# S-INFO-015 Projet d'analyse et de conception et S-INFO-106 Projet de développement logiciel

# **"Gestion de Consommation d'énergie"**

## Une application client-serveur

Enseignant: Professeur Tom Mens
Assistants: Pierre Hauweele, Sebastien Bonte, Jeremy Dubrulle

Année Académique 2022-2023
Sciences Informatiques

Faculté des Sciences, Université de Mons

*Date de ce document: 24 septembre 2022*

CHANGE LOG
v.0.2.0: Ajout de système de notification (24 septembre)
v.0.1.0: Ajout de technologie - JavaScript (16 septembre)

**Résumé**

Le projet de génie logiciel consiste à réaliser, en groupe de quatre personnes, et en plusieurs phases, une **application de gestion d'énergie** en Java. Pour s'assurer d'une séparation des préoccupations, le projet doit suivre le modèle **client-serveur**, avec en **backend** un serveur contenant la logique de l'application et une base de données, et en **frontend** la couche de présentation composé par plusieurs clients legers qui sont des applications webs interagissant par un **API REST** avec le serveur.

Le but du projet consiste à planifier, modéliser (au moyen d'une maquette de l'interface utilisateur et de la conception en UML), implémenter (avec JavaScript, Java, etc.), tester (avec JUnit) et documenter (avec JavaDoc) cette application. La qualité des modèles UML, du code source, et des tests logiciels est considérée comme *au moins aussi importante* que la fonctionnalité, robustesse et convivialité de l'application développée.

UMONS Université de Mons

Faculté des Sciences

# Table des matières

# 1 Énoncé

## 1.1 Préambule

☞ Le projet est composé de deux parties :
- la phase d'**analyse et de conception** correspond à l'AA S-INFO-015 en Q1
- la phase de **développement logiciel** correspond à l'AA S-INFO-106 en Q2

Selon l'UE qui se trouve dans votre PAE, vous devez soit suivre les deux AA ou uniquement la deuxième. Dans ce dernier cas, veuillez ignorer la partie de ce document qui concerne la phase d'analyse et de conception.

Le but de l'AA S-INFO-015 consiste à **modéliser** un système logiciel non-trivial au moyen d'une maquette de l'interface utilisateur et de la conception en UML. Le but de l'AA S-INFO-106 consiste à **implémenter** ce système avec un ou plusieurs langages de programmation, **tester** avec un framework de tests unitaires, et **documenter** le code et les tests (avec des outils comme JavaDoc). La **qualité interne** du projet (à la fois au niveau des modèles, du code source, des tests unitaires et de la documentation) ainsi que la **testabilité** du produit logiciel sont considérées comme *au moins aussi importante* que la **qualité externe** (la fonctionnalité, robustesse, convivialité, ergonomie et fluidité) de l'application développée.

Le projet consiste à réaliser en groupe une **application logicielle de gestion de portefeuille d'énergie et d'approvisionnement énergétique pour des habitations**. Bien que le projet fasse partie d'un cours universitaire, son contenu correspond à des besoins réels, il ne s'agit donc pas juste d'un sujet artificiel.

Le projet doit suivre le modèle **client-serveur** avec une architecture logicielle à trois couches, afin d'assurer une **séparation de préoccupations** entre la **couche de présentation** (le **frontend**) qui sera développé comme une application web. Le **backend** se trouvera sur un **serveur** qui contiendra la **couche logique** ainsi que la **couche de données** (contenant la base des données). L'interaction entre le frontend et le backend doit se faire par le biais d'une **API REST**.

L'ampleur du projet est considérable, il est donc primordial de commencer à travailler sur le projet dès le début, de bien planifier les choses, et de travailler régulièrement. Les enseignants du cours figureront comme clients et utilisateurs externes de votre application. Si vous trouvez des incohérences, ambiguïtés ou incomplétudes dans l'énoncé, veuillez donc contacter les enseignants pour clarifier les choses. Cela permet d'éviter beaucoup de problèmes ou malentendus pendant la modélisation et l'implémentation du projet, qui pourraient impacter négativement votre note finale.

## 1.2 Introduction et contexte du projet

Chaque groupe sera composé de quatre personnes [1] et proposera des services de gestion de l'approvisionnement énergétique pour des habitations. Chaque groupe peut être vu comme une entreprise qui commercialise des applications web de gestion énergétique à des particuliers et entreprises. La première application web à réaliser doit permettre au propriétaire d'une maison, qui est client d'un ou plusieurs fournisseurs d'énergie, de gérer son portefeuille énergétique et le suivi de sa consommation énergétique. La deuxième application web doit permettre à un fournisseur d'énergie de gérer le suivi des contrats d'énergie avec ses clients. La logique métier et la base de données nécessaires pour le bon fonctionnement des deux applications web (le frontend) seront hébergées sur le serveur (le backend) qui se synchronisera avec le frontend à travers une API REST.

*Par exemple, supposons que Jeanne est client, pour sa maison, chez un fournisseur de gaz F1 et chez un fournisseur d'électricité F2. Elle utilisera son application web cliente pour visualiser et manipuler son portefeuille énergétique (qui regroupe toute information pertinente pour la consommation pour sa maison pour les deux contrats d'énergie). Pour leur part, les deux fournisseurs F1 et F2 utiliseront (de manière continue), une autre application web qui leur permet de gérer les contrats d'énergie de tous leurs clients (dont Jeanne). Les deux applications web (du client et du fournisseur respectivement) ne communiquent pas directement entre elles. Toute interaction passe par l'API vers le serveur qui héberge la logique métier et la base de données. Par exemple, si Jeanne souhaite créer un nouveau contrat énergétique chez F2 (par exemple, pour livrer de l'électricité pour son appartement à la mer), elle peut introduire la demande par son application cliente, qui enverra une requête par HTTPS au serveur. L'application fournisseur utilisée par F1 sera notifiée par le serveur (par une autre requête HTTPS) de la demande de Jeanne. Un employé du fournisseur pourra alors traiter la demande, après quoi l'application fournisseur enverra la décision au serveur par une autre requête HTTPS. En cas d'acceptation, le serveur contactera (par une requête) l'application cliente utilisée par Jeanne afin de mettre à jour son portefeuille énergétique en incluant le nouveau contrat de livraison d'énergie. En cas de refus, le serveur avertira l'application cliente de celui-ci.*

La terminologie nécessaire pour la compréhension de l'énoncé est décrite en section 1.3. Les exigences technologiques sont détaillées en section 1.4. Chaque groupe doit implémenter **ensemble** les fonctionnalités de base décrite en section 1.5. Chaque membre du groupe doit réaliser **individuellement** une des extensions (décrites en section 1.6) et les intégrer dans le projet de son groupe. Le choix des extensions se fera en accord commun entre les étudiants et les enseignants.

1. De plus petits groupes seront uniquement permis dans des cas exceptionnels.

## 1.3 Terminologie spécifique au domaine énergétique

Un **consommateur** (un individu ou une organisation) peut consommer différents types d'énergie (gaz naturel, électricité, eau, ...) qui lui parviennent via des **points de fourniture d'énergie**. Chaque point de fourniture d'énergie est associé à une localisation géographique précise, généralement une habitation, et est identifié par un **code EAN** (signifiant « European Article Numbering ») composé de 18 chiffres. La consommation de l'énergie de ce point de fourniture peut typiquement être mesurée à l'aide d'un **compteur** qui peut être **mécanique** ou **numérique**. Un compteur peut également être **simple** (mono-horaire) ou **double** (bi-horaire). Ce dernier type de compteur sépare la consommation en **heures creuses** de la consommation en **heures pleines**, pour bénéficier de tarifs différents en fonction du moment d'utilisation. Les heures creuses peuvent dépendre de la commune ou du fournisseur d'énergie, la configuration du système logiciel à réaliser devrait donc pouvoir configurer ces paramètres.

Chaque point de fourniture est approvisionné par un **fournisseur d'énergie**. Pour l'électricité et le gaz, le consommateur a le choix parmi plusieurs entreprises. La liste complète est actualisée des fournisseurs peut être consultée ici :

`https://www.comparateur-energie.be/fournisseurs-electricite-et-gaz`

Pour la distribution d'eau potable, il n'y a par contre qu'un seul fournisseur par territoire :

`https://www.aquawal.be/fr/connaitre-votre-distributeur-d-eau.html?IDC=319`

Un consommateur souhaitant s'approvisionner en énergie doit conclure un **contrat** avec un fournisseur. Typiquement, il y aura un contrat différent pour chaque type d'énergie, mais certains fournisseurs vous permettront de conclure un contrat groupé pour combiner l'électricité et le gaz. Le type et les conditions du contrat dépendent de plusieurs paramètres (par exemple, réductions tarifaires liées à la fidélité, ou encore un tarif social), et si le fournisseur le permet le consommateur peut choisir entre un contrat à **tarif fixe** (c.-à-d. que pendant une durée déterminée, l'énergie consommée est facturée à un certain montant défini et fixe) ou un contrat à **tarif flexible** (c.-à-d. que le prix de l'énergie dépendra du marché et variera tout au long de la période couverte par le contrat).

Afin d'obtenir un échantillon réaliste et représentatif des fournisseurs et des types de contrats d'énergie, nous permettons à l'ensemble d'étudiants (tous les groupes confondus) de regarder les offres et contrats types proposés par au moins 5 fournisseurs, tenant compte des restrictions géographiques et en gardant à l'esprit que chaque type d'énergie (gaz, électricité et eau) devra être représenté.

