le cnam
Paris

CONCEPTION ET DÉVELOPPEMENT D'UNE APLICATION DE DOMOTIQUE

YOHAN VOISIN
MAMITIANA RAJAONSON

le cnam
Paris

Licence professionnelle en informatique

Web, Mobile et Business Intelligence

Rapport de projet

Conception et développement d'une application de domotique

par

Yohan VOISIN

et Mamitiana RAJAONSON

Présenté le 08 mars 2023

Enseignante : Tatiana AUBONNET

Année universitaire 2022-2023

Table des matières

Introduction

Nous vivons aujourd'hui dans une société où le digital, le numérique sont synonymes de performance et d'amélioration continue. Les outils nouvellement développés se doivent d'apporter du confort dans notre vie de tous les jours. En raison de ses impacts positifs sur la qualité de nos vies, l'utilisation de la domotique permet une meilleure gestion de l'énergie, une sécurité renforcée dans les habitats et un réel contrôle des équipements mis à disposition.

Le développement de la domotique a connu une forte croissance ces dernières années, avec des entreprises telles que Google, Amazon, Apple, Philips Hue ou encore Somfy proposant chacune leur solution logicielle ou matérielle pour la maison connectée.

Dans le cadre de notre projet Java, nous avons décidé d'explorer ce monde qui est lié à la domotique en concevant et en développant une application Java afin de relier et centraliser le contrôle et la gestion des équipements d'une habitation connectée.

Nous allons commencer par une phase d'analyse de conception, suivie de la présentation de notre environnement technique de développement et enfin nous aborderons l'implémentation réelle de notre projet.

Partie I : Analyse et conception

1. L'expression du besoin

Nous cherchons une solution de domotique simple, intuitive qui nous permettra de surveiller et de contrôler nos appareils électroniques à distance. Nous souhaitons une application qui nous permettra de réguler la température, de contrôler les lumières, les volets et d'automatiser certaines de ces opérations en fonction d'un horaire défini. Nous avons également besoin d'une solution qui soit adaptable, pour que nous puissions intégrer de nouveaux appareils à l'avenir.

Nous voulons nous assurer que notre système de domotique est protégé et que seules les personnes autorisées ont accès à nos équipements. Pour cela, l'accès aux équipements, par la gestion de notre installation doit être protégé par un compte utilisateur. L'utilisateur doit ensuite être en mesure de visualiser facilement ses équipements ainsi que les pièces qui composent son domicile. Les données doivent être enregistrées, stockées de manière pérenne pour garantir la sécurité et la fiabilité de notre système. Nous souhaitons également une solution flexible, qui nous permette d'ajouter, de supprimer, de modifier facilement nos équipements et pièces selon nos besoins. Nous sommes convaincus que la solution de domotique doit être simple à utiliser, sûre et facilement adaptable à nos besoins futurs.

Enfin, nous aimerions que l'application de domotique intègre une fonctionnalité de visualisation des données météo actuelles et des prévisions à venir. Cela nous permettrait d'adapter certaines de nos actions quotidiennes, comme l'ouverture ou la fermeture des volets, en fonction des conditions météorologiques.

Par exemple, si nous savons qu'il va pleuvoir dans l'après-midi, nous pourrions programmer l'application pour fermer automatiquement les fenêtres et les volets à l'avance afin d'éviter que la pluie ne rentre dans la maison. De même, si nous savons qu'il va faire chaud, nous pourrions programmer l'application pour ouvrir les volets et les fenêtres à certaines heures de la journée afin de faire entrer de l'air frais, de réduire la température à l'intérieur de la maison.

En somme, la visualisation des données météo sur l'écran d'accueil de l'application nous permettrait de mieux adapter nos gestes, nos habitudes à notre environnement et d'optimiser

l'utilisation de nos équipements domotiques. Cela contribuerait également à améliorer le confort et la qualité de vie dans notre maison.

