|  |  |  |
| --- | --- | --- |
| EPCF2 | … Janvier  2024 | |
| Dossier projet – Fresh ! L’application WEB, la conception et développement de la persistance des données en intégrant les recommandations de sécurité. | | EXPERNET |

Table des matières

[I. Liste des compétences du référentiel qui sont couvertes par le projet 1](#_Toc156228675)

[II. Résumé du projet en anglais d'une longueur d’environ 5 à 10 lignes soit 100 à 120 mots, ou environ 600 caractères espaces non compris 3](#_Toc156228676)

[III. Cahier des charges, expression des besoins, ou spécifications fonctionnelles du projet 4](#_Toc156228677)

[IV. Spécifications fonctionnelles du projet 5](#_Toc156228678)

[V. Spécifications techniques du projet, élaborées par le candidat, y compris pour la sécurité 6](#_Toc156228679)

[VI. Réalisations du candidat comportant les extraits de code les plus significatifs, et en les argumentant, y compris pour la sécurité 7](#_Toc156228680)

[VII. Présentation du jeu d’essai élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues) 8](#_Toc156228681)

[VIII. Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité 9](#_Toc156228682)

[IX. Description d’une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet 10](#_Toc156228683)

# Liste des compétences du référentiel qui sont couvertes par le projet

|  |  |
| --- | --- |
| Concevoir et développer la persistance des données en intégrant les recommandations de sécurité | Mettre en place une base de données |
| Concevoir une base de données |
| Développer des composants dans le langage d’une base de données |

Le projet couvre obligatoirement les compétences suivantes :

* Mettre en place une base de données
* Concevoir une base de données
* Développer des composants dans le langage d’une base de données

# Résumé du projet en anglais d'une longueur d’environ 5 à 10 lignes soit 100 à 120 mots, ou environ 600 caractères espaces non compris

Fresh is a web application designed to manage refrigerator inventory by tracking expiration dates for each food item, thereby reducing waste. The principle is straightforward: users log in, create up to 2 fridges, and can add 100 food items to each fridge. A grouping tool prevents users from duplicating items within a fridge. Users have the ability to modify or delete items, providing a clear indication of when an item is consumed.

Additionally, users receive alerts notifying them if a food item has expired or is nearing its expiration date. This system empowers users to proactively manage their food inventory, minimizing waste and promoting efficient consumption practices.

# Cahier des charges, expression des besoins, ou spécifications fonctionnelles du projet

Le présent cahier des charges définit les exigences et les spécifications pour le développement d'une application web d’inventaire d’aliments.

• Contexte

Fresh (Entreprise) souhaite mettre en place une solution permettant à n’importe qui, la possibilité d’avoir un inventaire d’aliments disponible pour les réfrigérateurs des utilisateurs.

• Fonctionnalités

Les fonctionnalités permettent de savoir ce qui est fonctionnel pour l’utilisateur

1. La plateforme a une page d’enregistrement

* L’utilisateur doit se créer un compte pour accéder aux autres pages.
* L’utilisateur doit mettre les informations tel que :
  + - * + Son nom,
        + Son prénom,
        + Son adresse mail,
        + Son mot de passe,
        + Une case à cocher pour les conditions d’utilisations.

1. La plateforme a une page de connexion.

* Identification de l’utilisateur connecté.
* L’utilisateur peut se déconnecter.
* L’utilisateur peut accéder à ses informations.

1. La plateforme a une page de compte utilisateur.

* L’utilisateur peut modifier les informations de son compte.

1. La plateforme a un système qui permet d’activer un compte avec une vérification par courriel.

* Un compte qui n’est pas activé n’a pas accès aux autres pages.

1. La plateforme a une page d’accueil.

* Listage des alertes de péremption d’aliments.
* Avec option de masquage/affiche.
* 1er masquage automatique après 10 secondes.
* Listage des réfrigérateurs.
* L’utilisateur peut cliquer sur un réfrigérateur pour accéder aux aliments disponibles dans ce dernier.
* Possibilité de crée un réfrigérateur si l’utilisateur à moins de 2 réfrigérateurs.

1. La plateforme a une page pour chaque réfrigérateur de chaque utilisateur.

* Listage des aliments dans le réfrigérateur avec :
  + Le nom,
  + La date d’ajout,
  + La quantité
  + La date d’expiration,
  + Un message permettant de savoir si l’aliment est périmé ou sera périmé (en-dessous de 4 jours supérieur à la date d’accès à la page).
* Possibilité d’ajouter un aliment dans le réfrigérateur.
* Possibilité de modifier un aliment dans le réfrigérateur.
* Possibilité de regrouper des aliments qui ont presque le même nom et qui ont la même date d’expiration.
* Possibilité de supprimer des aliments.
* Possibilité de modifier le nom du réfrigérateur.
* Possibilité de supprimer le réfrigérateur avec tous les aliments.

Fonctionnalité de la base de données

1. Une procédure est nécessaire pour récupérer une liste de dernières alertes de chaque aliment périmé ou qui seront périmés (en-dessous de 4 jours à la date) à partir du jour d’appel (généralement appelée JOUR-J).
2. Un déclencheur est nécessaire pour insérer les alertes pour les aliments périmé ou qui seront périmés (ce-dernier s’exécutera après l’insertion d’une nourriture).
3. Un événement est nécessaire pour insérer tous les jours, les alertes pour les aliments périmé ou qui seront périmés.

• Technologies utilisées

* Conception de la plateforme
  + Modèle conceptuel de données avec Looping.
* Développement de la plateforme
* Back-end en PHP 8.3.1, Symfony 7.0.2
* Front-end en CSS 3, NodeJs 21.6.0 (npm 9.6.5), Webpack Encore 4.5.0, TailwindCss 3.4.0, daisyUi 4.6.0
* Persistance des données
* La base de données avec MySQL 8.3.0 en locale.

• Livrable et délais

* La plateforme complète et fonctionnelle doit être livrée conformément aux spécifications.
* La date de livraison prévue pour la plateforme est le 22/01/2024.

# Spécifications fonctionnelles du projet

• Développement de la plateforme

* + La gestion des composants de développement s’est faite avec
    - Composer 2.6.6 et NodeJs 21.6.0 (npm 9.6.5)
  + Les ressources (assets) utilisées sont traitées par
    - Webpack et PostCSS.
  + Outil de version avec Git 2.38.1

• Sécurité de la plateforme

* + Un système d’authentification est mis en place sur la plateforme avec une page de connexion.
  + Les mots de passes des utilisateurs sont chiffrés depuis la plateforme en utilisant bCrypt.