Différentes **unités** peuvent être utilisées pour exprimer la **quantité d'énergie**. Par exemple, la quantité d'électricité (consommée ou produite) est exprimée en **kilowatt-heure (kWh)**, tandis que la quantité de gaz ou d'eau est exprimée en **mètre cube ($m^3$)**. Concernant le gaz et particulièrement s'il s'agit d'un contrat combiné pour l'électricité et le gaz, il se peut que la facture mentionne la consommation de gaz constatée en $kWh$, sur base d'une conversion effectuée par le fournisseur. La formule de conversion de la quantité de gaz consommée en $m^3$ est : $1m^3 = 10,3kWh$.

Si vous désirez plus de renseignements sur le marché d'énergie en Belgique et en Wallonie en particulier, vous les trouverez sur les sites suivants. Nous vous conseillons de les consulter avant et pendant le développement du projet, car ils fournissent toute une série d'informations utiles :

- La CWAPE : commission wallonne pour l'énergie, l'organisme officiel de régulation des marchés wallons de l'électricité et du gaz : `https://www.cwape.be`
- Le CREG : commission (belge) de régulation de l'électricité et du gaz : `https://www.creg.be/fr`
- `https://www.comparateur-energie.be` et `https://www.comparateur-energie.be/blog`
- `http://www.synergrid.be`

## 1.4 Exigences technologiques

☞ Un des défis (et des apports) de ce projet consiste en un choix et utilisation judicieux des technologies, frameworks et librairies logiciels qui vous aideront dans votre travail. C'est votre choix, et c'est à vous de vous documenter sur le meilleur choix, et justifier ce choix dans le rapport d'implémentation. Evitez d'utiliser des frameworks trop complexes ayant une courbe d'apprentissage trop élevée. Evitez également de vouloir réinventer la roue en développant tout par vous même.

Pensez bien à la facilité d'installation et d'évaluation de votre projet par les enseignants. Les applications développées doivent être multiplateformes, donc il faut éviter les technologies qui ne fonctionnent que sur un système d'exploitation spécifique (par exemple Windows), ou qui nécessitent des outils ou licences particulières. L'application devrait être installable et exécutable sans aucun souci sur des machines tournant sous Windows, Linux et Mac OS X.

### 1.4.1 Architecture client-serveur à trois couches

Le système à réaliser doit suivre une **architecture client-serveur à trois couches**. Une telle architecture permet d'avoir une séparation plus ou moins stricte des préoccupations grâce à un couplage faible entre la **couche de présentation** présentant l'interface graphique, la **couche logique** présentant la logique "métier" de l'application, et la **couche de données** qui se trouve sur un serveur lié à une base de données. Cette séparation permet de changer la technologie de l'interface graphique ou de la base de données sans que cela affecte la logique de l'application. Voici quelques références expliquant l'architecture à trois couches :

- `https://fr.wikipedia.org/wiki/Architecture_trois_tiers`
- `https://www.ibm.com/cloud/learn/three-tier-architecture`
- `https://www.techopedia.com/definition/24649/three-tier-architecture`

**Frontend.** La couche de présentation, aussi appelée le **frontend**, doit être réalisé par des **applications web** légères, contenant uniquement les aspects de visualisation et interaction avec l'utilisateur. Ces applications web doivent tourner dans n'importe quel type de navigateur web récent, en utilisant des technologies JavaScript. Vous êtes libre dans les choix technologiques pour développer l'application web, soit en écrivant du code directement en JavaScript ou TypeScript (ou un autre langage de programmation compatible avec JavaScript), ou en utilisant un framework de développement web front-end comme par exemple Vue.js, React, Angular, etc...

Voici quelques tutoriels et guides sur comment écrire du code JavaScript :

- The official ECMAScript language specification `https://tc39.es/ecma262/` and the ECMAScript Internationalization API specification `https://tc39.es/ecma402/`
- The Modern JavaScript Tutorial `https://javascript.info`
- W3Schools JavaScript Tutorial `https://www.w3schools.com/js/`
- TutorialsPoint JavaScript Tutorial `https://www.tutorialspoint.com/javascript`
- JavaTPoint JavaScript Tutorial `https://www.javatpoint.com/javascript-tutorial`
- MDN Mozilla Javascript guides and tutorials for beginners, intermediate and advanced programmers `https://developer.mozilla.org/en-US/docs/Web/javascript`
- Google JavaScript Style Guide `https://google.github.io/styleguide/jsguide.html`
- WordPress JavaScript Coding Standards `https://developer.wordpress.org/coding-standards/wordpress-coding-standards/javascript/`

Et voici quelques références utiles sur les frameworks web pour JavaScript :

- `https://www.simform.com/blog/javascript-frameworks/`
- `https://dzone.com/articles/the-best-javascript-frameworks-for-2022`
- `https://hackr.io/blog/best-javascript-frameworks`
- `https://www.makeuseof.com/most-popular-javascript-frameworks/`

**Backend.** La vraie "logique" de l'application doit être réalisé comme une application traditionnelle en Java qui sera hébergé sur le serveur (le **backend**), et qui s'occupera de l'interaction avec la base de données.

Vous êtes libre dans les choix technologiques pour gérer les données, pour autant que cette gestion sera totalement transparente pour le frontend. Exemples de technologies à considérées sont : une base de données traditionnelle en SQL (par exemple, en PostgreSQL ou MySQL), une solution légère avec du SQLite, une solution de type non-relationnelle (NoSQL). Il est aussi possible de considérer l'utilisation des solutions de type ORM (object-relational mapping).

### 1.4.2 Architecture REST

Au niveau technologique, il est imposé d'utiliser une architecture "RESTful" pour l'interaction entre le frontend et le backend. REST correspond à un ensemble de contraintes architecturales, et il y a de nombreuses façons de réaliser une telle architecture.

Toute la logique métier du système logiciel ainsi que les données concernant les fournisseurs, les clients, les habitations, les contrats d'énergie et l'historique de leur consommation énergétique seront hébergées dans une base de données sur un serveur. Les applications web clientes ne stockeront pas de données en local. Les applications frontend seront donc principalement de simples "consommatrices" de données hébergées sur le serveur en backend. Cependant, rien n'empêche l'utilisation d'un cache local sur les applications si cela permet d'augmenter la performance des applications.

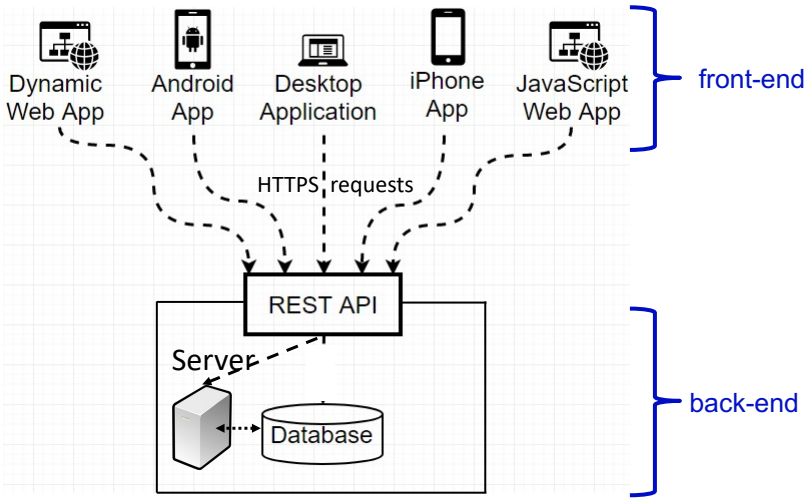


FIGURE 1 – Architecture client-serveur de type "RESTful".

Au niveau de l'architecture du système, nous distinguons les composants visualisés dans la figure 1 :

- Une application **serveur** en **Java** qui implémentera la logique métier du système et s'occupera de l'interaction avec la couche de données.
- Une **base de données** hébergée par le serveur, permettant de stocker ou récupérer des données.
- Un front-end de plusieurs applications **clientes**, qui seront des applications web en utilisant la technologie **JavaScript**. Les applications web ne sont pas autorisées à communiquer directement entre elles. Toute communication se fait par l'intermédiaire du serveur.
- Une **API** (Application Programming Interface) de type **REST** (REpresentational State Transfer) cachera les détails du serveur, et permettra à n'importe quelle application cliente de manipuler les données par l'API qui lui est fournie. L'interaction entre l'application cliente et le serveur se fera par le protocole **HTTPS** (secure hypertext transfer protocol), en utilisant des méthodes paramétrées de type GET, POST, PUT et DELETE.
  La communication entre les clients et le serveur doit être "sans état" (anglais : *stateless*), impliquant que le serveur ne stockera aucune information cliente entre plusieurs requêtes : chaque requête sera indépendante et totalement déconnectée des requêtes précédentes.

— Les données structurées qui seront transférées entre clients et serveur devront être encodées de préférence en format **JSON** ou **YAML** à cause de leur format qui est à la fois structuré et lisible par des humains [2]. Des exemples de fichiers YAML sont disponibles aux figures 2, 3, 4 et 5. Notez qu'il s'agit bien d'exemples purement illustratifs : nous ne donnons aucune garantie sur la pertinence des formats présentés ici ni de leur contenu ; c'est aux membres du groupe de réfléchir à la manière d'organiser et de structurer les données.
— Pour réaliser certaines extensions de la section 1.6, d'autres composants de type application client sont envisageables. Chacune de ces applications interagira de la même manière avec le serveur.

