() :var
+ getDate() :var
+ getId() :var
+ hydrate(donnees) :var
+ ligneCommande() :var
+ setClient(client) :var
+ setDate() :var
+ setId(id) :var
CommandeManager
- crud :var
- db :var
+ __construct(db) :var
+ add(commande) :var
+ count() :var
+ selectAllCommandsByClientId(clientId) :var
+ selectAllWithClientDetail() :var
LigneCommande
- commande :var
- id :var
- nomProduit :var
- prixProduit :var
- produit :var
+ __construct(donnees) :var
+ getCommande() :var
+ getId() :var
+ getNomProduit() :var
+ getPrixProduit() :var
+ getProduit() :var
+ hydrate(donnees) :var
+ setCommande(commande) :var
+ setId(id) :var
+ setNomProduit(nomProduit) :var
+ setPrixProduit(prixProduit) :var
+ setProduit(produit) :var
LigneCommandeManager
- crud :var
- db :var
+ __construct(db) :var
+ add(ligneCommande) :var
+ selectAllLigneCommandeByCommandeId(CommandeId) :var
+ totalCommandeByCommandeId(CommandeId) :var
Identify
+ identifyMyAccount() :var
+ identifyMyAdminAccount() :var
+ returnId() :var
+ returnIdentifiant() :var
CRUD
- db :var
+ __construct(db) :var
+ delete(qr, binds) :var
- execute(qr, binds) :var
+ insert(table, values) :var
+ PDOType(value) :var
+ select(qr, binds, mode) :var
+ update(table, values, where, binds) :var
DbConnect
- db :var
- dbname :var
- dsn :var
- open :var = false
- passwd :var
- port :var = "port3306;"
- username :var
+ __construct(params) :var
- connect() :var
+ getDb() :var
- setDbname(dbname) :var
- setDsn(dsn) :var
- setPasswd(passwd) :var
- setPort(port) :var
- setUser(username) :var

Description des méthodes de chaque classe (les getters et setters sont omis)

Classe	Nom de la méthode	contrôle de l identifiant @return boolean
Client	__construct(array $donnees)	* constructeur * @param array $donnees
	identifiantValide()	* contrôle de l identifiant * @return boolean
	emailValide()	* contrôle de l'email * @return boolean
	passwordValide()	* contrôle du mot de passe * @return boolean
	nomValide()	* contrôle du nom * @return boolean
	prenomValide()	* contrôle du prénom * @return boolean
	adresseValide()	* contrôle de l'adresse * @return boolean
	doubleSaisieVerif($a, $b)	* contrôle de la double saisie * @param string $a * @param string $b * @return boolean
	hydrate(array $donnees)	* hydratation de l'objet * @param array $donnees
ClientManager	__construct($db)	* constructeur * @param DbConnect $db
	add($client)	* ajoute un client * @param Client $client * @return int (lastInsertId)
	select($identifiantClient)	* recuperer les informations d'un client via son identifiant * @param string $identifiantClient * @return Client

	selectById($idClient)	* recuperer les informations d'un client via son id * @param id $idClient * @return Client
	selectByCommande($idCommande)	* recuperer les informations d'un client via un numéro de commande * @param id $idCommande * @return Client
	update($client)	* mise à jour d'un client * @param Client $client * @return void
	count()	* compte le nombre de clients dans la BDD * @return int
	existsIdentifiant($identifiant)	* tester si Identifiant est existant dans la BDD * @param string $identifiant * @return boolean
	returnId($identifiant)	* return id du client avec l identifiant * @param string $identifiant * @return int
	existsEmail($email)	* tester si email est existant dans la BDD * @param string $email * @return boolean
	existsEmailExceptSelfEmail($client)	* tester si l'email est existant dans la BDD en excluant l'objet testé * @param Client $client * @return boolean
	checkPassword($identifiant, $password)	* verifier le couple identifiant / mot de passe * @param string $identifiant * @param string $password * @return boolean
Produit	__construct(array $donnees)	* constructeur * @param array $donnees
	nomValide()	* contrôle le nom * @return boolean
	descriptionValide()	* contrôle la description * @return boolean
	prixValide()	* contrôle le prix

		* @return boolean
	hydrate(array $donnees)	* hydratation de l'objet * @param array $donnees
ProduitManager	__construct($db)	* constructeur * @param DbConnect $db
	add($produit)	* ajoute un produit * @param Produit $produit * @return int (lastInsertId)
	select($idProduit)	* recuperer les informations d'un produit via son id * @param int $idProduit * @return Produit
	selectActif($idProduit)	* recupere les informations d'un produit si display=true * @param type $idProduit * @return Produit
	selectAll()	* recuperer tous les produits * @return array[Produit]
	selectAllJoinCategorie()	* recuperer tous les produits avec les informations de la table catégorie * @return array[\entity\Produit]
	selectAllByCategorie($categorieId)	* récupere tous les produits d'une catégorie * @param int $categorieId * @return array[\entity\Produit]
	selectAllActifByCategorie($categorieId)	* récupère tous les produits avec display=true d'une catégorier * @param int $categorieId * @return array[\entity\Produit]
	update($produit)	* modification d'un produit * @param Produit $produit * @return void
	existsNom($nom)	* tester si le nom est existant dans la BDD * @param string $nom * @return boolean
	existsNomExceptSelfNom($produit)	* tester si le nom est existant dans la BDD en excluant l'objet testé * @param Produit $produit * @return boolean
	count()	* connaitre le nombre de produits dans la BDD