• Les routes dans le code sont sécurisées en vérifiant si l’utilisateur est connecté.

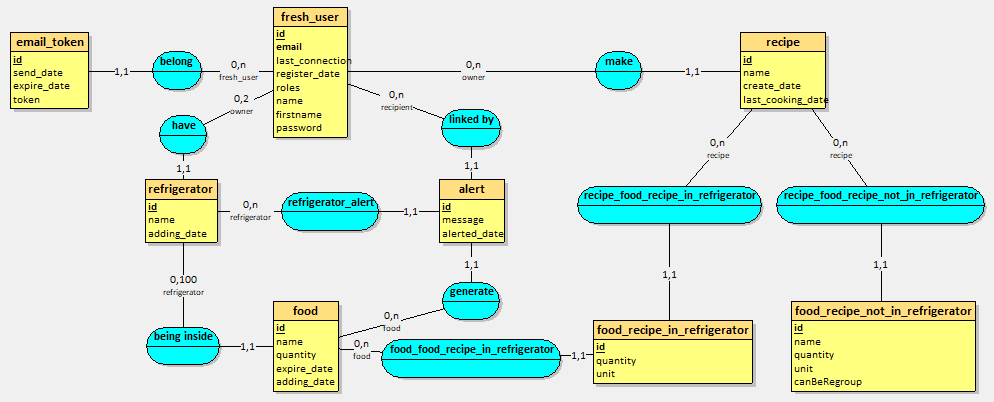
• La manipulation de la base de données dans le code est sécurisée avec des requêtes préparées.

• Les classes dans le code sont encapsulées.

• La base de données à un utilisateur root avec un mot de passe fort.

• L’application a son propre utilisateur authentifié avec un mot de passe fort, sur le serveur de gestion de base de données, il a toutes les permissions sur une seule base (base exclusivement dédiée à l’application)

# Spécifications techniques du projet, élaborées par le candidat, y compris pour la sécurité



Dans ce MCD, on retrouve 8 entités et 8 associations.

L’entité « fresh\_user » possède un identifiant « id ».

L’entité « email\_token » possède un identifiant « id » et est associé à 1 et 1 seul « fresh\_user », l’entité « fresh\_user » peut posséder 0 ou plusieurs « email\_token ».

L’entité « refrigerator » possède un identifiant « id » et est associé à 1 et 1 seul « fresh\_user », l’entité « fresh\_user » peut avoir de 0 à 2 « refrigerator » au maximum.

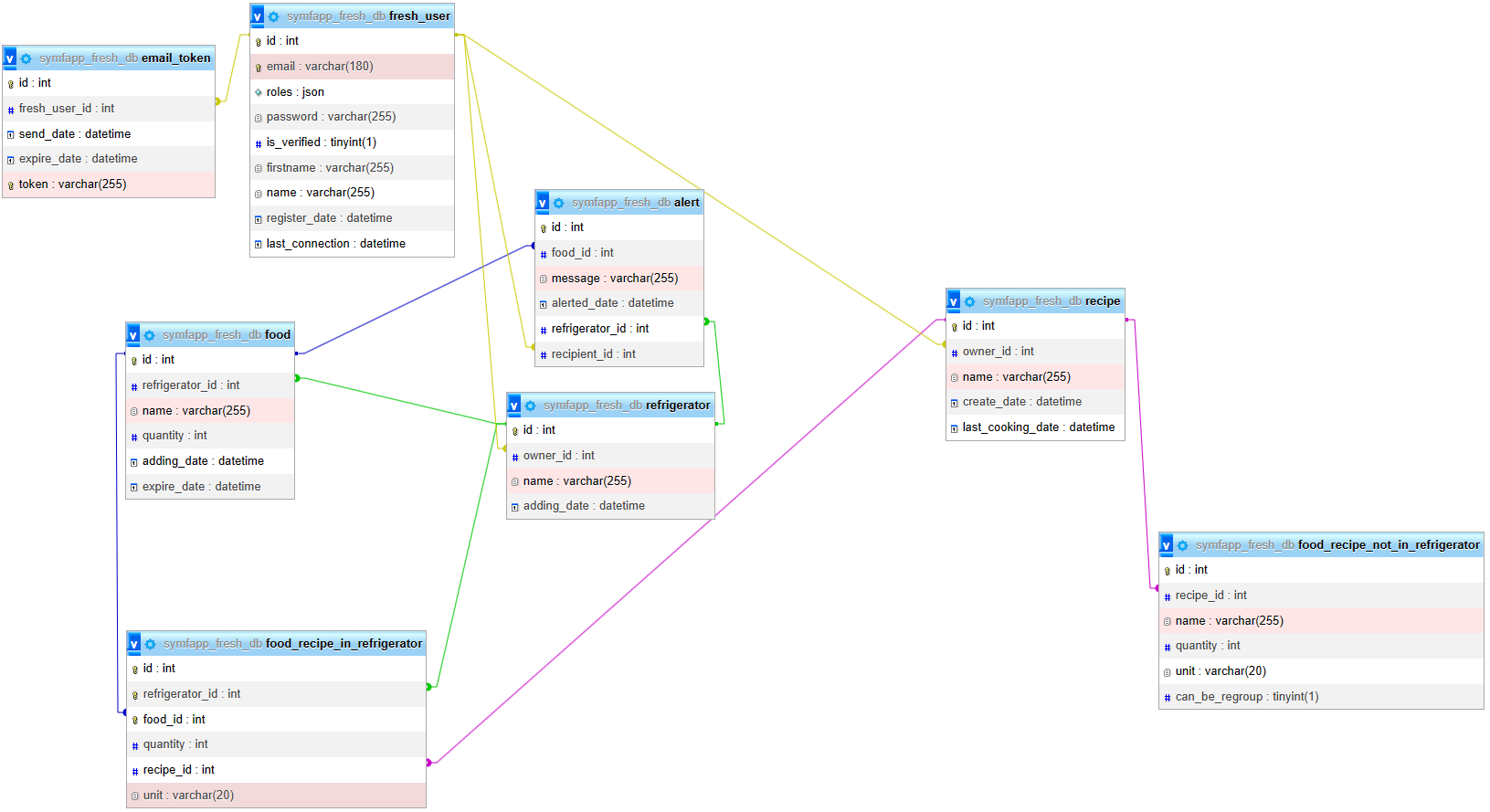
L’entité « food » possède un identifiant « id » et est associé à 1 et 1 seul « refrigerator », l’entité « refrigerator » peut avoir de 0 à 100 « food ».