Puisque le système à réaliser nécessite plusieurs applications qui communiqueront toutes avec le même serveur via une API, il convient de décider quelle partie de la logique devrait se trouver du côté serveur (pour éviter que chaque application doive implémenter la même logique) ou plutôt du côté client (pour éviter que le serveur ait une charge de travail trop importante).

```
portefeuille:
  id: p1589156768969
  # l'habitation liée à ce portefeuille
  habitation: h0338384312557
  # les utilisateurs ayant accès à ce portefeuille
  users:
    - type: gestionnaire
      user-id: u0004590088929
    - type: lecture-seule
      user-id: u3088953646046
    - type: lecture-ecriture
      user-id: u2722500332198
  # les points de fourniture gérés par le portefeuille
  points-de-fourniture:
   - ean-18: 803429457858963427
   - ean-18: 946430795197304526
```

FIGURE 2 – Exemple d'un fichier YAML représentant un portefeuille énergétique

```
consommation:
  # le compteur est identifié de manière unique par son code EAN-18
  compteur: 803429457858963427
  releves:
    - date: 2017-01-05
      valeur: 7085
    - date: 2017-01-12
      valeur: 7123
    - date: 2017-01-19
      valeur: 7135
    - date: 2017-01-26
      valeur: 7142
```

FIGURE 3 – Exemple d'un fichier YAML représentant des données de consommation liées à un compteur.

Documentez-vous donc bien afin de faire un choix technologique le plus approprié. Voici quelques pointeurs potentiellement utiles :

— Representational state transfer
`https://en.wikipedia.org/wiki/Representational_state_transfer`
— REST API Tutorial
`https://restfulapi.net/rest-api-design-tutorial-with-example/`

2. Avec l'accord des enseignants, et sur demande, bien justifiée, d'autres formats de données comme le **XML** pourront également être acceptés

```yaml
fournisseur:
 nom: Lampiris
 siege:
  rue: Saint Laurent
  numero: 54
  ville: Liege
  code-postal: 4000
 contrats:
  # un premier contrat pour un client et un point de fourniture
  - client: E09156532
    numero-contrat: C12954278
    debut: 2018-07-01
    fin: 2020-06-30
    ean-18: 130539057492609625
  # un deuxième contract pour un autre client et deux points de fourniture
  # (le contrat est encore en cours, donc il n'y a pas de date de fin)
  - client: E08712638
    numero-contrat: C13871234
    ean-18: 8034294578589634 27
    ean-18: 946430795197304526
    debut: 2020-07-01
```

FIGURE 4 – Exemple d'un fichier YAML représentant les contrats conclus entre un fournisseur et ses clients consommateurs.

```yaml
habitation:
 habitation-id: h0338384312557
 adresse:
   rue: Avenue Maistriau
   numero: 15
   batiment: De Vinci
   local: 2.18
   ville: Mons
   code-postal: 7000
 points-de-fourniture:
  - point: electricite
    ean-18: 8034294578589634 27
  - point: gaz
    ean-18: 946430795197304526
  - point: eau
    ean-18: 794929267075728442
```

FIGURE 5 – Exemple d'un fichier YAML représentant les points de fourniture d'une habitation.

— How to create a (basic) REST API in Java
`https://happycoding.io/tutorials/java-server/rest-api`
— Top 10 Best Java REST and Microservice Frameworks
`https://rapidapi.com/blog/top-java-rest-frameworks/`
— Building REST services with Spring
`https://spring.io/guides/tutorials/rest/`
— Developing Restful APIs in Java Tutorial
`https://www.youtube.com/watch?v=5jQSat1cKMo`
— JSON Tutorial `https://www.tutorialspoint.com/json/`
— Framework MVC pour des sites web Java (avec support pour REST/JSON)
`https://struts.apache.org/`
— Hébergeur gratuit qui gère Java
`https://www.alwaysdata.com/fr/`

### 1.4.3 Testabilité de l'application

Lors de la réalisation des fonctionnalités demandées, il est **obligatoire** d'offrir un moyen automatique aux enseignants (et à vous-même !) pour tester et simuler toute fonctionnalité implémentée. Nous insistons sur l'utilisation d'un processus de **test-driven development** [3]. Tout au long du développement vous devez mettre en place des **tests unitaires**, avant et après chaque ajout de fonctionnalité ou chaque refactoring, ainsi que l'ajout des tests de régression chaque fois qu'un bug ou défaut est découvert. Les tests serviront comme spécification et documentation active et continue de votre application. Si un test échoue, alors cela implique qu'il y a un problème avec la fonctionnalité développée. Il est donc essentiel que des tests unitaires non triviaux et automatisés soient présents pour tester la fonctionnalité et le comportement de votre application.

Dans la programmation orientée objet, des **mock objects** sont des objets simulés qui imitent le comportement d'objets réels de manière contrôlée, le plus souvent dans le cadre de tests unitaires. La principale raison de créer de tels mock objects est de pouvoir tester une unité du système logiciel sans avoir à se soucier des modules dépendants. La fonctionnalité de ces dépendances est "simulée" par les mock objects. Ceci est particulièrement important si les fonctions simulées sont difficiles ou longues à obtenir (par exemple parce qu'elles impliquent des calculs complexes), si on désire tester l'interaction avec des composants auxquels on a pas accès lors du développement de l'application (par exemple un serveur distant), si le résultat est non déterministe, ou s'il est trop dangereux d'accéder à une base de données en production lors des tests. Les mock objects sont aussi utilisés souvent pour simuler l'interaction avec une interface graphique.

Un autre aspect de la testabilité de votre application est que votre solution technique proposée doit être totalement transparente pour l'utilisateur du point de vue interaction avec la base de données (qui pourrait se trouver en local ou en ligne). Les enseignants doivent être capables d'exécuter et de tester l'application sans avoir à interagir directement avec la base de données. Par exemple, les enseignants ne devront pas installer un client SQL pour exécuter des commandes ou des scripts fournis pour configurer ou peupler la base de données. Les livrables fournis lors de la phase d'implémentation (commande gradle et fichier jar auto-exécutable) doivent suffire pour exécuter et tester l'application.

☞ Pour faciliter les tests de certaines fonctionnalités, il se peut que vous deviez mettre en place des fonctionnalités auxiliaires (accessibles aux tests unitaires) qui ne sont pas directement disponibles dans l'interface utilisateur (ou peut-être uniquement par le biais d'une sorte de "mode debug"). Par exemple, si l'on veut tester qu'une certaine fonctionnalité se déclenche à une date précise (par exemple calculer la consommation énergétique mensuelle à la fin du mois sur base des données provenant d'un compteur numérique, ou calculer la facture énergétique à la fin du contrat), on pourra mettre en place soit un système permettant d'accélérer le temps (par exemple une seconde correspondra à un jour), ou un système permettant de modifier la date interne de l'application.

3. `http://www.agiledata.org/essays/tdd.html`

## 1.5 Fonctionnalités de base

☞ Les fonctionnalités décrites dans cette section doivent obligatoirement être implémentées par chaque groupe. La collaboration entre différents groupes est strictement proscrite et sera assimilée à un plagiat !

Deux applications web différentes doivent être réalisées, chacune visant un type d'utilisateur distinct. La section 1.5.3 décrit l'application de gestion de portefeuilles énergétiques pour les consommateurs ayant des contrats d'énergie chez un (ou plusieurs) fournisseur d'énergie. La section 1.5.4 décrit l'application permettant à un fournisseur d'énergie de gérer tous les contrats d'énergie permettant de livrer de l'énergie à ses clients.

Les deux types d'application web s'occupent de la visualisation et de l'interaction avec l'utilisateur. La logique métier se trouvera dans une application Java en backend sur le serveur, accessible par l'API REST. Cette application en backend gère les données stockées dans une base de données hébergée sur le serveur également. Dans cette base de données, tous les clients, contrats d'énergie et fournisseurs d'énergie sont représentés de manière unique, et l'historique de la consommation énergétique, la gestion des contrats d'énergie (y compris leur création et fermeture) et les portefeuilles énergétiques y sont capturés.

### 1.5.1 Internationalisation (i18n) et localisation (l10n)

Les applications clientes (en frontend) et l'application serveur (en backend) doivent supporter plusieurs langues, suivant les bonnes pratiques d'internationalisation (i18n) et de localisation (l10n). Au minimum deux langues doivent être supportées (l'anglais et le français) avec la possibilité de facilement ajouter d'autres langues.

Lors de la création d'un login, chaque utilisateur doit choisir sa langue préférée. Il peut toujours changer son choix. L'application utilisera alors cette langue après chaque connexion et utilisation de l'application par l'utilisateur.