2. Diagramme de cas d'utilisation

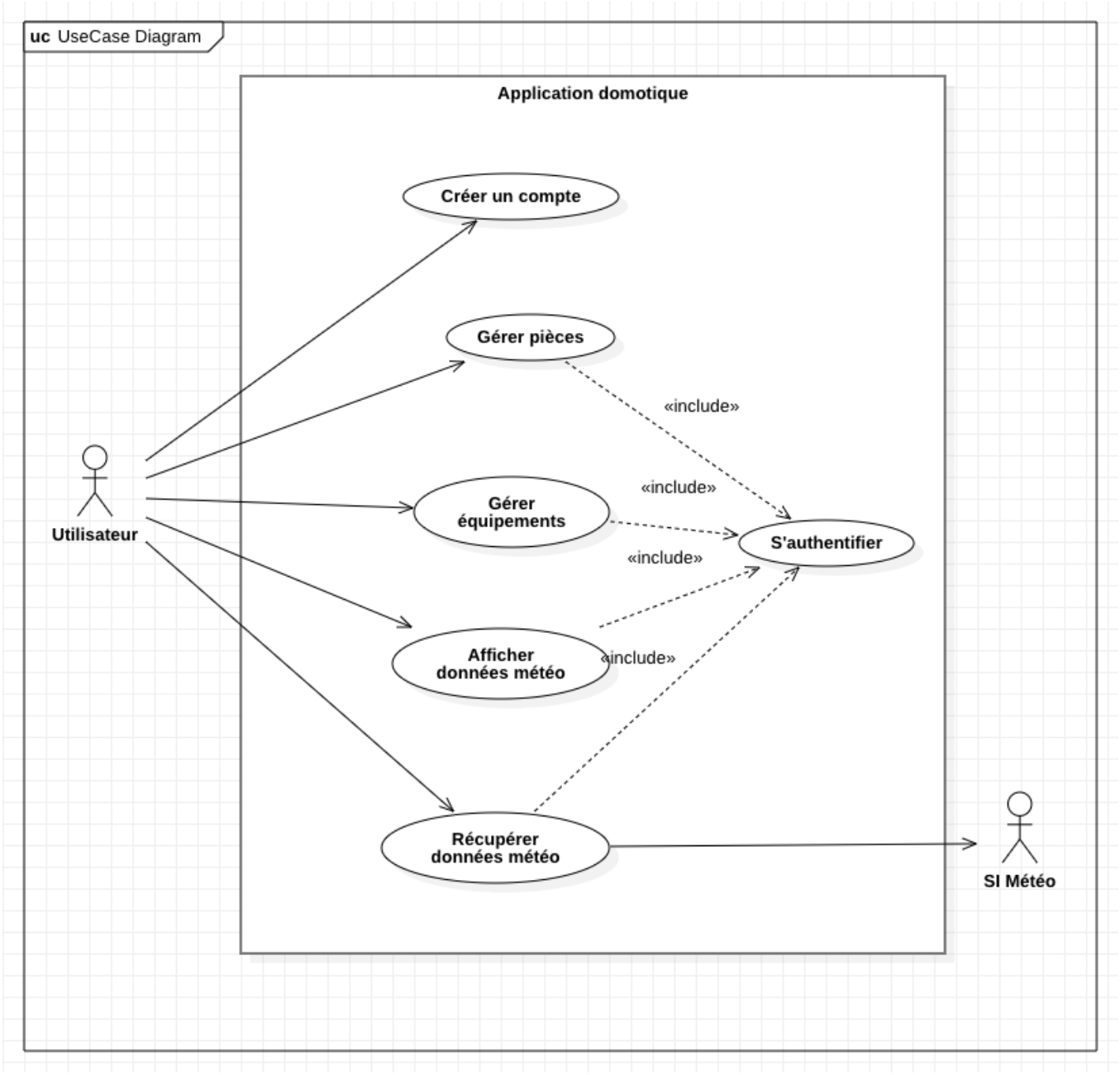

Figure 1 - Diagramme de cas d'utilisation

3. Diagramme de séquence : processus d'ajout d'un équipement

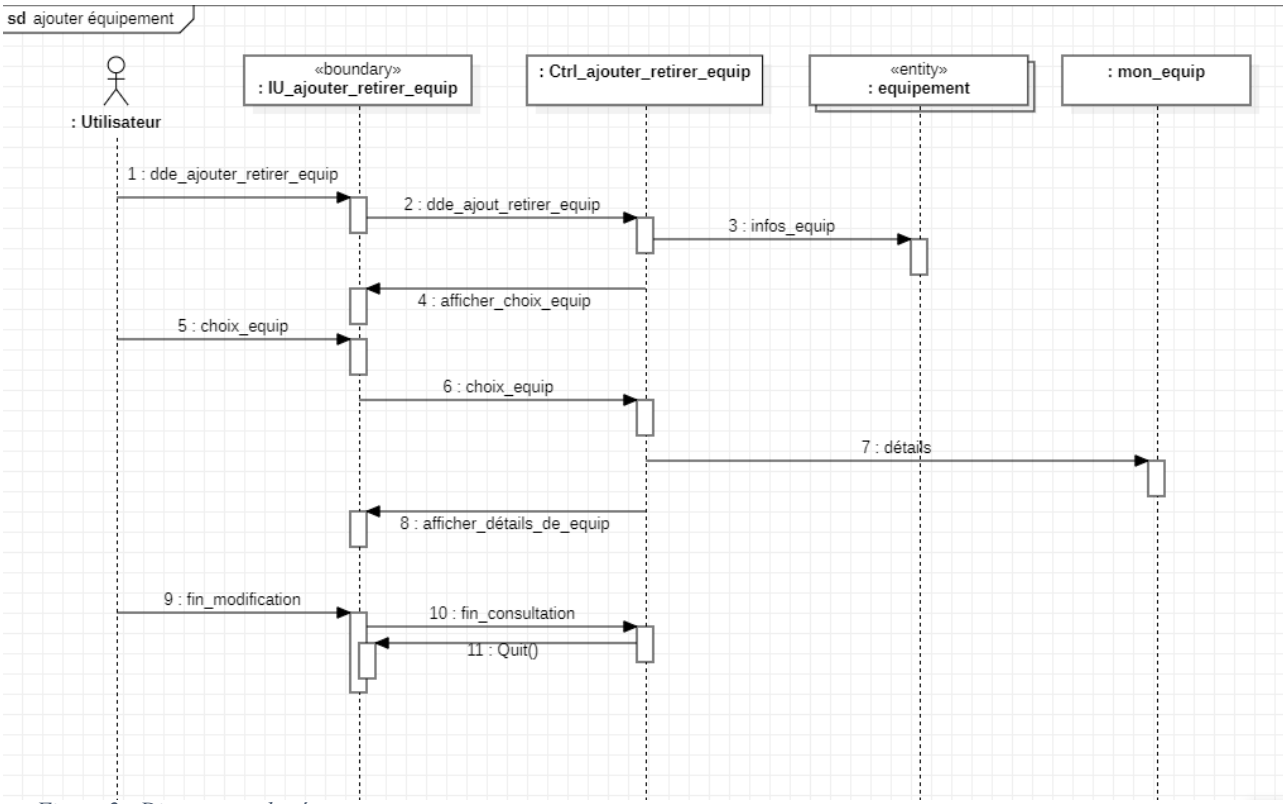

Figure 2 - Diagramme de séquence

4. Diagramme de classe

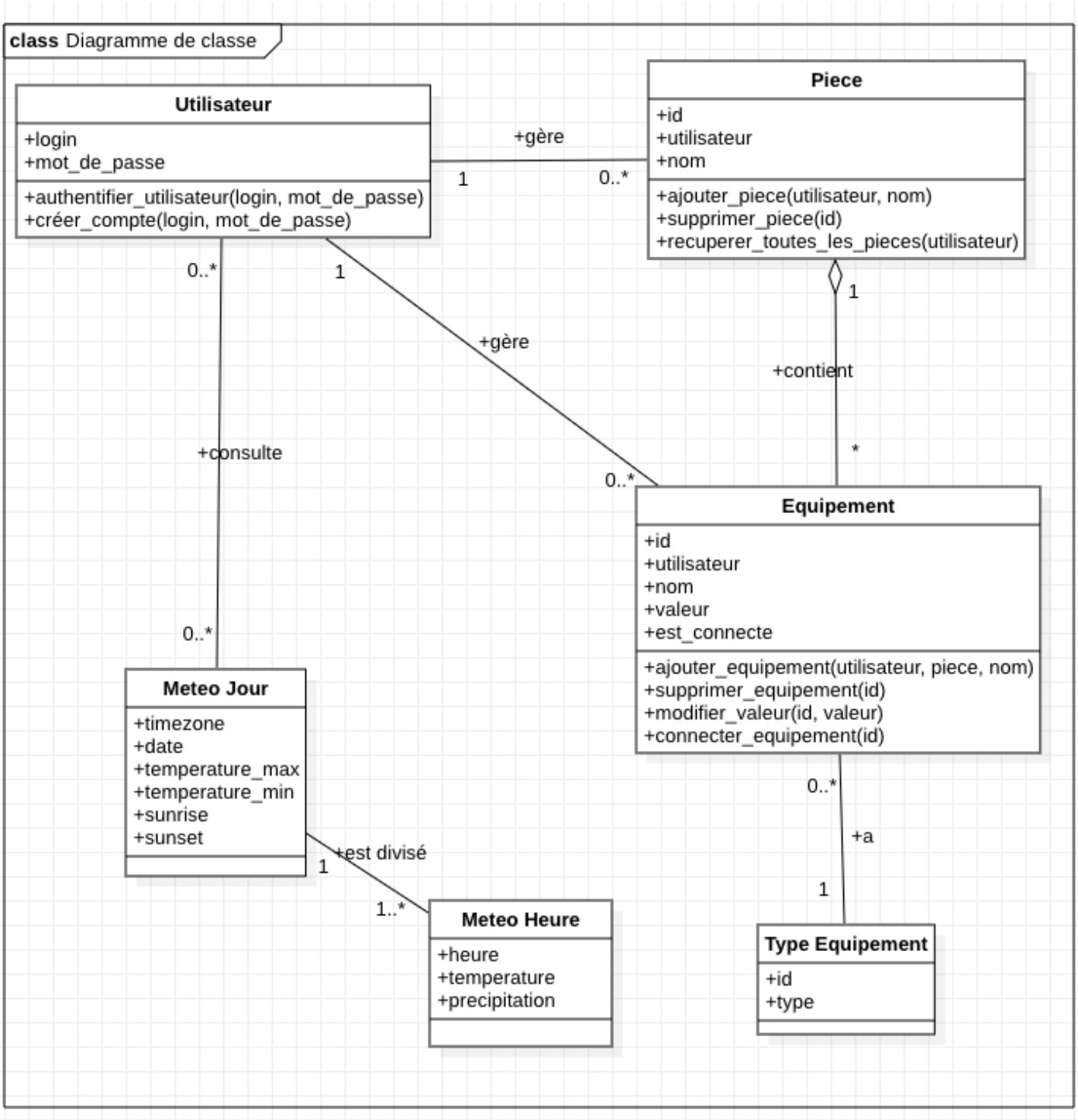

Figure 3 - Diagramme de classe

5. Spécifications ouvertes et fermées

- Authentification et autorisation :

Le site web doit permettre l'authentification et l'autorisation des utilisateurs, avec la possibilité de créer un compte et de gérer les rôles et les autorisations.

- Base de données :

Le site web doit être alimenté par une base de données extensible, telle que MySQL ou PostgreSQL, pour stocker les informations utilisateur, ses équipements et autres données.

- Interface web :

La plateforme du site web doit être développée en utilisant des technologies web, telles que HTML, CSS, Java et Java EE, et être compatible avec les navigateurs web couramment utilisés.

- Sécurité :

Le site web doit être conçu avec des mesures de sécurité efficaces pour protéger les données utilisateur, éviter les attaques de piratage, et garantir la confidentialité et l'intégrité des données.

- Performances :

Le site web doit être optimisé pour des temps de chargement rapides et des performances élevées, la compression et l'optimisation d'images.

- Intégration API :

Le site web doit être capable d'intégrer des API tiers, tel que les outils de récupération de données météo.

Partie II : Choix de l'environnement de développement

1. IDE

Nous avons choisi d'utiliser NetBeans 16 en tant qu'IDE pour notre projet. Pour le développement local, notre application tourne actuellement sur un serveur Apache Tomcat version 10.1.5.