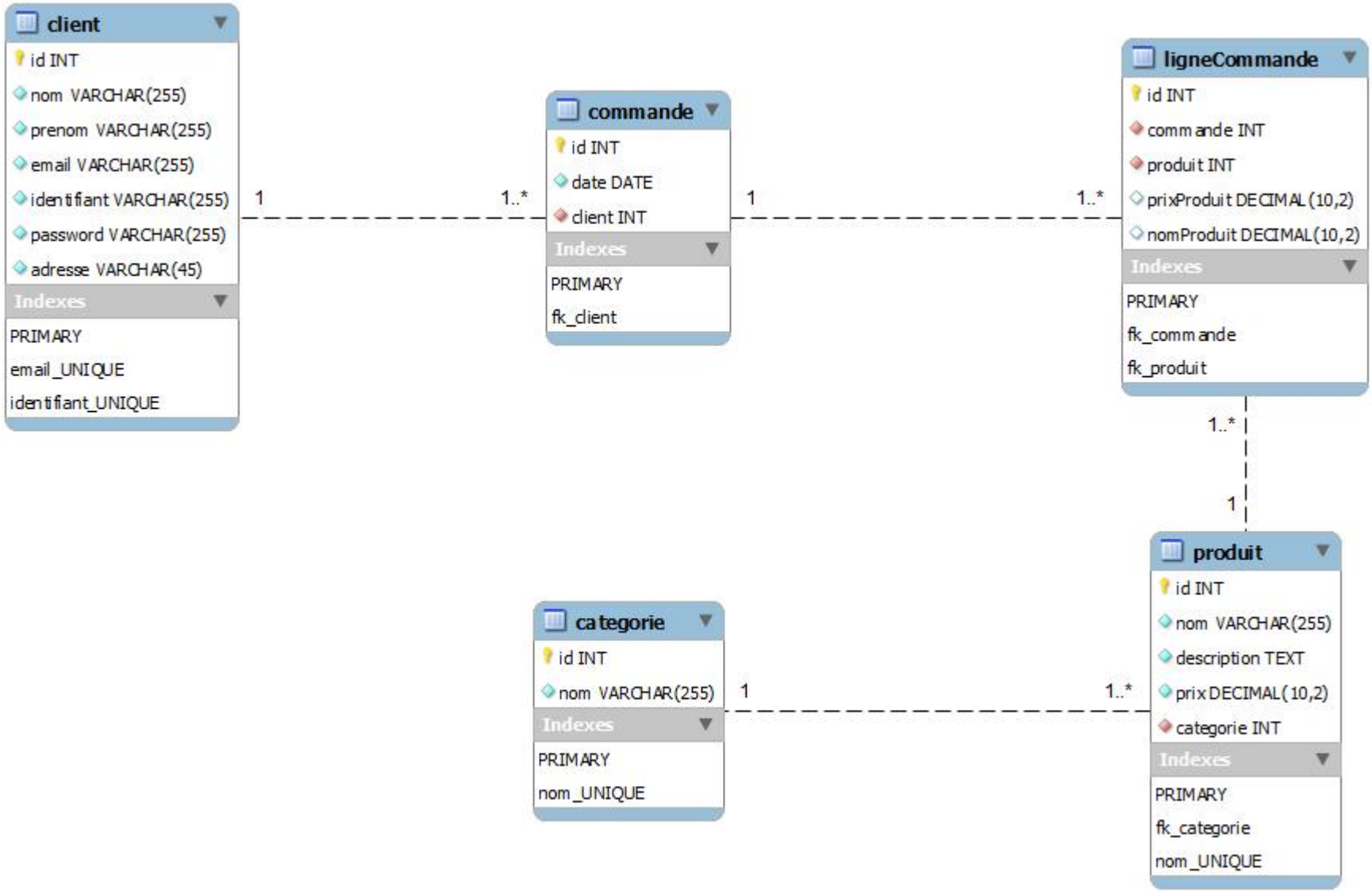
		* @return int
Catégorie	__construct(array $donnees)	* constructeur * @param array $donnees
	nomValide()	* Contrôle du nom * @return boolean
	hydrate(array $donnees)	* hydratation * @param array $donnees
CategorieManager	__construct($db)	* Constructeur * @param DbConnect $db * @return void
	add($categorie)	* Ajoute une catégorie * @param Categorie $categorie * @return int (lastInsertId)
	select($idCategorie)	* récupérer les informations d'une categorie via son id * @param int $idCategorie * @return Categorie
	selectAll()	* récupérer toutes les catégories * @return array[Categorie]
	selectAllIfProduit()	* récupérer toutes les catégories qui possèdent au moins un produit avec display=true * @return array[Categorie]
	update($categorie)	* mise à jour d'une catégorie * @param Categorie $categorie * @return void
	count()	* connaitre le nombre de catégories dans la BDD * @return int
	existsNom($nom)	* tester si le nom est existant dans la BDD * @param string $nom * @return boolean
	existsNomExceptSelfNom($categorie)	* tester si le nom est existant dans la BDD en excluant l'objet testé * @param Categorie $categorie * @return boolean
Commande	__construct(array $donnees = array())	* Constructeur * @param array $donnees