Voici quelques références utiles concernant l'internationalisation :

- `https://en.wikipedia.org/wiki/Internationalization_and_localization`
- Java Internationalisation guide
  `https://docs.oracle.com/en/java/javase/17/intl/`
- ECMAScript Internationalization API Specification
  `https://402.ecma-international.org`
- `https://www.freecodecamp.org/news/how-to-get-started-with-internationalization-in-javascript-c09a0d2cd834/`
- `https://www.sitepoint.com/javascript-internationalization-api-i18n/`

### 1.5.2 Système générique de notification

Parce que l'architecture client-serveur interdit la communication directe entre les applications en frontend, l'interaction entre ces deux applications web se fera de manière indirecte par un système générique de notifications qui est géré par l'application serveur et sa base de données. Lorsqu'une action est requise de la part d'un utilisateur d'une des applications clients, une notification lui sera envoyée par le serveur par un mécanisme "pull-based" : l'application vérifiera à des intervalles réguliers si de nouvelles notifications sont disponibles pour l'utilisateur connecté à l'application. Cet utilisateur pourra également déclencher manuellement le téléchargement de nouvelles notifications. L'utilisateur aura la possibilité de consulter toutes les notifications le concernant, et d'effectuer les actions requises par celles-ci. Le système de notifications doit être suffisamment générique pour pouvoir gérer les différentes actions possibles, y inclus celles qui seront proposées par les extensions de la fonctionnalité de base. La figure 6 illustre le processus d'un tel système de notifications.

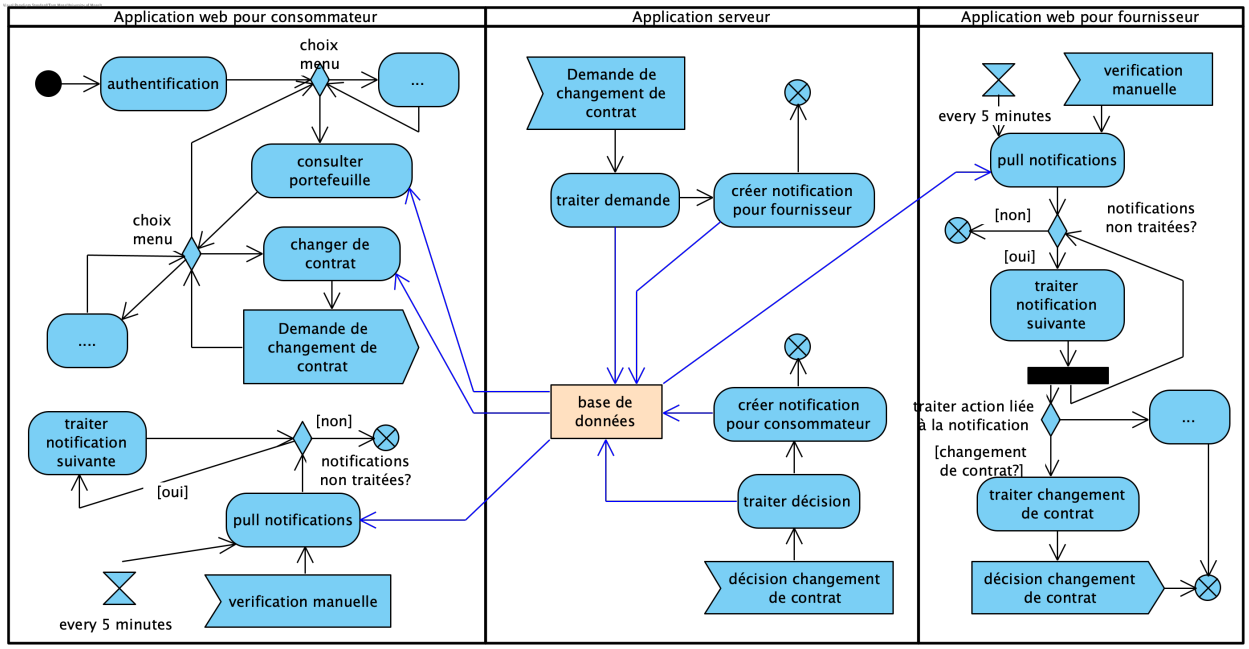


FIGURE 6 – Diagramme d'activités UML illustrant le processus du système de notifications.

*À titre d'exemple, supposons qu'un consommateur désire créer un nouveau contrat avec un fournisseur. Cette demande sera envoyé par une requête vers l'API du serveur. L'application serveur traitera la demande en mettant à jour la base de données, et créera une notification à traiter par l'application fournisseur. Le fournisseur qui utilise son application web recevra cette notification et agira en acceptant ou refusant le contrat, en enverra cette décision par une requête vers l'API du serveur. L'application serveur enregistrera la réponse dans la base de données, et génèrera une nouvelle notification à traiter cette fois-ci par l'application consommateur, afin d'informer celui-ci de la décision du fournisseur. Le consommateur qui utilise son application web recevra cette notification.*

### 1.5.3 Application “consommateur” de gestion de portefeuilles énergétiques

Cette application permet aux consommateurs d'énergie de créer et de gérer leurs portefeuilles énergétiques. On entend par « portefeuille » un ensemble de données de consommation d'énergie, pour une entité indépendante énergétiquement (typiquement une habitation), qui peut contenir un nombre quelconque de points de fourniture d'énergie correspondant à un ou plusieurs types d'énergie.

La figure 7 montre deux exemples de portefeuilles. Le premier contient de l'information correspondant à l'eau, l'électricité et le gaz pour la résidence principale d'une famille. Le deuxième contient de l'information pour l'eau et l'électricité pour la résidence secondaire de la famille.

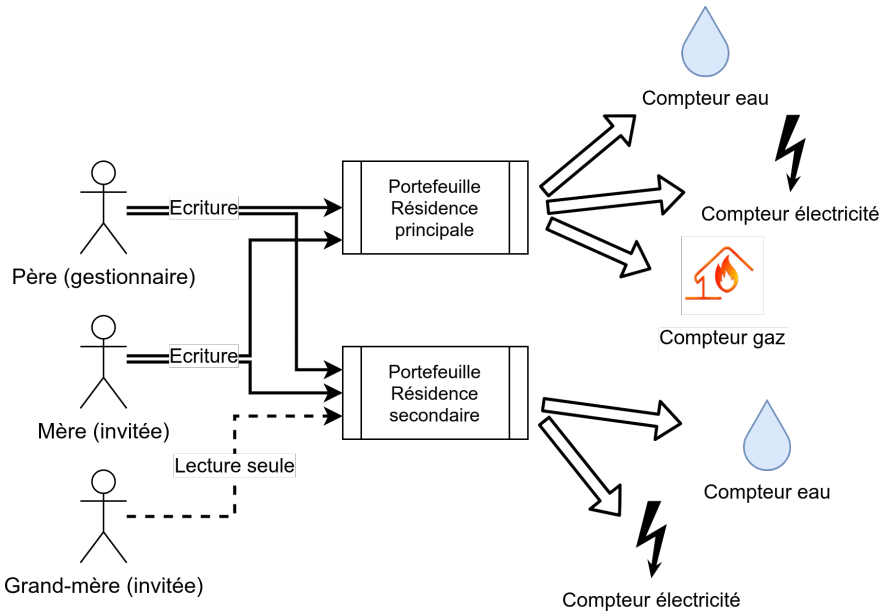


FIGURE 7 – Exemple de 2 portefeuilles d'énergie

Au premier lancement de l'application, l'utilisateur doit pouvoir choisir entre se connecter à un compte existant ou en créer un nouveau (une fonction de réinitialisation de mot de passe doit également être présente). Il doit ensuite pouvoir gérer les portefeuilles auxquels il est lié (aucun au moment de la création d'un nouveau compte). S'il crée un nouveau portefeuille, il en devient automatiquement le gestionnaire. Il doit dès lors pouvoir y ajouter un ou plusieurs points de fourniture d'énergie et y associer un fournisseur et un type de contrat. Il peut également supprimer un point de fourniture ou même le portefeuille complet, ainsi que modifier les paramètres de ceux-ci.

**Gestion de compteurs et saisie manuelle de consommation.** L'application doit permettre au consommateur de voir tous les compteurs qui lui sont (ou ont été) affectés par les fournisseurs, avec l'historique d'affectation dans le temps. Il peut ainsi voir quels compteurs sont actuellement ouverts pour lui (et depuis quand). Il peut également voir lesquels de ses compteurs ont été clôturés (et depuis quand) par le fournisseur.

Un consommateur pourra associer, par une saisie manuelle, des données de consommation à un compteur, en précisant la date et la quantité d'énergie indiquée sur le compteur (cette quantité est exprimée en $kWh$ pour électricité, et en $m^3$ pour gaz et eau). Après introduction des données, le consommateur peut les envoyer au serveur afin de les insérer dans la base de données et de notifier le fournisseur. Puisqu'il est toujours possible de commettre des erreurs lors d'une introduction manuelle, il doit être possible pour le fournisseur (dans l'application 1.5.4) de modifier des données qui ont été introduites manuellement par le consommateur.

**Visualisation et exportation de la consommation.** Pour un portefeuille donné, il doit être possible de visualiser et/ou d'exporter les données historiques de la consommation d'énergie pour chaque point de fourniture associé au portefeuille, à différentes granularités temporelles : consommation journalière, hebdomadaire, mensuelle ou annuelle. Les données de consommation ne sont disponibles au gestionnaire d'un portefeuille que pendant la période pendant laquelle il est détenteur d'un contrat pour le(s) point(s) de fourniture concerné(s) : s'il existait un précédent portefeuille reprenant les mêmes EANs, les données ne doivent pas être accessibles.

La visualisation de la consommation doit se faire sous forme de tableaux, ainsi que sous forme de graphes (par exemple : line charts, pie charts, bar charts, box plots, spider charts, violin plots, ...). Pour faciliter la visualisation, toute consommation de gaz sera exprimée en $kWh$.