2. Versioning du code

Afin de gérer efficacement les différentes versions de notre code source au fil du temps, pour faciliter la collaboration à distance et pour suivre les modifications apportées au code, nous avons décidé d'utiliser Git avec un dépôt sur GitHub.

Le projet est disponible à l'adresse GitHub suivante : https://github.com/mrajaonson/lpacsid-java-domotique et est disponible sous licence GPLv3 https://www.gnu.org/licenses/gpl-3.0.html

3. Interface utilisateur

Pour développer l'interface web utilisateur, nous avons opté pour la plateforme Java EE qui fournit un ensemble de spécifications pour les développeurs telles que la gestion des services web, la connectivité aux bases de données, la gestion des sessions utilisateurs, la sécurité et la gestion des transactions pour construire des applications évolutives.

4. Base de données

Pour le stockage et la gestion des données, nous avons choisi d'utiliser MySQL en version 8 associé à PhpMyAdmin en interface web. Lors de la création des tables, nous avons défini les contraintes de clés étrangères pour garantir l'intégrité de la base de données.

```
CREATE TABLE `piece` (
    `id` INT NOT NULL UNIQUE AUTO_INCREMENT,
    `utilisateur` VARCHAR(20) NOT NULL,
    `nom` VARCHAR(20) NOT NULL,
    PRIMARY KEY (`utilisateur`, `nom`),
    FOREIGN KEY (`utilisateur`) REFERENCES users(`login`)
);
```

Figure 4 - SQL création de la table "PIECE"

Partie III : Implémentation

1. Interface web utilisateur

Après la phase d'analyse et de conception du projet, nous avons commencé le développement de l'application. Nous avons opté pour Java EE avec son architecture MVC (Modèle-Vue-Contrôleur) et une architecture DAO (Data Access Object) pour gérer les transactions avec notre base de données.