L’entité « alert » possède un identifiant « id » et est associé à 1 et 1 seul « refrigerator », 1 et 1 seul « fresh\_user » et 1 et 1 seul « food », les entités « refrigerator », « food », « fresh\_user » peuvent apparaitre dans 0 ou plusieurs « alert ».

L’entité « recipe » possède un identifiant « id » et est associé à 1 et 1 seul « fresh\_user », l’entité « fresh\_user » peut créer 0 ou plusieurs recettes.

L’entité « food\_recipe\_not\_in\_refrigerator » possède un identifiant « id » et est associé à 1 et 1 seul « recipe », l’entité « recipe » peut contenir 0 ou plusieurs « food\_recipe\_not\_in\_refrigerator ».

L’entité « food\_recipe \_in\_refrigerator » possède un identifiant « id » et est associé à 1 et 1 seul « recipe » et à 1 et 1 seul « food », les entités « recipe » et « food » peuvent contenir 0 ou plusieurs « food\_recipe \_in\_refrigerator ».



Dans ce Modèle Physique des Données, on observe 8 tables.

Voici en modèle textuel :

**fresh\_user = (id *INT***, **email *VARCHAR(255)***, last\_connection *DATETIME*, register\_date *DATETIME*, roles ***array***, name ***VARCHAR(255)***, firstname ***VARCHAR(255)***, password ***VARCHAR(255)*);**

**refrigerator = (id *INT***, name ***VARCHAR(255)***, adding\_date *DATETIME, #id\_owner***);**

**#id\_owner** fait référence à **id** de la table fresh\_user.

**email\_token = (id *INT***, send\_date *DATETIME*, expire\_date *DATETIME*, token *VARCHAR(255), #id\_fresh\_user***);**

**#id\_fresh\_user** fait référence à **id** de la table fresh\_user.

**recipe = (id *INT***, name *VARCHAR(255)*, create\_date *DATETIME*, last\_cooking\_date *DATETIME, #id\_owner***);**

**#id\_owner** fait référence à **id** de la table fresh\_user.

**food\_recipe\_not\_in\_refrigerator = (id *INT***, quantity ***INT***, unit ***VARCHAR(20)***, canBeRegroup *BOOLEAN, #id\_recipe***);**

**#id\_recipe** fait référence à **id** de la table recipe.

**food = (id *INT***, name ***VARCHAR(255)***, quantity ***INT***, expire\_date *DATETIME*, adding\_date *DATETIME, #id\_refrigerator***);**

**#id\_refrigerator** fait référence à **id** de la table refrigerator.

**alert = (id *INT***, message ***VARCHAR(255)****, #id\_recipient, #id\_food, #id\_refrigerator***);**

**#id\_recipient** fait référence à **id** de la table fresh\_user.

**#id\_food** fait référence à **id** de la table food.

**#id\_refrigerator** fait référence à **id** de la table refrigerator.

**food\_recipe\_in\_refrigerator = (id *INT***, quantity ***INT***, unit ***VARCHAR(20)****, #id\_recipe, #id\_food***);**

**#id\_recipe** fait référence à **id** de la table recipe.

**#id\_food** fait référence à **id** de la table food.

Une fois transformé en base de données, elle se nomme « symfapp\_fresh\_db », la sécurisation d’accès a dû être faite.

L’utilisateur « root » a un mot de passe fort. L’utilisateur « symfapp\_fresh » a été créé et possède aussi un mot de passe fort, il a tous les droits sur la base de données « symfapp\_fresh\_db ».

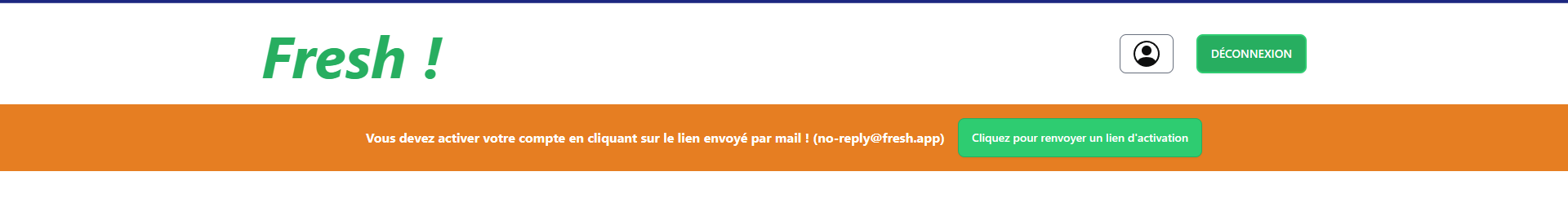
L’utilisateur « root » se connecte à partir de l’hôte localhost. L’utilisateur symfapp\_fresh peut se connecter à partir de n’importe quel hôte.

|  |  |
| --- | --- |
| Utilisateur | Permission |
| root | GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, SHUTDOWN, PROCESS, FILE, REFERENCES, INDEX, ALTER, SHOW DATABASES, SUPER, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE TABLESPACE, CREATE ROLE, DROP ROLE ON \*.\* TO `root`@`localhost` WITH GRANT OPTION;  GRANT ALLOW\_NONEXISTENT\_DEFINER,APPLICATION\_PASSWORD\_ADMIN,AUDIT\_ABORT\_EXEMPT,AUDIT\_ADMIN,AUTHENTICATION\_POLICY\_ADMIN,BACKUP\_ADMIN,BINLOG\_ADMIN,BINLOG\_ENCRYPTION\_ADMIN,CLONE\_ADMIN,CONNECTION\_ADMIN,ENCRYPTION\_KEY\_ADMIN,FIREWALL\_EXEMPT,FLUSH\_OPTIMIZER\_COSTS,FLUSH\_STATUS,FLUSH\_TABLES,FLUSH\_USER\_RESOURCES,GROUP\_REPLICATION\_ADMIN,GROUP\_REPLICATION\_STREAM,INNODB\_REDO\_LOG\_ARCHIVE,INNODB\_REDO\_LOG\_ENABLE,PASSWORDLESS\_USER\_ADMIN,PERSIST\_RO\_VARIABLES\_ADMIN,REPLICATION\_APPLIER,REPLICATION\_SLAVE\_ADMIN,RESOURCE\_GROUP\_ADMIN,RESOURCE\_GROUP\_USER,ROLE\_ADMIN,SENSITIVE\_VARIABLES\_OBSERVER,SERVICE\_CONNECTION\_ADMIN,SESSION\_VARIABLES\_ADMIN,SET\_ANY\_DEFINER,SHOW\_ROUTINE,SYSTEM\_USER,SYSTEM\_VARIABLES\_ADMIN,TABLE\_ENCRYPTION\_ADMIN,TELEMETRY\_LOG\_ADMIN,XA\_RECOVER\_ADMIN ON \*.\* TO `root`@`localhost` WITH GRANT OPTION;  GRANT PROXY ON ``@`` TO `root`@`localhost` WITH GRANT OPTION; |
| symfapp\_fresh | GRANT USAGE ON \*.\* TO `symfapp\_fresh`@`%`;  GRANT ALL PRIVILEGES ON `symfapp\\_fresh\\_%`.\* TO `symfapp\_fresh`@`%`;  GRANT ALL PRIVILEGES ON `symfapp\\_fresh\\_db`.\* TO `symfapp\_fresh`@`%`; |