L'exportation des fichiers doit pouvoir se faire en format JSON ainsi qu'au moins un des formats suivants : CSV (comma-separated values) ou XML.

☞ Du point de vue technique, plusieurs bibliothèques graphiques existent pour réaliser les visualisations. De la meme manière, plusieurs bibliothèques existent pour la gestion de formats de données à l'aide de parseurs dédiés. Le choix de la librairie la plus appropriée sera laissé à la discrétion de chaque groupe.

### 1.5.4 Application pour les fournisseurs d'énergie

Cette application fournira une interface commune aux fournisseurs d'énergie. Par simplicité, nous supposons qu'il y a un seul login unique par fournisseur dans l'application. Ceci évite la création d'un compte individuel pour chaque employé du fournisseur. Tout comme dans l'application consommateur de la section 1.5.3, le login est protégé par un mot de passe, qui peut être changé dans l'application.

**Gestion de clients et de contrats.** L'application doit permettre à un fournisseur d'ajouter de nouveaux clients consommateurs, de supprimer des clients existants, et de gérer les contrats d'énergie pour ses clients. Si un consommateur possède un contrat chez plusieurs fournisseurs, chaque fournisseur aura uniquement accès aux contrats avec ses propres clients pour des raisons de confidentialité. Un fournisseur n'aura pas accès non plus à la notion de "portefeuille énergétique", car ce concept fait partie uniquement de l'application de gestion de portefeuilles énergétiques décrite en section 1.5.3.

Un employé d'un fournisseur d'énergie qui est connecté à l'application aura l'autorité de créer ou supprimer des clients, d'ouvrir de nouveaux contrats pour un de ses clients et de clôturer ou modifier des contrats existants. Tout cela pourra se faire directement à l'initiative du fournisseur, ou indirectement en réponse à une demande venant d'un client (par l'application de la section 1.5.3), qui sera transférée par le serveur à l'application fournisseur.

**Gestion de compteurs.** Un fournisseur pourra associer des compteurs à ses consommateurs. Pendant la période où un compteur est associé à un consommateur, le compteur est considéré *ouvert* pour ce consommateur. Un même compteur ne peut pas être associé à plusieurs consommateurs simultanément. Pour réaffecter un compteur à un autre consommateur, le fournisseur doit d'abord clôturer le compteur (c.-à-d. le désaffecter du consommateur actuel), et ensuite ouvrir le compteur pour l'affecter au nouveau consommateur. La date de réouverture doit être postérieure ou égale à la date de clôture. Cette notion de période d'association d'un compteur à un consommateur est importante pour l'application de gestion de portefeuilles énergétiques, car un gestionnaire de portefeuille peut uniquement visualiser la période de consommation pendant laquelle le compteur lui appartenait. Associer un compteur à un consommateur implique donc que le fournisseur mette fin au contrat existant pour le(s) compteur(s) concernés (si ce n'est pas déjà le cas) et en établisse un nouveau.

**Gestion de consommation.** Un fournisseur pourra associer des données de consommation à un compteur, et/ou modifier des données erronées. L'introduction des données se fera par un import d'un fichier (en format csv, XML, JSON ou YAML) contenant les données de consommation associées à un ou plusieurs compteurs. Lors de l'importation, il faut veiller à ne pas introduire de doublons ou des incohérences. Par exemple, si la date liée à certaines données importées pour le même compteur correspond à des données déjà présentes dans le système, il faut avoir le choix entre l'annulation de l'import, le remplacement des anciennes données par les nouvelles, ou uniquement l'ajout des nouvelles données en ignorant celles qui ont des conflits.

Le fournisseur peut supprimer des données historiques déjà associées à un compteur, soit partiellement ou totalement. La suppression des données nécessite une confirmation (par exemple par une boite de dialogue) pour éviter la suppression des données "par erreur". Si le fournisseur supprime des données d'un compteur qui est encore actif pour un consommateur, l'application doit en informer ce consommateur (au travers du serveur) au moment où ce dernier se connecte à l'application consommateur.

## 1.6 Fonctionnalité des extensions

Chaque groupe d'étudiants doit choisir un ensemble d'extensions différentes, en accord commun avec les enseignants. Chaque extension doit être exécutable indépendamment des applications définies pour la fonctionnalité de base. Un exécutable indépendant sera donc produit pour chaque extension demandée.

### 1.6.1 Gestion d'utilisateurs

Un gestionnaire d'un portefeuille énergétique dans l'application "consommateur" peut inviter d'autres utilisateurs (qui ont déjà un compte sur l'application) sur son portefeuille. Il peut alors définir (et changer par la suite) leur niveau d'accès :

1. **Lecture seule :** permet uniquement de visualiser les données d'un portefeuille (comme les contrats, compteurs et fournisseurs associés, ainsi que l'historique de consommation pour chaque compteur). Aucun changement ne peut cependant être réalisé par un utilisateur qui ne dispose que de ce niveau d'accès.
2. **Lecture/écriture :** permet, en plus du privilège de lecture, d'ajouter des données de consommation (relevés des compteurs) à un portefeuille d'énergie (via une application telle que celle décrite en section 1.5.4).
3. **Gestion :** Ce type d'accès est uniquement disponible au gestionnaire du portefeuille. Un gestionnaire a le droit de changer les niveaux d'accès d'un utilisateur du portefeuille, ou d'ajouter et de supprimer des utilisateurs du portefeuille. Le gestionnaire bénéficie évidemment aussi du niveau d'accès « lecture/écriture » sur son portefeuille.

Reprenant l'exemple de la figure 7, le père de famille sera gestionnaire de ces deux résidences, et la mère aura accès lecture/écriture. La grand-mère, qui utilise la résidence secondaire régulièrement aura accès en lecture seule afin de mieux surveiller sa consommation énergétique.

Après avoir été invité à un portefeuille par son gestionnaire, lorsqu'un utilisateur se connecte à l'application de gestion des portefeuilles, il devra confirmer ou refuser l'invitation. S'il confirme, il verra apparaitre le portefeuille avec les accès (lecture seule ou lecture/écriture) qui lui ont été attribués.

L'interface visuelle devra être adaptée en fonction du statut (gestionnaire ou invité) et du niveau d'accès de l'utilisateur du portefeuille sélectionné, en gardant à l'esprit qu'un même utilisateur peut être gestionnaire sur certains portefeuilles et invité sur d'autres avec différents niveaux d'accès.

### 1.6.2 Gestion de portefeuilles agrégés

[Cette extension ne peut pas être choisie en combinaison avec l'extension de la section 1.6.3.]

L'application consommateur de la section 1.5.3 permet de gérer des "*portefeuilles simples*" qui sont constitués d'un ensemble de points de fourniture d'énergie (et les données de consommation historiques associées), pour une seule entité indépendante énergétiquement (typiquement, une habitation comme une maison ou un appartement).

Cette extension vise à étendre cette notion de portefeuille à des "*portefeuilles agrégés*", constitués d'un ou plusieurs sous-portefeuilles, possiblement de manière hiérarchique. Ainsi, un portefeuille agrégé peut aussi inclure les données énergétiques de plusieurs habitations. Cette extension permet de créer des portefeuilles pour couvrir, entre autres, les situations suivantes :

- Reprenant la situation visualisée en figure 7, le père d'une famille ayant plusieurs résidences pourra créer et gérer un portefeuille agrégé regroupant les deux portefeuilles simples (une pour chaque résidence), afin d'avoir une vue globale sur la consommation énergétique de la famille.
- Un propriétaire d'un bâtiment composé de plusieurs kots étudiants souhaite surveiller la consommation du bâtiment entier, ainsi que de chaque kot individuel. Pour ce faire, il créera un sous-portefeuille pour chaque kot, avec accès lecture pour l'étudiant occupant le kot (pendant la période d'occupation uniquement, correspondant à son contrat de location). Il créera également un sous-portefeulle pour les points de fourniture d'énergie associés aux salles en commun du bâtiment (comme la cuisine, les couloirs, et toutes les salles de bain et toilettes partagées).

— Le syndic d'un bâtiment (typiquement, une agence immobilière) souhaite gérer un immeuble d'appartements. Le locataire ou propriétaire de chaque appartement aura son propre portefeuille simple, et le syndic gérera le portefeuille agrégé qui est composé de l'ensemble des portefeuilles simples. La différence principale avec le cas précédent (kots étudiants) est que le syndic sera gestionnaire du portefeuille agrégé, mais aura uniquement accès en lecture seule (ou lecture/écriture si le propriétaire de l'appartement le permet) aux sous-portefeuilles.

Cette extension vise également à gérer le concept de sous-compteurs. En effet, il est possible qu'un bâtiment divisé en plusieurs habitations ne possède qu'un point de fourniture par type d'énergie. Cela se produit lorsque le propriétaire de l'habitation, dont il est initialement l'unique habitant, décide de subdiviser physiquement son habitation en plusieurs habitations distinctes. Étant donné qu'il ne possède à ce stade qu'un point de fourniture (et donc un seul contrat) par type d'énergie, il n'est pas capable de distinguer la consommation de chaque sous-habitation. Dans ce cas, le propriétaire peut installer à ses frais des sous-compteurs (n'ayant pas d'identifiant EAN), reliés au compteur principal, qui mesurent la consommation d'une zone précise du bâtiment et permet de justifier le montant qu'il réclame à chaque locataire du bâtiment. De ce fait, un relevé de consommation dans une telle habitation reste associé au compteur principal qui possède un code EAN, mais doit inclure un identifiant de sous-compteur unique (local à l'immeuble), qui permet par la suite d'isoler la consommation totale associée à chaque locataire.