Pour améliorer l'expérience utilisateur, nous avons choisi d'utiliser un framework css pour le développement de l'interface web, nous avons implémenté **Bulma** dans la version **0.9.4** (https://bulma.io/), il s'agit d'un framework CSS open source, moderne et simple, et qui fonctionne sans code JavaScript. Nous avons également choisi d'utiliser un framework css pour inclure plus simplement les icônes, nous avons optés pour **Bootstrap Icons** en version **1.10.0**, également open source. Nous avons choisi d'importer les fichiers css dans notre projet et de ne pas utiliser de lien internet. Au total, les fichiers font 588 Ko.

Figure 5 - Structure des fichiers css

2. Architecture DAO

Nous avons choisi d'utiliser l'architecture MVC (Modèle-Vue-Contrôleur) et l'approche DAO (Data Access Object) pour ce projet. L'architecture MVC, utilisée avec Java EE, permet de séparer les différentes responsabilités des différentes parties de l'application. Ainsi, nous pouvons centraliser dans le "modèle" la logique associée à la base de données. Java EE permet de séparer facilement la "vue" avec l'intégration des JSP, la partie "contrôleur" gère les interactions entre la vue et le modèle grâce aux servlets.

Cette approche nous permet de séparer les responsabilités des différentes parties de l'application, ce qui facilite le développement, la maintenance et l'évolution de l'application.

L'architecture DAO nous permet de séparer encore davantage la logique de l'application et l'accès aux données, ce qui facilite les évolutions futures et la lisibilité du code. Le "modèle" n'a en effet plus connaissance de la base de données, le DAO agit comme interface entre ces deux parties. Nous avons donc mis en place des interfaces DAO qui définissent les méthodes utilisées pour accéder aux données telle la lecture, la création, la mise à jour et la suppression. Nous avons ensuite ajouté des classes qui implémentent ces interfaces DAO, fournissant une implémentation concrète des méthodes définies. Enfin, nous avons une fabrique DAO (ou DAO Factory) qui détermine l'implémentation DAO qui doit être créée. Les DAO Factory sont appelés dans la méthode « init » de chaque servlet de notre application.

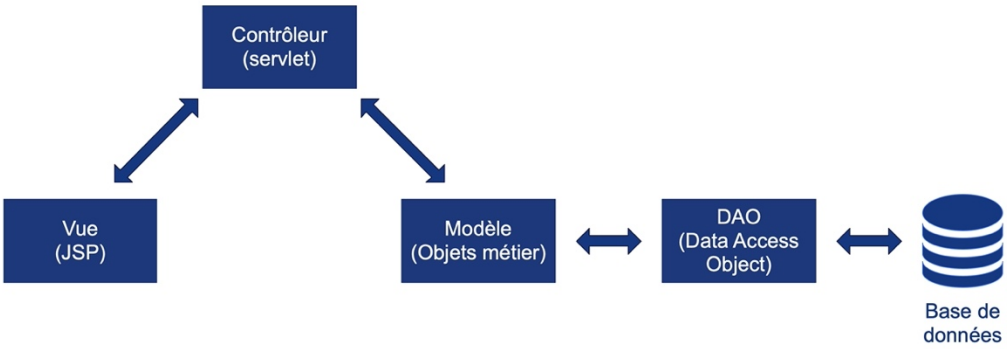

Figure 6 - Schéma de l'architecture DAO

Pour chaque entité de notre base de données, nous avons développé des objets beans pour faciliter les transactions entre l'application et la base de données. La connexion en elle-même avec la base est assurée par le paquet mysql-connector-j-8.0.32.jar ajouté en dépendance dans l'application.

```java
public class Piece {

    private String id;
    private String utilisateur;
    private String nom;

    public Piece(int id, String utilisateur, String nom) {
        this.id = "P" + id;
        this.utilisateur = utilisateur;
        this.nom = nom;
    }

    public String getId() {
        return id;
    }

    public void setId(int id) {
        this.id = "P" + id;
    }

    public String getUtilisateur() {
        return utilisateur;
    }

    public void setUtilisateur(String utilisateur) {
        this.utilisateur = utilisateur;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }
}
```

Figure 7 - Classe "Piece"

```java
public interface PieceDao {

    public void createPiece(String utilisateur, String nom);

    public Piece readPiece(String utilisateur, String nom);

    public void updatePiece(Piece piece);

    public void deletePiece(String utilisateur, String nom);

    public void deletePieceById(String localId);

    public List<Piece> getAllPieces(String utilisateur);

}
```

Figure 8 - Interface DAO de la classe Piece

```java
@Override
public void createPiece(String utilisateur, String nom) {
    Connection connexion = null;
    PreparedStatement preparedStatement = null;

    try {
        connexion = daoFactory.getConnection();
        String query = "INSERT INTO piece (utilisateur, nom) VALUES(?, ?);";
        preparedStatement = connexion.prepareStatement(query);

        preparedStatement.setString(1, utilisateur);
        preparedStatement.setString(2, nom);

        preparedStatement.executeUpdate();

        System.out.println("INSERT INTO piece " + utilisateur + " " + nom);
    } catch (SQLException e) {
        try {
            if (connexion != null) {
                connexion.rollback();
            }
        } catch (SQLException e2) {
        }
        Logger.getLogger(PieceDaoImpl.class.getName()).log(Level.SEVERE, null, e);
    } finally {
        try {
            if (preparedStatement != null) {
                preparedStatement.close();
            }
            if (connexion != null) {
                connexion.close();
            }
        } catch (SQLException e) {
            Logger.getLogger(PieceDaoImpl.class.getName()).log(Level.SEVERE, null, e);
        }
    }
}
```

Figure 9 - Implémentation de la méthode de création d'une pièce

3. Servlets

3.1. Authentification

La servlet « Auth » gère les requêtes associées à l'authentification, la gestion de la session utilisateur et des cookies. Lorsque l'utilisateur arrive sur l'écran d'authentification, il est invité à renseigner son nom d'utilisateur et son mot de passe. Une fois ces informations soumises, la servlet récupère les champs correspondants, utilisant le DAO utilisateur pour vérifier qu'en base l'utilisateur existe et si le mot de passe renseigné correspond bien à celui enregistré en base.