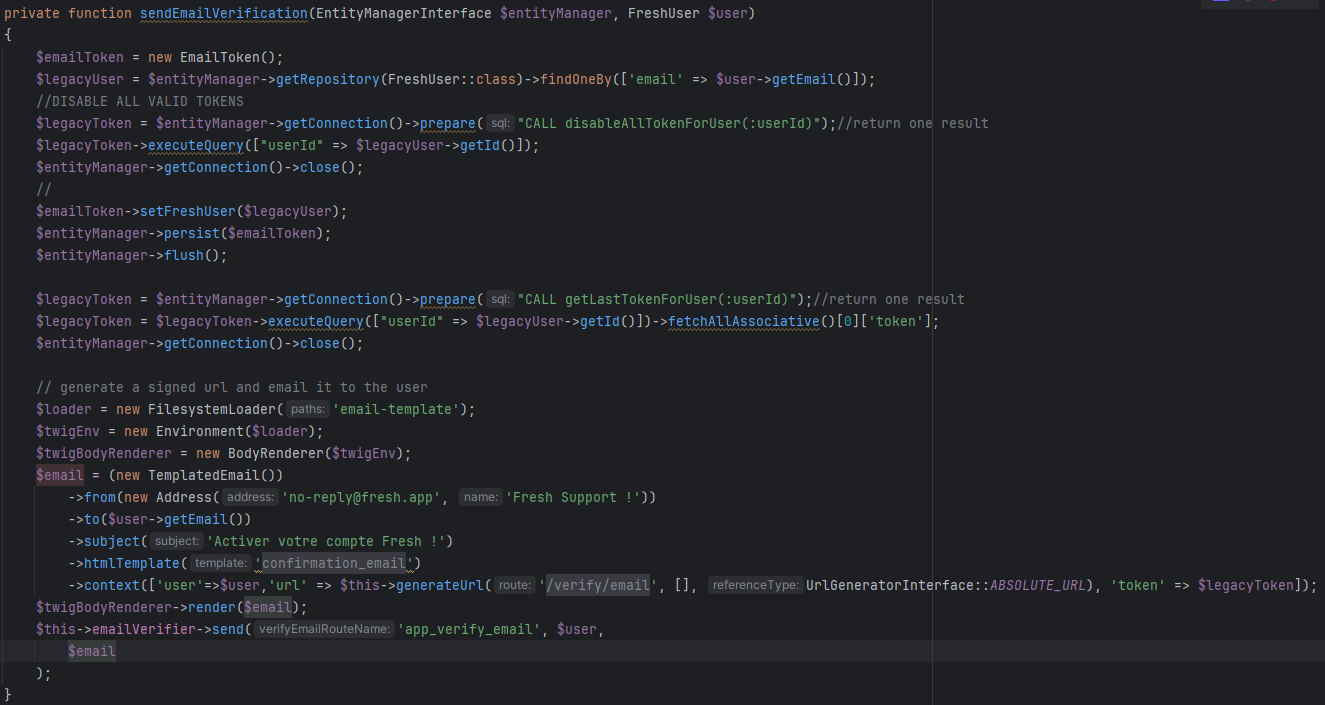
# Réalisations du candidat comportant les extraits de code les plus significatifs, et en les argumentant, y compris pour la sécurité

Développement de la vérification de compte :

L’utilisateur doit se créer un compte (sauf s’il possède déjà un) et se connecter.

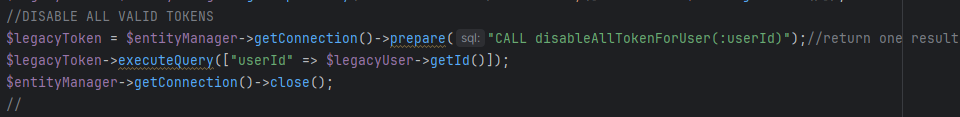
Il doit confirmer son compte pour pouvoir utiliser l’application.

Un courriel est envoyé à l’utilisateur, l’utilisateur peut cliquer sur le bouton.

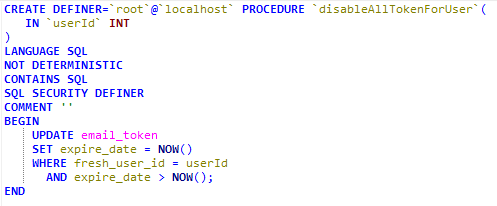


Un EmailToken est créé permettant l’utilisateur d’activer son compte sans se connecter.

Il faut d’abord désactiver tous ses EmailToken déjà existant.



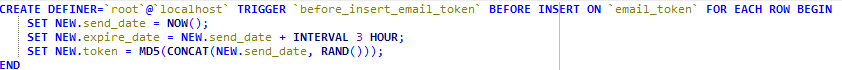
La procédure « disableAllTokenForUser(int userId) » est donc appelée avec une requête préparée (pour la sécurité).



Création de la procédure.

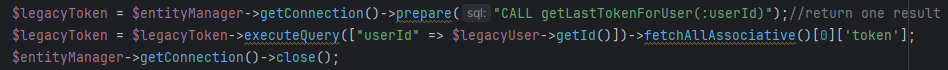
La procédure met toutes les « expire\_date » (date d’expiration) qui ont dépassé la date d’aujourd’hui, à la date d’aujourd’hui(la date d’expiration sera donc aujourd’hui), pour les lignes d’ « email\_token » ayant l’utilisateur. Une date d’expiration égale ou inférieure à la date d’aujourd’hui donnent des « email\_token » invalide qui ne permettent pas d’activer un compte utilisateur.