### 1.6.3 Gestion énergétique pour organisations

[Cette extension ne peut pas être choisie en combinaison avec l'extension de la section 1.6.2.]

Le but de cette extension est de "commercialiser" le système logiciel pour des **organisations** (par exemple une entreprise, une école, une université, une ASBL). L'idée étant que la fonctionnalité de base, décrite en section 1.5, correspond à une version gratuite (*community edition*) avec fonctionnalité réduite.

Afin de bénéficier des fonctionnalités supplémentaires, il est possible d'activer une version étendue (*enterprise edition*) **payante**, directement au sein de l'application (*in app purchase*), en fournissant ses données de paiement. Il est évident que vous pouvez "simuler" cet aspect paiement dans le cadre de votre projet. Les fonctionnalités supplémentaires fournies par la version étendue deviennent uniquement disponibles à partir de la réception du paiement, et ceci pour une durée d'un an.

L'abonnement annuel peut être prolongé à n'importe quel moment, pour étendre la durée d'une année supplémentaire. À chaque lancement de l'application, une vérification s'effectuera par une connexion sur le serveur afin de déterminer si l'utilisateur peut toujours utiliser les fonctionnalités étendues.

La fonctionnalité étendue permettra à une organisation de gérer sa consommation énergétique, tenant compte de la structure de l'organisation. Une organisation est typiquement composée de manière hiérarchique de plusieurs niveaux contenant des divisions, départements, unités, services, ou autres. Le nom variera selon le type et la taille de l'organisation, et du niveau hiérarchique où on se trouve. Chaque "division" est associée à un ensemble de lieux de travail (par exemple des bâtiments ou des locaux dans ces bâtiments) contenant des points de fourniture d'énergie. Différentes "divisions" au même niveau hiérarchique ne peuvent pas partager les mêmes lieux de travail et points de fourniture d'énergie.

Lors de l'authentification à l'application, l'utilisateur doit préciser s'il se connecte en tant que particulier ou en tant que membre d'une organisation. Dans ce dernier cas, il faut préciser (ou sélectionner dans une liste) le nom ou identifiant de l'organisation et le nom ou identifiant de la "division" pour laquelle il est gestionnaire. Au moins une personne par division spécifique doit avoir accès à l'application pour gérer la consommation d'énergie de cette division.

L'extension "organisation" doit fournir les mêmes fonctionnalités que l'application de base, mais pour chaque division dans la hiérarchie organisationnelle. Par exemple, le directeur de l'entreprise (ou un suppléant) doit pouvoir surveiller la consommation, production et facturation énergétique globale de l'entreprise, et analyser comment cette consommation est repartie sur les différentes divisions au premier niveau de la hiérarchie. Le responsable d'une division doit pouvoir faire la même chose pour sa division et les sous-divisions qui la composent.

Il doit aussi être possible de mesurer et comparer la consommation énergétique des bâtiments occupés par

l'organisation ou la division. Ceci peut être très utile pour déterminer si, et où, il faut prendre des mesures pour réduire la consommation énergétique (par exemple, rénover et isoler un bâtiment, éteindre les lumières pendant la nuit, changer des lampes à incandescence pour des ampoules LED).

### 1.6.4 Analyse statistique de la consommation énergétique

L'application de gestion de portefeuilles (section 1.5.3) permet juste de *visualiser* les données historiques de la consommation énergétique associée au portefeuille. Cette extension ajoute à cela une interface visuelle permettant d'*analyser des statistiques* de la consommation. Cette fonctionnalité doit inclure :

- Le calcul des statistiques de base pour chaque séquence de données. Par exemple : la moyenne et l'écart type ; la médiane, les quartiles, l'écart interquartile et le minimum et maximum.
- Des tendances temporelles de la consommation. Ceci inclut : (a) la saisonnalité (la consommation sera différente pendant l'été que pendant l'hiver par exemple) ; (b) des tendances de croissance (par exemple, en comparant la consommation de l'année précédente avec celle de l'année en cours)
- Des comparaisons de la consommation entre des habitations ayant des caractéristiques comparables (même nombre d'habitants, même type d'habitation, même taille). Pour réaliser cette fonctionnalité, il n'est pas nécessaire d'aller chercher ce type d'information quelque part, vous pouvez aussi créer une simulation qui se base sur de fausses données, tout en restant assez réaliste.
- La détection de problèmes, grâce à l'analyse et la détection des changements soudains dans la consommation. Par exemple, si l'application détecte un pic de consommation excédant de la consommation habituelle en eau ou gaz sur une courte période, il y a probablement une fuite d'eau ou de gaz, ce qui pose un risque majeur (explosion dans le cas du gaz, inondation dans le cas d'eau). L'application devra donc immédiatement notifier les utilisateurs de cela de manière automatique, que ce soit par email, SMS ou réseau social (p. ex. Slack, Discord, message direct sur Twitter ou Whatsapp, ...). Les moyens de notification sont laissés au choix des étudiants.

Afin de pouvoir effectuer les analyses en temps réel, l'application devra fonctionner en permanence.

### 1.6.5 Auto-production d'électricité

Alors que l'application de base permet uniquement de surveiller la *consommation* d'énergie, cette extension permet également de gérer *l'auto-production* d'énergie. Chaque *consommateur* d'énergie peut donc devenir un *producteur* potentiel.

Pour la *production d'électricité*, le consommateur peut installer des panneaux photovoltaïques ou des éoliennes. Ils seront associés à un nouveau point de fourniture d'énergie avec un compteur numérique (avec code EAN) associé, permettant de mesurer la quantité d'électricité qui est produite et injectée au réseau de distribution d'électricité.

Le but de cette extension est de surveiller la production d'électricité, et de l'inclure en toute transparence aux autres fonctionnalités fournies par l'application. En particulier, il faut faire attention aux fonctionnalités liées à la visualisation et l'analyse de la consommation. Il doit toujours être possible d'avoir une vue séparée sur l'énergie *consommée* (retirée du réseau d'électricité) ou *produite* (injectée au réseau d'électricité) uniquement, ainsi qu'une vue groupée sur l'ensemble de l'énergie produite et consommée.

L'application devra notifier l'utilisateur par email, SMS ou réseau social (p. ex. Slack, Discord, message direct sur Twitter ou Whatsapp, ...) si la quantité d'énergie injectée au réseau de distribution dépasse un certain seuil (par exemple 1000 kWh), afin que l'utilisateur puisse demander un "certificat vert" pour sa production et permettre à un utilisateur ayant accès en écriture d'entériner l'acquisition du certificat. L'application devra garder la trace historique de tous les certificats verts demandés et acquis.

Cette extension doit également permettre d'analyser et de visualiser le rendement de la production d'électricité. Le rendement énergétique d'une installation produisant de l'énergie est défini par le nombre de kWh d'énergie produit par unité de temps. En fonction de la durée de vie de l'installation, le rendement diminue au fil du temps (par exemple, panneaux photovoltaïques qui se dégradent). Le rendement fluctue aussi en accord avec des facteurs météorologiques (sans vent, les éoliennes ne produisent pas ; sans soleil les panneaux non plus).

L'extension doit également permettre de comparer le rendement *actuel* des panneaux solaires avec leur rendement *théorique*. Le rendement d'un panneau solaire désigne la production réelle d'électricité par rapport à l'ensoleillement. Les données d'ensoleillement en Belgique sont disponibles sur la plateforme Moodle, dans le fichier "heuresDeSoleilBelgique.xls". Par exemple, un panneau solaire avec un rendement de 20% peut convertir 20% d'énergie solaire en électricité. Le taux de rendement d'un panneau solaire varie, selon la technologie utilisée dans le panneau et sa qualité, entre 15% et 22%. Cette variation est due à différents facteurs, comme la superficie du panneau et la puissance crête de celui-ci. Pour plus d'informations, voir : `https://www.panneausolaire-info.be/rendement`.

### 1.6.6 Budget énergétique et rapportage

Cette extension est similaire à la fonctionnalité de surveillance de la *consommation énergétique* en kWh (voir section 1.5.3), sauf qu'il s'agit ici d'une surveillance du *budget énergétique* en Euros. Le tarif que l'utilisateur doit payer pour une source d'énergie dépend de plusieurs facteurs : le type de contrat conclu avec le fournisseur d'énergie (par exemple contrat fixe ou contrat variable), le prix réel de l'énergie, l'utilisation ou non d'un compteur double (permettant de bénéficier d'un tarif réduit pendant les heures creuses), les coûts facturés par votre gestionnaire de réseau de distribution, et les redevances et taxes imposées par la commune ou le gouvernement, ... [4] Informez-vous bien sur l'ensemble de ces coûts, afin que le calcul soit le plus réaliste possible.

Ces facteurs doivent être fournis à l'application afin de pouvoir calculer le budget énergétique associé à un portefeuille. Afin de limiter son budget mensuel de consommation d'énergie, l'utilisateur doit avoir la possibilité de fournir un seuil. Si le budget énergétique mensuel dépasse ce seuil, il doit être notifié automatiquement par l'application.