- Si l'authentification est réussie, la servlet enregistre dans les attributs de la session le nom de l'utilisateur, crée un cookie contenant également son nom d'utilisateur avec une durée de validité définie à trois jours. Ce cookie est ensuite renvoyé au client. Enfin, l'utilisateur est redirigé vers la page d'accueil de l'application, où il peut accéder aux fonctionnalités de l'application réservées aux utilisateurs connectés.

- Sinon, l'authentification échoue, l'utilisateur est redirigé vers la même page d'authentification et un message d'erreur apparaît.

Dans la méthode « doGet » de cette servlet, on vérifie dans un premier temps si le client nous communique un cookie valide ou si un utilisateur existe dans les attributs de session.

- Si oui, l'utilisateur est automatiquement redirigé vers la page d'accueil, où il peut accéder à ses fonctionnalités.
- Sinon, il est invité à se réauthentifier avec son identifiant et mot de passe. Cette méthode est donc utilisée pour vérifier que l'utilisateur est bien authentifié avant d'autoriser l'accès aux pages réservées aux utilisateurs connectés.

```java
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    HttpSession session = request.getSession();
    ServletContext contexte = getServletContext();
    RequestDispatcher dispatcher;

    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for (Cookie cookie : cookies) {
            if (cookie.getName().equals("user")) {
                session.setAttribute("isAuth", true);
                session.setAttribute("user", cookie.getValue());
            }
        }
    }

    boolean isAuth = this.isAuth(session);

    if (isAuth) {
        response.sendRedirect("Home");
    } else {
        dispatcher = contexte.getRequestDispatcher("/jsp/auth.jsp");
        dispatcher.forward(request, response);
    }
}
```

Figure 10 - Méthode doGet de la servlet Auth

```java
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    HttpSession session = request.getSession();
    ServletContext contexte = getServletContext();
    RequestDispatcher dispatcher;

    String username = request.getParameter("login");
    String password = request.getParameter("pass");

    // Authentifaction du user
    Boolean isAuth = (Boolean) userDao.validateUser(username, password);

    System.out.println("isAuth " + isAuth);

    if (isAuth) {
        session.setAttribute("isAuth", isAuth);
        session.setAttribute("user", username);

        Cookie cookie = new Cookie("user", username);
        // Validité du cookie en secondes : 60s * 60m * 24h * 3j
        cookie.setMaxAge(60 * 60 * 24 * 3);
        response.addCookie(cookie);

        response.sendRedirect("Home");
    } else {
        request.setAttribute("loginError", "Login et/ou mot de passe incorrect");
        dispatcher = contexte.getRequestDispatcher("/jsp/auth.jsp");
        dispatcher.forward(request, response);
    }
}
```

Figure 13 – Méthode doPost de la servlet Auth

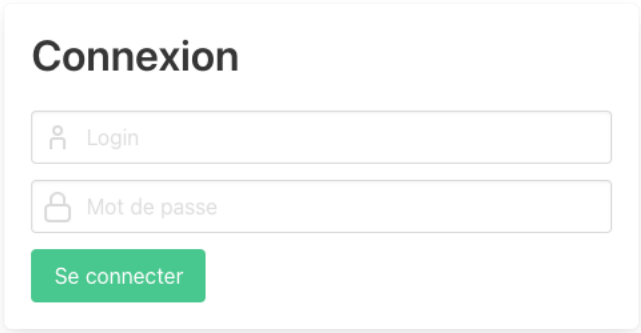

Figure 11 – Écran de connexion de l'application

Figure 12 - Message d'erreur en cas de saisie incorrecte

3.2. Accueil

Cette servlet est chargée de gérer les requêtes associées à la page d'accueil de l'application, qui regroupe l'ensemble de la gestion des équipements domotiques.

Tout d'abord, à chaque requête « doGet » et « doPost », on vérifie systématiquement si l'utilisateur est connecté grâce à un cookie valide ou aux attributs de la session. Si ces éléments ne sont pas valides, le client est automatiquement redirigé vers la page d'authentification.

Dans un second temps, nous récupérons les données météorologiques du jour ainsi que les prévisions pour les trois prochains jours en base de données. Ces informations, sous forme d'objet, sont communiquées aux JSP pour affichage.

Nous récupérons également dans la base de données tous les équipements liés à l'utilisateur. Pour chaque équipement, nous récupérons sa valeur ou son état et ses informations. Les équipements sont ensuite affichés par type et pour chaque type, des contrôleurs spécifiques sont affichés à l'écran.

- Par exemple, pour les équipements de type « lumière », un switch est affiché, pour les équipements de type « volet », des boutons haut et bas sont affichés, et pour les radiateurs, un contrôle de température est affiché. Chaque bouton est associé à un formulaire de contrôle, et à chaque action « post », la valeur est retransmise à la base de données. Ce mécanisme simule la connexion à un pont qui permettrait de communiquer avec des équipements physiques.

Dans la JSP de l'accueil, nous retrouvons également un en-tête qui affiche les différents menus de l'application, un bouton de déconnexion et un bouton qui permet de récupérer les dernières données météo. Lors du clic sur le bouton de déconnexion, le cookie utilisateur est défini à null, sa durée de validité est modifiée à 0 seconde, nous renvoyons ce cookie à l'utilisateur pour écraser le précèdent. Le client est ensuite redirigé sur la page d'authentification.