Lorsque qu’un « email\_token » est créé, un déclencheur s’exécute avant la persistance.

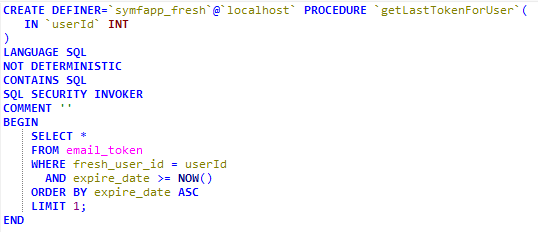


La date d’envoie est celle du jour d’exécution, la validité est de 3 heures et le token est généré automatiquement et hashé en MD5.

Puis on récupère le token d’« email\_token » généré par MySQL.



En faisant appel à la procédure « getLastTokenForUser(int userId) ».



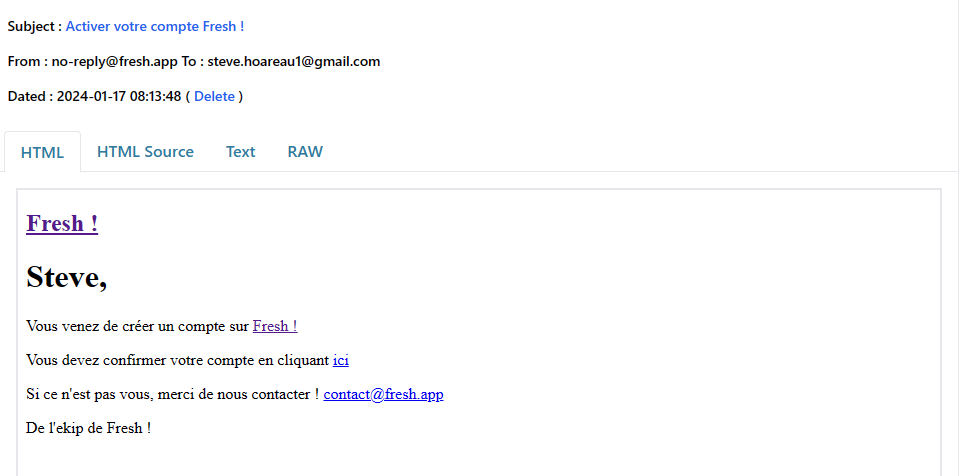
Création de la procédure.

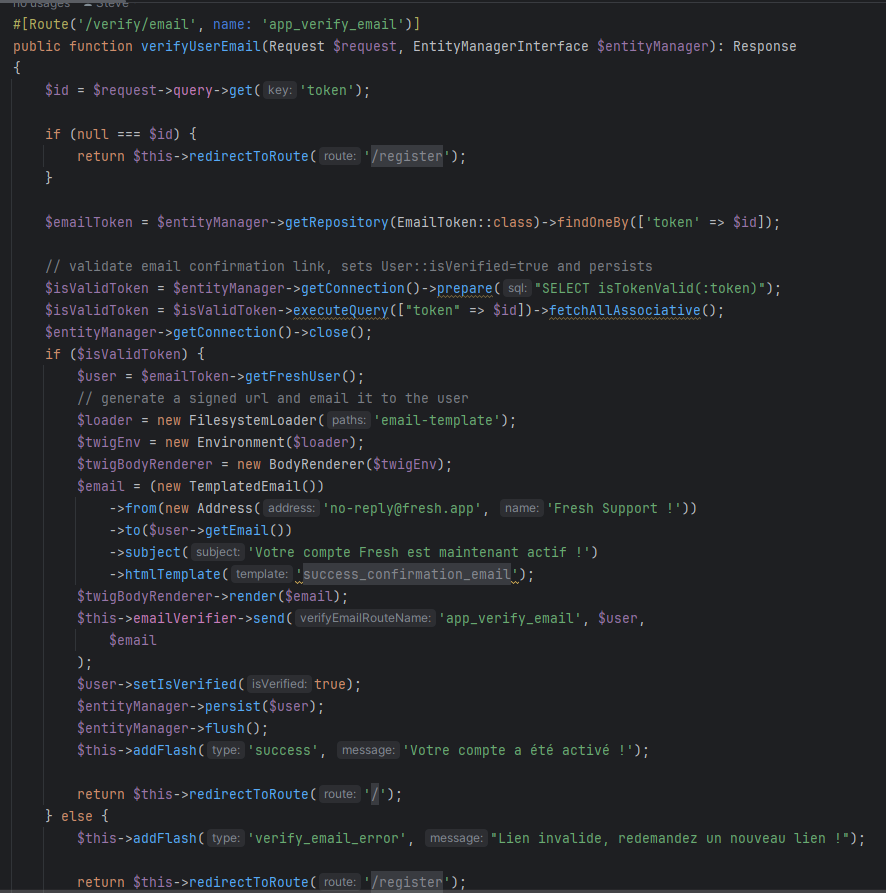
La procédure sélectionne le dernier « email\_token » généré pour l’utilisateur en prenant en compte que la date d’expiration doit être supérieure ou égale à la date d’aujourd’hui.

Rappel de la table « email\_token » :



Puis on envoie le courriel avec le token récupéré.

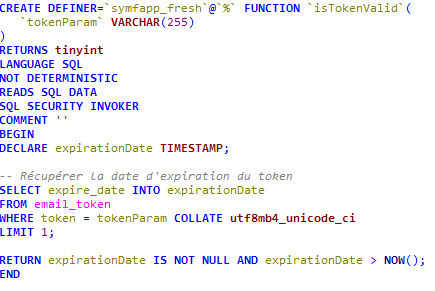


Redirige vers la route (avec l’url du site) /verify/email?token=f0f8dadf306b41ae96a73cd177187470

Lorsque l’utilisateur clique sur le lien, on vérifie que le token est valide.



On fait donc appel à la fonction « isTokenValid(varchar(255) tokenParam) » en requête préparée.



Création de la fonction.