L'extension doit permettre de visualiser et d'analyser le budget et la facturation énergétique d'un ou plusieurs portefeuilles de manière mensuelle et annuelle. Il doit être possible de générer des rapports globaux et détaillés du budget énergétique (par source d'énergie ; par portefeuille) pour une certaine durée de temps précisée par une date de départ et une date de fin. Ce rapport doit être visualisable par l'application, et doit être exportable en format pdf.

Finalement, l'extension doit permettre de comparer son budget énergétique avec d'autres habitations ayant des caractéristiques similaires : c'est similaire à la comparaison de la consommation d'énergie (voir section 1.5.3), sauf que de nombreux autres facteurs peuvent joueur un rôle important dans le calcul du budget énergétique : le contrat d'énergie, le fournisseur d'énergie, le gestionnaire de réseaux de distribution, les taxes, ...

### 1.6.7 Facturation et paiement d'acomptes

Le client d'un fournisseur d'énergie doit payer sa facture de consommation annuelle une fois par an, à la fin de son contrat annuel. Pour éviter de payer la somme totale en une seule fois, le fournisseur demande un paiement mensuel d'un acompte (une avance), qui correspond plus ou moins à un douzième du montant annuel, en estimant la consommation actuelle sur base de la consommation historique (les années précédentes) et du prix énergétique actuel. Si le consommateur dispose d'un compteur numérique, ou si des données de consommation récentes sont disponibles, le montant de l'acompte peut être calculé encore plus précisément.

L'application doit permettre à un consommateur de visualiser l'historique de ses factures annuels et acomptes mensuels, ainsi que leur statut de paiement. L'application doit permettre à l'utilisateur de modifier les acomptes mensuels à payer. L'application montrera l'acompte actuel, et proposera un montant "idéal" au client. Le client peut accepter cette proposition, ou il peut proposer par lui-même un autre montant plus élevé ou plus réduite. Une réduction (respectivement augmentation) de l'acompte ne peut être inférieure (respectivement supérieure) que de 20% du montant "idéal".

L'application doit permettre à un consommateur de changer son mode de paiement. Par défaut, le mode de paiement sera par prélèvement automatique (domiciliation). Le consommateur peut changer les données bancaires pour ce prélèvement, ou peut changer cela on en paiement manuel. Dans le cas où il choisi un

4. `https://www.comparateur-energie.be/blog/comprendre-facture-electricite-gaz/`

paiement manuel, le consommateur recevra, par l'application client, les demandes de paiement des acomptes et factures futurs. L'application serveur notifiera les consommateurs de manière automatique chaque fois qu'une nouvelle demande de paiement devient disponible. Cette notification pourra se faire soit par email, SMS ou réseau social (par exemple Slack, Discord, message direct sur Twitter ou Whatsapp, ...). Les moyens de notification sont laissés au choix des étudiants.

Le consommateur pourra régler ses paiements manuels en effectuant un virement ou en scannant un code QR présent sur la facture. Ce paiement manuel pourra se faire en liant l'application avec une autre application bancaire (bancontact, payconiq, application de votre banque, visa, ...) qui s'occupera du paiement. Il est évident que vous pouvez "simuler" cet aspect paiement dans le cadre de votre projet.

### 1.6.8 Génération de données de consommation et prédiction de consommation future

Le but de cette extension est de permettre à l'utilisateur d'effectuer des prévisions futures de sa consommation d'énergie, en effectuant des simulations tenant compte de toute une série de paramètres. Par exemple, des changements dans la composition de famille, l'installation des panneaux solaires, des rénovations et isolations de la maison, la récupération d'eau de pluie, ...

Cette extension doit également permettre d'effectuer des simulations et comparaisons de la consommation pour déterminer si c'est intéressant de changer le type de contrat chez son fournisseur d'énergie, ou si c'est intéressant de changer de fournisseur. L'extension devra pouvoir effectuer ces calculs pour divers fournisseurs d'énergie et proposer à l'utilisateur le(s) meilleur(s) choix de contrat(s) et de fournisseur(s), tenant compte de ses désidérata.s Cette fonctionnalité devrait être la plus réaliste possible, en se basant sur des données réelles, et similaires à ce qu'on peut trouver sur des comparateurs de contrats d'énergie que l'on peut trouver sur Internet, par exemple, le CREG scan. [5]

Finalement, cette extension doit inclure un **générateur de données**. Pour un "portefeuille d'énergie" donné, de fausses données (réalistes) seront ainsi générées, permettant de tester l'application facilement sans devoir introduire toutes les données manuellement. Les deux scénarios d'utilisation suivants sont attendus :

- Pendant que l'application est active, des relevés de consommation seront générés de manière périodique (en "temps réel") pour chaque compteur du portefeuille d'énergie.
- Pour une période renseignée (p. ex. depuis le début du contrat du point de fourniture jusqu'à maintenant), des relevés seront générés *a posteriori* pour chaque jour inclus dans cette période.

Étant donné que la consommation varie d'une habitation à une autre et d'un ménage à un autre, cette génération devra prendre en compte différents facteurs (qui seront des paramètres réglables dans l'application) tels que la Performance Énergétique du Bâtiment (PEB) et le nombre d'habitants occupant le logement. Le PEB se base notamment sur cinq indicateurs spécifiques : les besoins en chaleur du logement (qui prennent en compte l'isolation, l'étanchéité à l'air ...), la performance d'une ou plusieurs installations de chauffage, la performance d'une ou plusieurs installations d'eau chaude sanitaire, la présence d'un système de ventilation, la présence de systèmes utilisant des énergies renouvelables et/ou produisant de l'électricité. Des informations précises sur l'interprétation de ces indicateurs sont disponibles sur la brochure du Service Énergie de la Région Wallonne [6]. Un effort devra être fait sur la prise en compte du moment de la journée sur les relevés de consommation dans le cas d'utilisation d'énergies renouvelables, même si cela est difficilement quantifiable.

### 1.6.9 Jeu sérieux : réduction de consommation

Le but de cette extension est de fournir un **jeu sérieux** [7] permettant de conscientiser et d'inciter les utilisateurs de l'application à réduire leur consommation énergétique et empreinte carbone. L'objectif à court ou moyen terme pour l'utilisateur sera de réduire sa facture énergétique mensuelle, en suivant des conseils énergétiques qui lui seront proposés par l'application. Voici à titre d'exemple quelques conseils permettant de réduire sa consommation d'énergie : [8]

5. `https://www.creg.be/fr/cregscan`
6. `https://energie.wallonie.be/servlet/Repository/32423.pdf?ID=32423`
7. fr.wikipedia.org/wiki/Jeu_sérieux
8. Voir www.bienici.com/article/6-bonnes-idees-pour-reduire-sa-consommation-energetique et fr.wikihow.com/réduire-sa-consommation-énergétique.

— Éteignez l'éclairage quand vous n'en avez pas besoin.
— Remplacez vos ampoules à incandescence par des ampoules LED.
— Peignez vos murs et plafonds en couleurs clairs, reflétant plus de lumière et ainsi nécessitant moins d'éclairage électrique.
— Réduisez votre consommation d'eau. Par exemple, prenez une douche plutôt qu'un bain. Récupérez l'eau de pluie pour vos WC et pour arroser les plantes.
— Ne laissez pas vos appareils en mode veille.
— Utilisez des multiprises à interrupteur.
— Chauffez uniquement durant les heures d'occupation du logement.
— Réduisez la température intérieure de quelques degrés.
— Adaptez votre chauffage central à la température extérieure.
— Remplacez vos appareils énergivores (étiquetage énergétique C à G) par des appareils économes (étiquetage A+++ à B).
— Remplacez vos radiateurs électriques par un chauffage central.
— En cas d'un compteur double, utilisez les appareils électroménagers pendant les heures creuses.
— Fermez les volets plutôt que d'utiliser la climatisation s'il fait trop chaud.
— Installez des appareils écologiques (chaudière basse température ; chaudière à condensation), panneaux photovoltaïques ; pompes à chaleur ; ...
— Isolez votre logement (double ou triple vitrage ; isolation du toit, isolation thermique des murs)

L'aspect "jeu" devra se manifester sous plusieurs formes. Par exemple, en attribuant des *badges* quand l'utilisateur réalise des défis, ou en attribuant un score (numérique ou catégorique) à chaque utilisateur. La façon de concevoir cet aspect "jeu" est laissée à votre discrétion (modulo l'accord des enseignants).

Cette extension devrait permettre la réalisation de trois types de défis. La réalisation des objectifs pour chaque type de défi doit être mesurable par l'application (par exemple grâce à la surveillance de la consommation en section 1.5.3) :

— Les **défis personnels**. Par exemple, l'utilisateur de l'application pourrait avoir comme objectifs d'atteindre une réduction de consommation énergétique de X% en 12 mois ; de suivre au moins N conseils énergétiques ; de réduire sa facture annuelle par X% ; de réduire sa consommation d'énergie non verte ; d'augmenter sa production d'énergie par X% ; ...
— Les **défis coopératifs**. C'est comme les défis personnels, avec la différence que *plusieurs utilisateurs* ayant installé l'application doivent réaliser *ensemble* les objectifs proposés. Ces défis coopératifs impliquent l'implémentation d'un mécanisme permettant l'accès en lecture aux données des portefeuilles de tous les participants au défi coopératif.
— La **compétition** : Ici, des utilisateurs se mettent en compétition pour réaliser des défis. Pour permettre la compétition, il faut mettre en place un mécanisme permettant d'échanger des informations entre portefeuilles.