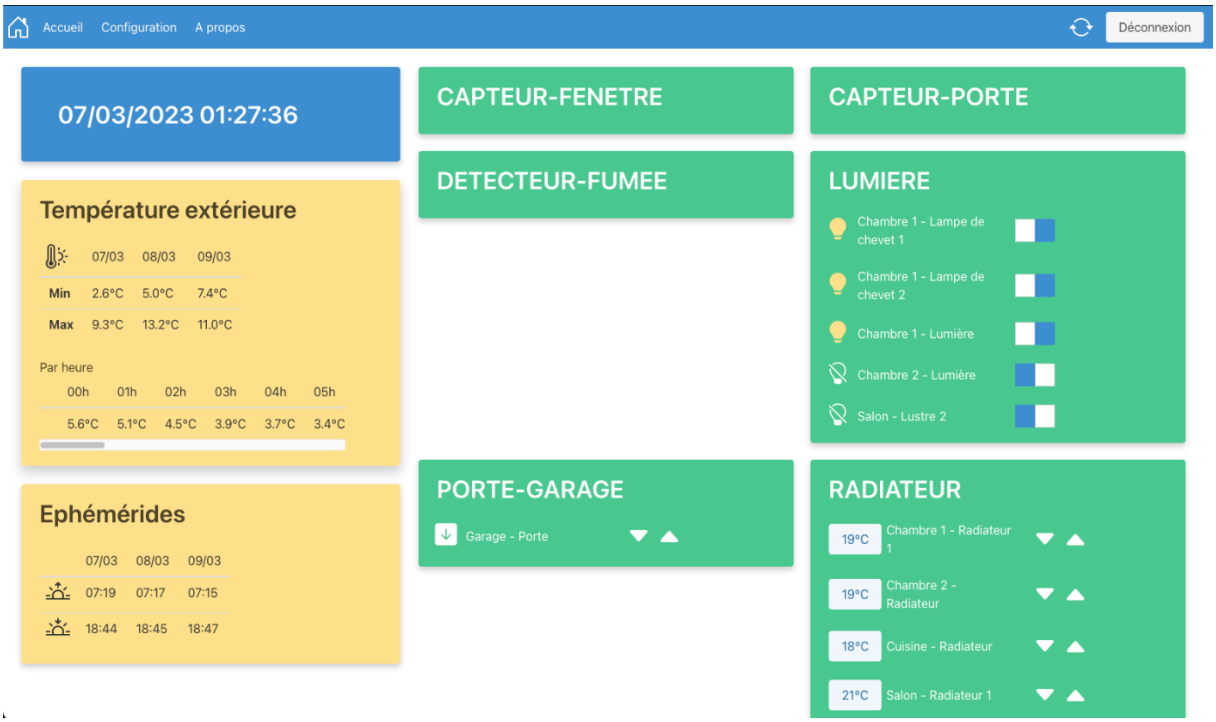

Figure 14 - Écran principale de l'application

3.3. Configuration

La servlet de configuration est une partie importante de l'application car elle permet à l'utilisateur de gérer les différentes pièces, les équipements de sa maison connectée. Elle est accessible à partir de la page d'accueil et contient deux formulaires permettant à l'utilisateur d'ajouter de nouvelles pièces ou de nouveaux équipements à son système domotique.

Pour ce faire, la servlet de configuration utilise les DAO pour communiquer avec la base de données, elle peut lire ou supprimer les pièces et les équipements liés à l'utilisateur.

En outre, la servlet de configuration renvoie également la liste de toutes les pièces et de tous les équipements de l'utilisateur, ce qui lui permet de les visualiser, de les gérer facilement. Pour chaque élément, l'utilisateur a la possibilité de le supprimer.

En résumé, la servlet de configuration est une partie clé de l'application, car elle permet à l'utilisateur de gérer les différents éléments de son système domotique, tels que les pièces et les équipements, en utilisant des formulaires simples et conviviaux.