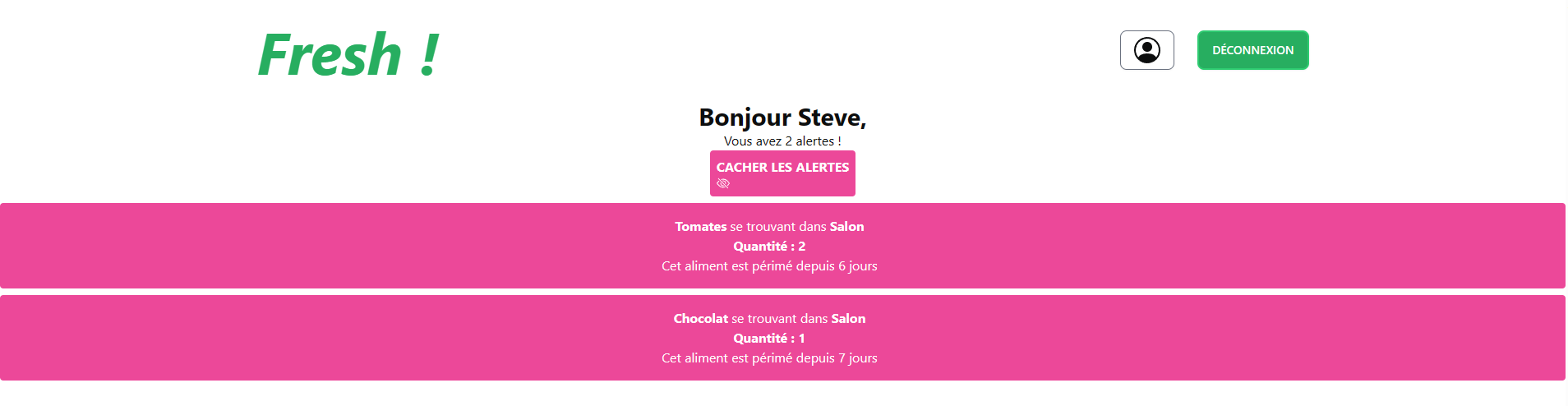
On vérifie juste pour le token (qui est unique) si sa date d’expiration est supérieur à la date d’aujourd’hui, ce qui renvoie vrai sinon faux en considérant que l’ « email\_token » est invalide.

Développement des alertes pour les aliments en phase de péremption :

L’utilisateur doit se créer un compte (sauf s’il possède déjà un) et se connecter.

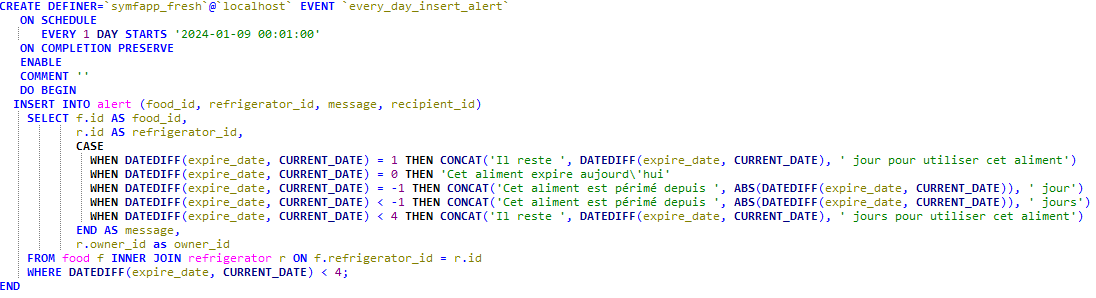
L’utilisateur doit avoir un compte vérifié.

L’utilisateur doit avoir au moins un frigo qui est créé et le frigo doit avoir au moins un aliment.



Des alertes s’affichent pendant au moins 10secondes.

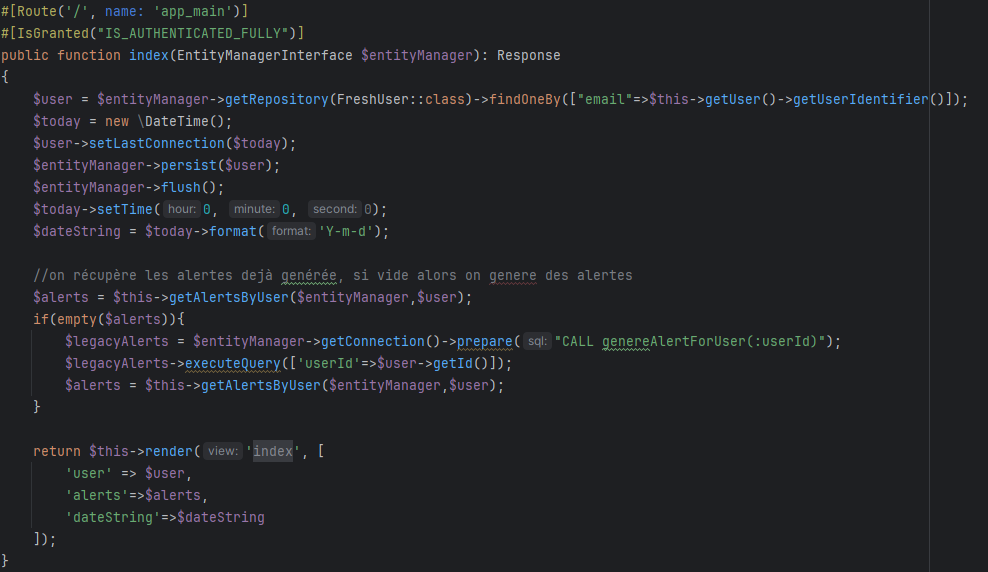
Tous les jours un événement sur la base de données MySQL est déclenché.



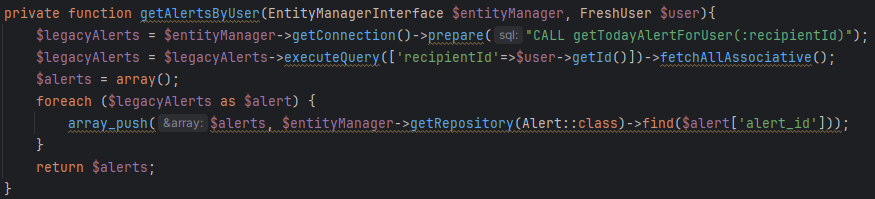
Création de l’événement.

L’événement va insérer automatiquement les aliments de tous les frigos pour ceux qui ont une date d’expiration proche de 3 jours ou moins de la date d’aujourd’hui dans la table « alert ».