	hydrate(array $donnees)	* hydratation * @param array $donnees
CommandeManager	__construct($db)	* constructeur * @param DbConnect $db
	add($commande)	* ajoute une commande * @param Commande $commande * @return int (lastInsertId)
	selectAllCommandsByClientId($clientId)	* sélectionne toutes les commandes du client via son id * @param int $clientId * @return array
	selectAllWithClientDetail()	* récupérer toutes les commandes avec les lignes commandes associées * @return array
	count()	* compte le nombre de commandes dans la base * @return int
LigneCommande	__construct(array $donnees)	* constructeur * @param array $donnees
	hydrate(array $donnees)	* hydratation * @param array $donnees
LigneCommandeManager	__construct($db)	* Constructeur * @param DbConnect $db
	add($ligneCommande)	* ajoute une ligne de commande * @param ligneCommande $ligneCommande * @return int (lastInsertId)
	selectAllLigneCommandeByCommandeId($CommandeId)	* récupère toutes les lignes commandes d'une commande * @param int $CommandeId * @return array
	totalCommandeByCommandeId($CommandeId)	* calcul le prix total d'une commande avec les prix de chaque ligne commande * @return float
DbConnect	__construct($params)	constructeur
	connect()	* création de l'objet PDO avec les informations de connexion * @param PDO $db * @return void

CRUD	__construct($db)	* constructeur * @param PDO $db
	select($qr, $binds = array(), $mode = PDO::FETCH_ASSOC)	* Selection * @param array $qr * @param array $binds * @param int $mode * @return array
	insert($table, $values)	* Insertion * @param string $table * @param array $values * @return int
	PDOType($value)	* détermine le type PDO d'une valeur * @param mixed $value * @return integer PDO::PARAM
	delete($qr, $binds = array())	* Supression * @param array $qr * @param array $binds * @return boolean
	update($table, $values, $where = '', $binds = array())	* modification * @param string $table * @param array $values * @param string $where * @param array $binds * @return boolean
	execute($qr, $binds)	* binds de la requête * @param string $qr * @param array $binds * @return boolean
Identify	identifyMyAccount()	* Contrôle si la session du client est valide * @return void
	identifyMyAdminAccount()	* Contrôle si la session administrateur est valide * @return void
	returnId()	* retourne l'id du client en session * @return int

	returnIdentifiant()	* retourne l'identifiant du client en session * @return string
Panier	__construct()	* enregistre la date du jour
	setLignePanier($donnees)	* ajoute une ligne panier * @param array $donnees * @return void
	delLignePanier($idLignePanier)	* supprime une ligne panier * @param int $idLignePanier * @return void
	totalPanier()	* calcul le prix total du panier * @return float

Schéma de la base de données :



Les associations entre les tables et les contraintes permettent de garantir l'intégrité des données :

- Une commande doit obligatoirement avoir un numéro de client.
- une ligne commande doit obligatoirement avoir un numéro de commande et un produit.
- un produit doit obligatoirement avoir une catégorie.
- Un email et un identifiant de la table client doivent être uniques dans la base de données. Ces contraintes sont contrôlées par des méthodes de la classe ClientManager.
- Le nom d'un produit doit être unique dans la base de données. Cette contrainte est contrôlée par une méthode de la classe ProduitManager.
- Le nom d'une catégorie doit être unique dans la base de données. Cette contrainte est contrôlée par une méthode de la classe CatégorieManager.
- l'administrateur ne peut pas supprimer les produits et les catégories, il a le choix d'afficher les produits ou non sur le site .Une rubrique ne sera affichée que si elle possède des produits.

Exemple -> Etapes lors de la création d'une commande :

1. Création des lignes dans la table commande.
2. Création des lignes dans la table ligne commande.
   Le nom du produit et le prix du produit sont sauvegardés dans les lignes de la table ligneCommande. Cela permet de garder en historique ces informations. En effet, le nom et le prix d'un produit pourront changer sans affecter les informations contenues dans la table ligneCommande.

Améliorations envisagées :

- Permettre au client de changer son mot de passe ou de le retrouver en cas d'oubli.
- Lors de la validation de l'achat, permettre au client de rentrer une adresse de livraison différente de son adresse.
- Redéfinir les URL du site web grâce à l'URL rewriting afin d'améliorer le référencement