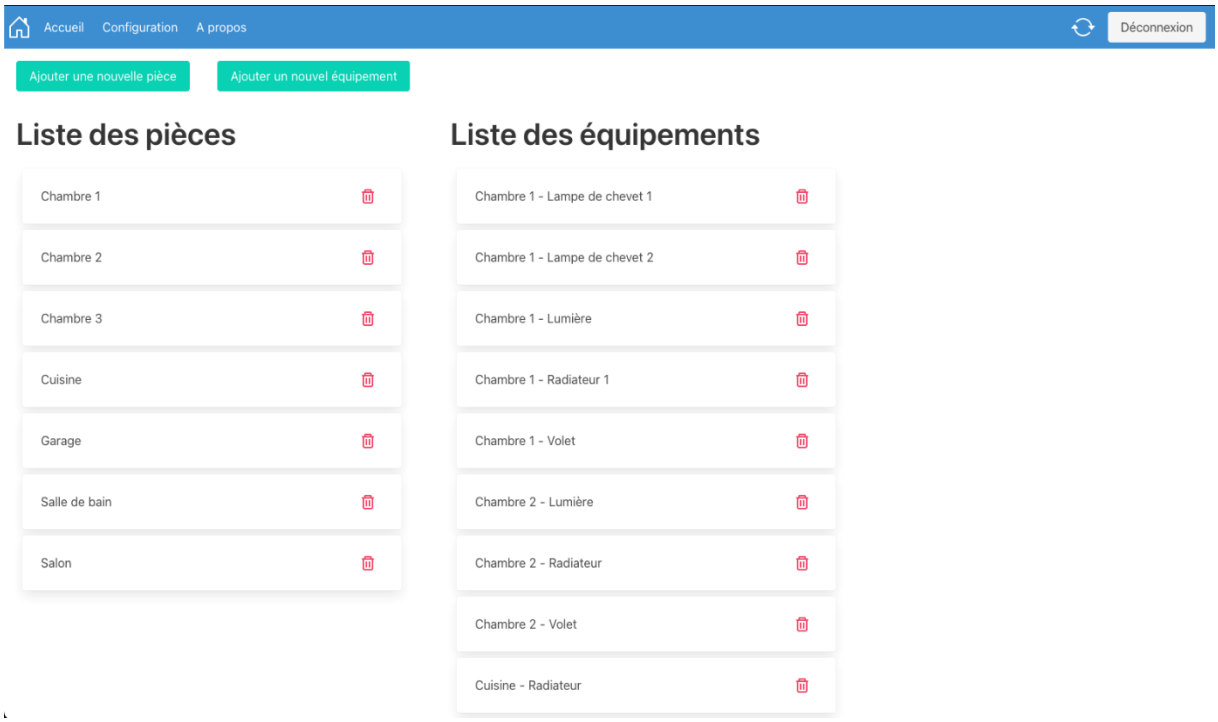

Figure 15 - Écran de configuration des équipements

4. API météo

Afin de fournir des données de météo fiable et à jour, notre application de domotique est connectée à une API de données météo. Nous avons choisi d'utiliser Open-Meteo https://open-meteo.com/, il s'agit d'une API météo open source. Sur l'écran d'accueil nous avons la possibilité de choisir les éléments dont on a besoin, en l'occurrence nous avons choisi de récupérer les prévisions météo sur 3 jours, les prévisions météo par heure de la journée et les heures de lever et de coucher du soleil sur 3 jours également.

Lorsqu'on sollicite l'API, on obtient en retour les données sous forme de JSON. Nous traitons ensuite ce JSON pour l'enregistrer en base de données. Cela nous permet de réutiliser ces données sans avoir à solliciter l'API à chaque fois, ce qui réduit considérablement le temps de traitement.

Afin de gérer plus simplement les données JSON que nous recevons, nous avons mis en place un utilitaire de JSON basé sur le paquet Gson de Google. Dans cette classe, nous avons mis en place différentes méthodes pour transformer des valeurs JSON en données String utilisables par notre application et pour récupérer de façon simple une valeur dans un JSON à partir d'une clé.

Nous avons regroupé ces traitements dans le package « utils » de notre application.

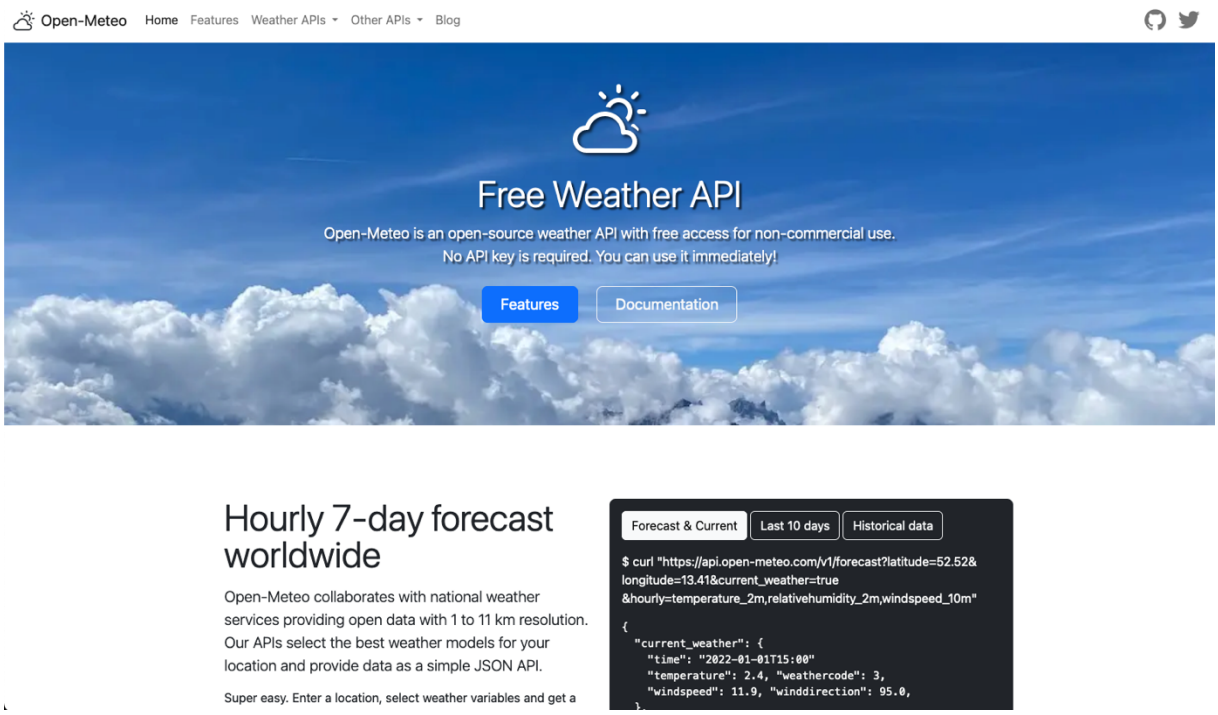

Figure 16 - Écran d'accueil d'Open-Meteo

```java
    public void fetchMeteo() throws IOException {
        String url = "https://api.open-meteo.com/v1/meteofrance?latitude=48.85&longitude=2.35&
hourly=temperature_2m,precipitation&
daily=weathercode,temperature_2m_max,temperature_2m_min,sunrise,sunset&timezone=auto";
        URL obj = new URL(url);
        HttpURLConnection con = (HttpURLConnection) obj.openConnection();
        con.setRequestMethod("GET");

        int responseCode = con.getResponseCode();
        System.out.println("fetch meteo " + responseCode);

        BufferedReader in;
        in = new BufferedReader(new InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuilder responseContent = new StringBuilder();
        while ((inputLine = in.readLine()) != null) {
            responseContent.append(inputLine);
        }
        in.close();
        System.out.println("fetch meteo " + responseContent);

        this.data = responseContent.toString();
    }
```

Figure 17 - Méthode de récupération des données météo via l'API

5. Schedule

Lorsque l'utilisateur arrive sur l'écran d'accueil de l'application, il a la possibilité de forcer la récupération et la mise à jour des données météo.