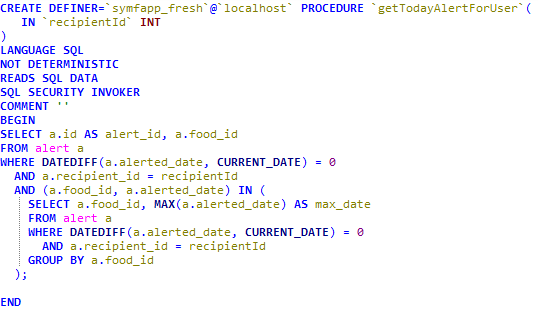
Et à chaque fois que l’utilisateur va sur la page d’accueil (page par défaut après un enregistrement ou une connexion)



On récupère les alertes pour l’utilisateur.



La procédure « getTodayAlertForUser(int recipientId) » est appelée en requête préparée.



Création de la procédure.

On récupère des alertes spécifiques dans la table « alert » pour l’utilisateur.

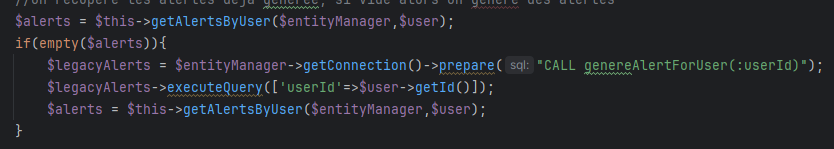
On veut les alertes qui concernent des aliments aujourd'hui (DATEDIFF(a.alerted\_date, CURRENT\_DATE) = 0).

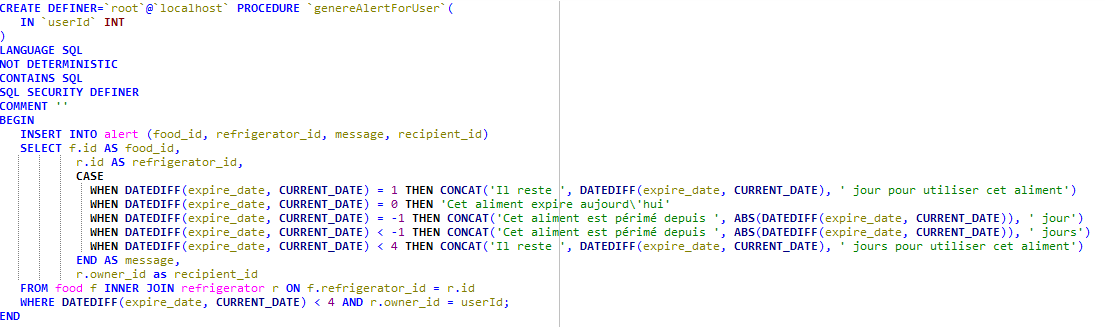
On cherche les alertes qui sont les dernières pour chaque type d'aliment. Donc, si la même alerte (même type d'aliment) a été émise plusieurs fois aujourd'hui, on récupère seulement la plus récente.

La requête utilise une sous-requête (SELECT a.food\_id, MAX(a.alerted\_date) AS max\_date ...) pour trouver la date la plus récente pour chaque type d'aliment, puis elle vérifie si l'alerte correspondante (type d'aliment et date) est également émise aujourd'hui pour l’utilisateur.

Ce qui permet d’avoir les alertes les plus récentes pour chaque aliment de l’utilisateur.

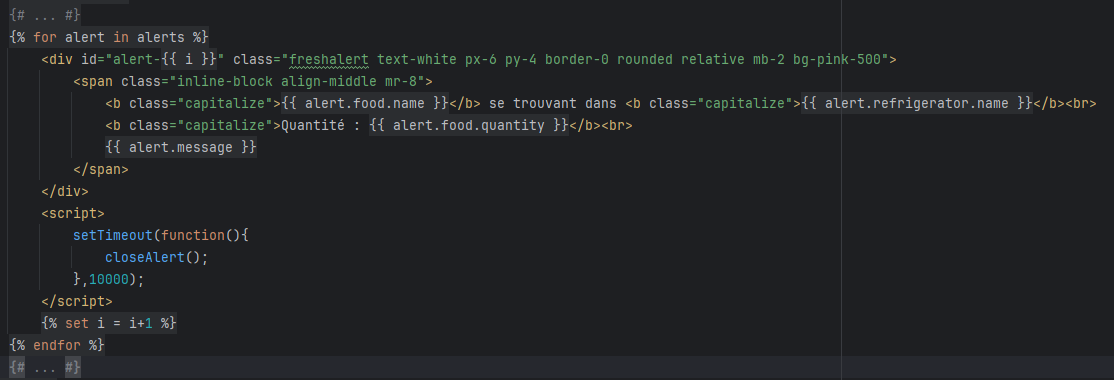
Les alertes sont générées à une heure précise. La procédure « genereAlertForUser(int userId ) » est appelée pour garantir que l’utilisateur a des alertes ou non.





Création de la procédure (Elle fait la même chose que l’événement).

Puis on va récupérer les alertes de l’utilisateur, et on les affiche.



# Présentation du jeu d’essai élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues)

Vérification du compte (email\_token):

*Rappel du trigger : before\_insert\_email\_token*

**BEGIN**

**SET** **NEW**.send\_date = **NOW**();

**SET** **NEW**.expire\_date = **NEW**.send\_date + **INTERVAL** 3 **HOUR**;

**SET** **NEW**.token = **MD5**(**CONCAT**(**NEW**.send\_date, **RAND**()));

**END**

* ***Données en entrée :***

L’identifiant d’un utilisateur.

**INSERT** **INTO** email\_token (`fresh\_user\_id`) **VALUES** (3);

* **Données attendues :**

Un entier positif.

* **Données obtenues :**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | fresh\_user\_id | send\_date | expire\_date | Token |
| 27 | 3 | 2024-01-17 15:38:33 | 2024-01-17 18:38:33 | 218b6ec2f5e8f2019fc13864bbeeb950 |

En ayant fait un **CALL** getLastTokenForUser(3);

A savoir que l’id est généré automatiquement (en incrémentation de 1).

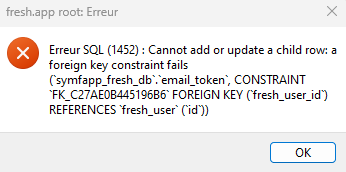
* **Si Données en entrée :**

L’identifiant d’un utilisateur qui n’existe pas.

**INSERT** **INTO** email\_token (`fresh\_user\_id`) **VALUES** (-1);

**Alors Données obtenues :**

Aucune (erreur)

****

Rappel de la fonction isTokenValid(varchar(255) tokenParam) :

**BEGIN**

**DECLARE** expirationDate **TIMESTAMP**;

*-- Récupérer la date d'expiration du token*

**SELECT** expire\_date **INTO** expirationDate

**FROM** email\_token

**WHERE** token = tokenParam **COLLATE** **utf8mb4\_unicode\_ci**

**LIMIT** 1;

**RETURN** expirationDate **IS** **NOT** **NULL** **AND** expirationDate > **NOW**();

**END**

* **Données en entrée :**

« 218b6ec2f5e8f2019fc13864bbeeb950 »

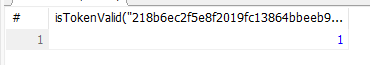
**SELECT** isTokenValid("218b6ec2f5e8f2019fc13864bbeeb950");

* **Données attendues :**

VARCHAR(255)

* **Données obtenues :**

Si la date d’expiration est supérieure à la date d’appel de la fonction :



Sinon :



* **Si données en entrée** :

« invalid\_token »

**SELECT** isTokenValid("invalid\_token");

**Alors données obtenue**:



Le token étant inexistant, la fonction retourne 0.

Insertion d’alertes :

*Rappel du trigger : before\_insert\_alert*

**BEGIN**

**SET** **NEW**.alerted\_date = **NOW**();

**END**

*Rappel du trigger : after\_insert\_food*

**INSERT** **INTO** alert (food\_id, refrigerator\_id, message, recipient\_id)

**SELECT** f.id **AS** food\_id, r.id **AS** refrigerator\_id,

**CASE**

**WHEN** **DATEDIFF**(expire\_date, **CURRENT\_DATE**) = 1 **THEN** **CONCAT**('Il reste ', **DATEDIFF**(expire\_date, **CURRENT\_DATE**), ' jour pour utiliser cet aliment')

**WHEN** **DATEDIFF**(expire\_date, **CURRENT\_DATE**) = 0 **THEN** 'Cet aliment périme aujourd\'hui'

**WHEN** **DATEDIFF**(expire\_date, **CURRENT\_DATE**) = -1 **THEN** **CONCAT**('Cet aliment est périmé depuis ', **ABS**(**DATEDIFF**(expire\_date, **CURRENT\_DATE**)), ' jour')

**WHEN** **DATEDIFF**(expire\_date, **CURRENT\_DATE**) < -1 **THEN** **CONCAT**('Cet aliment est périmé depuis ', **ABS**(**DATEDIFF**(expire\_date, **CURRENT\_DATE**)), ' jours') **WHEN** **DATEDIFF**(expire\_date, **CURRENT\_DATE**) < 4 **THEN** **CONCAT**('Il reste ', **DATEDIFF**(expire\_date, **CURRENT\_DATE**), ' jours pour utiliser cet aliment')

**END** **AS** message,

r.owner\_id **as** owner\_id

**FROM** food f **INNER** **JOIN** refrigerator r **ON** f.refrigerator\_id = r.id

**WHERE** **DATEDIFF**(expire\_date, **CURRENT\_DATE**) < 4 **AND** f.id = **NEW**.id;

* **Données en entrée :**

Un aliment.

**INSERT** **INTO** food (`refrigerator\_id`,`name`,`quantity`,`expire\_date`) **VALUES** (8,"Caramel",4, **NOW**() + **INTERVAL** 3 DAY);

* **Données attendues :**

Un aliment qui appartient à un frigo existant et ayant une date d’expiration qui est inférieur à 4 jours de la date d’insertion.

* **Données obtenues :**

Une alerte.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | food\_id | message | alerted\_date | refrigerator\_id | recipient\_id |
| 61 | 32 | « Il reste 3 jours pour utiliser cet aliment » | 2024-01-18 13:30:41 | 8 | 2 |

# Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité

Toutes les sources de la veille de sécurité sur les vulnérabilités de la plateforme ont été vérifiée sur le site <https://owasp.org/Top10>.

A01:2021 – Contrôles d'accès défaillants

Pour réduire la vulnérabilité d’accès défaillants (par changement d’url, par api, sans authentification), on a rajouté une condition dans chacune des routes de Symfony (sauf login) pour vérifier si l’utilisateur qui fait la requête est identifié.

Pour le changement d’url, Symfony traite par défaut une URL invalide.

La modification ou suppression d’éléments dans la base de données depuis Symfony est sécurisé par un token (CSRF-Token).

A02:2021 – Défaillances cryptographiques

Pour la vulnérabilité des défaillances cryptographiques il n’y a eu aucun traitement car les serveurs utilisés sont consacrés uniquement pour le développement et non la mise en production du projet.

A03:2021 – Injection

L’injection SQL est traitée par défaut par Symfony (composant Doctrine) en utilisant des requêtes préparées pour chaque traitement sur la base de données.

Les requêtes pour appeler les procédures/fonctions, sont des requêtes préparées.

A04:2021 – Conception non sécurisée

Le code de la plateforme est un code compilé, testé et fonctionnel.

A05:2021 – Mauvaise configuration de sécurité

La base de données peut être vulnérable si une personne a accès aux fichiers environnement de l’application. Cette vulnérabilité peut être réglée en délocalisant les identifiants de connexion.

A06:2021 – Composants vulnérables et obsolètes

Tous les composants sont à jour et ont été vérifiés par leurs auteurs (avec test d’intégrité, test unitaire avec une plateforme telle que Jenkins), mais on peut toujours trouver une faille.

En conclusion, la plateforme et sa base de données respecte la plupart des règles de sécurités et empêche-la plupart des failles. Les failles restantes doivent être régler avant la mise en production finale du projet.

Le projet ne contient pas de fichiers de journalisation permettant d’avoir un historique des requêtes.

# Description d’une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet

Le site web [Stack Overflow](https://stackoverflow.com/) a été utilisé pour la majorité des recherches effectuées. La situation de travail pouvant justifier cette utilisation est pour automatiser la génération de token pour la vérification d’un compte utilisateur. Pour le trigger `before\_insert\_email\_token`, la fonction RAND() a été trouvée.

L’intelligence artificielle [Chat-GPT](https://chat.openai.com/) a été utilisé pour les différents blocages de code.

La documentation [daisyUi](https://daisyui.com/components/) a été utilisé pour trouver les bonnes classes a ajouté pour le front-end de l’application.

La documentation [Symfony](https://symfony.com/) a été utilisé pour l’envoie de mail et les différents blocage de code.