Afin de fournir une meilleure expérience utilisateur, nous avons automatisé cette tâche en créant un planificateur de tâches au sein de notre application Java EE. Pour cela, nous avons développé une classe qui implémente l'interface ServletContextListener.

L'interface ServletContextListener nous permet de détecter les événements de cycle de vie d'un ServletContext (création ou destruction) dans une application web. Dans notre cas, nous l'utilisons pour créer un planificateur de tâches. Dans cette classe, nous utilisons les utilitaires Java « Timer » pour créer un « schedule ». Cette implémentation permet de planifier l'exécution d'une commande ou méthode donnée à des moments précis et de manière récurrente. Nous avons ainsi créé un « schedule » qui définit une date d'exécution fixée à 23h00 tous les jours, ainsi qu'une action à exécuter. Pour l'action à exécuter, nous avons programmé l'appel à l'API météo Open-Meteo et l'enregistrement en base de la réponse obtenue.

Ainsi, chaque jour à 23h00, notre application va récupérer les prévisions météo sur 3 jours, les prévisions météo par heure de la journée et les heures de lever et de coucher du soleil sur 3 jours également. Les données récupérées seront ensuite stockées en base de données pour être réutilisées ultérieurement.

En utilisant un planificateur de tâches, nous offrons une expérience utilisateur plus agréable en évitant à l'utilisateur d'avoir à forcer manuellement la mise à jour des données météo. De plus, cela permet de garantir la mise à jour régulière des données, sans intervention de l'utilisateur, et d'améliorer la fiabilité de notre application.

```java
private Date getDelay() {
    // Date et heure actuelle
    Calendar calendar = Calendar.getInstance();
    calendar.set(Calendar.HOUR_OF_DAY,23);
    calendar.set(Calendar.MINUTE, 0);
    calendar.set(Calendar.SECOND, 0);

    // Si heure actuelle passée, passer à heure du jour suivant
    if (calendar.getTime().compareTo(new Date()) < 0) {
        calendar.add(Calendar.DAY_OF_MONTH, 1);
    }

    return calendar.getTime();
}

private class DailyTask extends TimerTask {
    private final MeteoDailyDao meteoDailyDao1;
    private final MeteoHourlyDao meteoHourlyDao1;

    private DailyTask(MeteoDailyDao meteoDailyDao1, MeteoHourlyDao meteoHourlyDao1) {
        this.meteoDailyDao1 = meteoDailyDao1;
        this.meteoHourlyDao1 = meteoHourlyDao1;
    }

    @Override
    public void run() {
        try {
            System.out.println("Début de la récupération quotidienne de l'api météo");
            // Récupération et persistance en base des données météo
            Meteo meteo = new Meteo(this.meteoDailyDao1, this.meteoHourlyDao1);
            meteo.fetchMeteo();
            meteo.persistMeteo();
        } catch (ParseException | IOException ex) {
            Logger.getLogger(MeteoScheduler.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Figure 18 - Méthodes de la classe permettant de planifier la récupération des données météo

Conclusion et perspectives

En conclusion, la conception et le développement de cette application de domotique ont été une expérience passionnante et enrichissante pour notre équipe. Nous avons pu mettre en pratique des connaissances en matière de conception orientée objet, d'architecture de logiciels et de développement Java et Java EE.

Notre application de domotique, telle qu'elle est développée actuellement, se concentre sur l'interface graphique d'une architecture d'application de domotique. Nous avons fait le choix de faire abstraction de la partie « gateway » permettant de communiquer par différent protocoles réseaux, tels que Zigbee, aux équipements IOT d'une maison connectée. L'implémentation des actions sur notre base de données nous a permis de simuler ce mécanisme de communication.

L'interface permet ainsi à l'utilisateur de contrôler sa maison intelligente de manière simple et intuitive, grâce à une interface utilisateur conviviale et des fonctionnalités telles que la gestion des pièces et des équipements, la récupération des données météo, la planification de tâches et la gestion de l'authentification et des sessions utilisateur.

Nous avons également mis en place des mécanismes de gestion des cookies et des sessions utilisateur pour garantir la confidentialité et l'intégrité des données de l'utilisateur.

Tout au long du développement de cette application, nous nous sommes également attachés à utiliser des ressources disponibles en licence open source, nous permettant ainsi de partager notre code source également sous licence GPL.

Enfin, nous sommes convaincus que cette application de domotique offre de nombreuses possibilités d'extension et d'amélioration, que ce soit en termes de fonctionnalités, de performances ou de sécurité. L'implémentation réelle d'une connexion avec un pont permettant de communiquer avec des équipements réels seraient une perspective des plus intéressantes. Nous sommes fiers du travail accompli et sommes impatients de continuer à développer et à améliorer cette application à l'avenir.

Bibliographie

Assistant, H. (s. d.). *Documentation*. Home Assistant. https://www.home-assistant.io/docs/

Introduction. (s. d.). openHAB. https://www.openhab.org/docs/

Documentation. (s. d.). Bulma : Free, open source, and modern CSS framework based on Flexbox. https://bulma.io/documentation/

Bootstrap Icons. (s. d.). https://icons.getbootstrap.com/#usage

MeteoFrance API | Open-Meteo.com. (s. d.). https://open-meteo.com/en/docs/meteofrance-api

Cyrille, H. (s. d.). *Le pattern DAO*. Developpez.com. https://cyrille-herby.developpez.com/tutoriels/java/mapper-sa-base-donnees-avec-pattern-dao/

B. (2022, 19 septembre). *The DAO Pattern in Java*. Baeldung. https://www.baeldung.com/java-dao-pattern

Le modèle DAO en Java. (s. d.). https://www.codeflow.site/fr/article/java-dao-pattern