Typiquement, lors de la réalisation d'un défi, l'utilisateur devra introduire des informations dans l'application de manière manuelle. Par exemple, quelle est la mesure spécifique qu'il a mise en place pour réduire sa consommation, quand et pour quel prix ? En cas de changements dans sa situation pouvant avoir un effet sur sa consommation énergétique (par exemple, changement de composition de famille), il devra le signaler également.

Pour les variantes coopératives et compétitives du jeu, des données de l'utilisateur doivent être partagées avec ses coopérateurs ou ses compétiteurs. Ce partage doit se faire de manière anonymisée, tout en respectant les règles GDPR.

### 1.6.10 Application mobile sur Android

[Cette extension ne peut pas être choisie en combinaison avec l'extension de la section 1.6.11.]

Cette extension consiste à développer une application ayant les mêmes fonctionnalités que l'application web décrite dans la section 1.5.3, mais en tant qu'**application mobile sous Android**, en utilisant un langage de programmation supportés par Android Studio (par exemple Java, Kotlin ou Dart).

Les spécificités et conventions graphiques des plateformes mobiles doivent être prises en compte. Par exemple, puisque la place disponible pour l'affichage est réduite, l'interface doit être adaptée (les différences avec l'application desktop doivent être significatives). Les tailles d'écrans et les résolutions de différents appareils devraient également être prises en compte pour adapter au mieux l'expérience utilisateur. Puisqu'un smartphone bénéficie d'un écran tactile (et pour pallier au problème de place), l'application devra faire usage de "gestures" (par exemple, un "swipe" pour naviguer dans un menu) et, si approprié, des capteurs internes au téléphone comme un gyroscope ou un accéléromètre.

La phase d'analyse et de conception doit préciser les mécanismes permettant de gérer ces différences d'interactions entre l'application mobile et l'application web. L'application mobiledevrait fonctionner alternativement en mode portrait ou en mode paysage, en fonction de l'orientation du téléphone (basculement automatique). En fonction de l'orientation, certains éléments de l'interface devront peut-être être déplacés à divers endroits, pour préserver un affichage de qualité. L'étudiant devra pouvoir démontrer une version fonctionnelle de l'application sur une machine physique de type tablette ou smartphone et via un simulateur/émulateur (installé sur la machine de l'étudiant) pour l'autre type d'appareil (smartphone ou tablette).

☞ Il est à noter que cette extension implique l'utilisation de technologies non couvertes dans votre programme universitaire, et pour lesquelles les enseignants ne pourront pas fournir de support. Cette "extension" entraîne donc un "risque technique" plus élevé.

### 1.6.11 Application mobile sur iOS

[Cette extension ne peut pas être choisie en combinaison avec l'extension de la section 1.6.10.]

Cette extension est la même que l'extension de la section 1.6.10, sauf qu'on vise ici une **application mobile sous iOS**, permettant d'exécuter l'application sur un iPhone ou iPad.

# 2 Exigences

Chaque groupe d'étudiants doit *conjointement* réaliser les **fonctionnalités de base** (section 1.5) du système logiciel à réaliser. Chaque membre du groupe doit *individuellement* réaliser une **extension** choisie parmi les extensions proposées dans la section 1.6. Les enseignants veilleront à une répartition équilibrée des extensions entre les étudiants et les groupes.

Les différentes phases du projet doivent toutes inclure la fonctionnalité de base et les extensions choisies. Le système logiciel contenant uniquement les fonctionnalités de base devrait être rendu par le groupe comme un exécutable à part entière. Chaque membre du groupe doit réaliser un autre exécutable correspondant à son extension choisie : cet exécutable doit alors inclure toutes les fonctionnalités de base, étendues avec les fonctionnalités qui font partie de l'extension.

☞ Lors de l'évaluation de chaque phase du projet, en supposant un groupe composé de 4 étudiants, la fonctionnalité de base comptera pour 60% de la note finale, et les extensions individuelles réalisées par chaque étudiant compteront chacune pour 40% de la note finale. (Pour un groupe de 3 étudiants, la pondération sera 70% / 30% et pour un groupe de 2 étudiants 80% / 20%.)

## 2.1 Étapes clés et livrables

Le travail sera décomposé en deux phases. Au terme de chaque phase, des livrables précis seront rendus. Chaque livrable sera déposé sur la plateforme Moodle [9] selon les critères de recevabilité imposés dans la section 4.

**Phase 1 (en Q1) : Projet d'analyse et de conception (S-INFO-015)**

Cette phase concerne la maquette et la modélisation du projet logiciel.

**Diagrammes de conception UML.** En se basant sur le cahier des charges, un modèle de conception sera réalisé en UML. Les diagrammes suivants doivent **obligatoirement** être utilisés :
- les diagrammes de cas d'utilisation
- les interaction overview diagrams précisant les interactions entre les cas d'utilisation
- les diagrammes de classes (en précisant la structuration en paquetage)
- les diagrammes de séquences décrivant des scénarios de comportement non-trivial.

**Modèle de données.** Un modèle de données relationnelles doit être rendu, en utilisant les diagrammes d'entité-relation (ERD). Ce schéma doit préciser les différents types de données à manipuler par le système logiciel, ainsi que les attributs, les relations, et les clés primaires et étrangères. Le(s) modèle(s) de données doi(ven)t couvrir les fonctionnalités de base ainsi que les fonctionnalités des extensions choisies. Un texte explicatif doit accompagner votre modèle, pour expliquer l'utilité des tables et la logique sous-jacente.

☞ Comment réaliser un modèle de données relationnelles fait partie des prérequis pour ce projet. À titre d'illustration, la figure 2.1 montre un exemple d'un diagramme d'entité-relation, créé avec l'outil Visual Paradigm, pour une application de location de vidéo.

**Design du REST API.** Étant donné que le système à réaliser se base sur une API REST, il est également demandé de fournir un schéma ou modèle de votre API REST. Vous avez la liberté de choisir comment réaliser ce schéma. Certains outils permettent de faire un design visuel de l'API REST. Un exemple est illustré à la figure 2.1, en utilisant l'outil de modélisation Visual Paradigm.

**Maquette de l'interface utilisateur.** Un rapport présentant les **maquettes de l'interface utilisateur** doit être rendu. Il s'agit d'un prototype non fonctionnel de l'interface utilisateur illustrant de manière visuelle le design de la fonctionnalité de base et de chaque extension à réaliser. Cette maquette devrait présenter plusieurs visualisations illustrant les aspects importants
- de l'interface visuelle pour la fonctionnalité de base ;
- de l'interface visuelle pour chacune des extensions choisies.

9. `https://moodle.umons.ac.be`

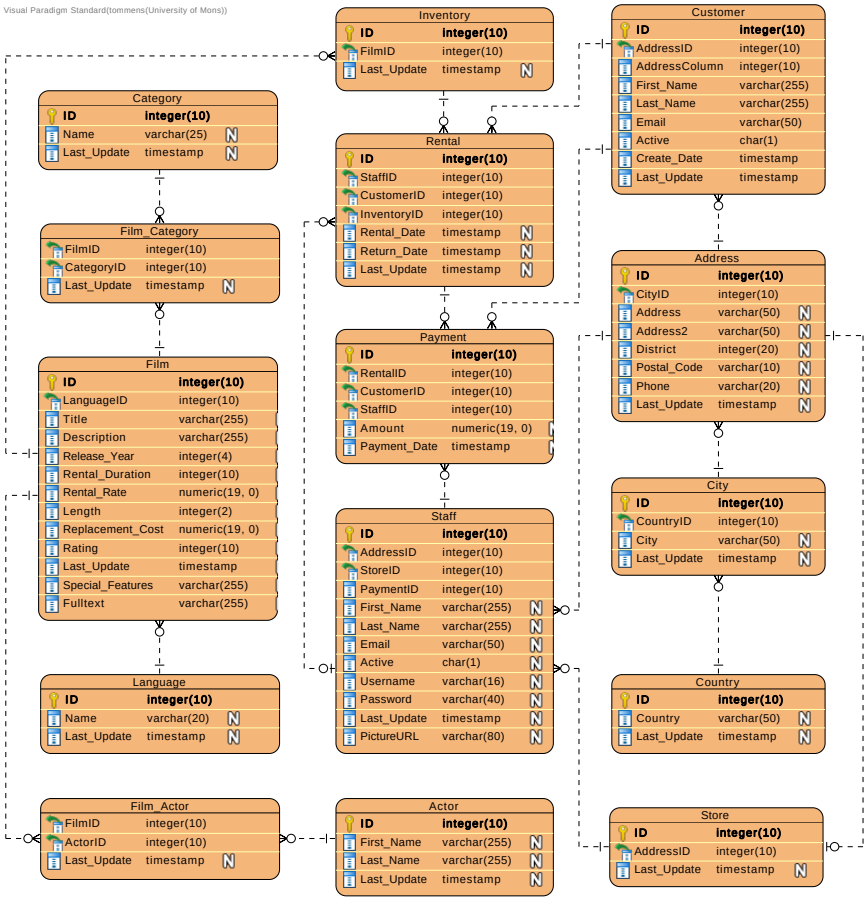


FIGURE 8 – Exemple d'un diagramme d'entité-relation pour représenter un modèle de données relationnelles.

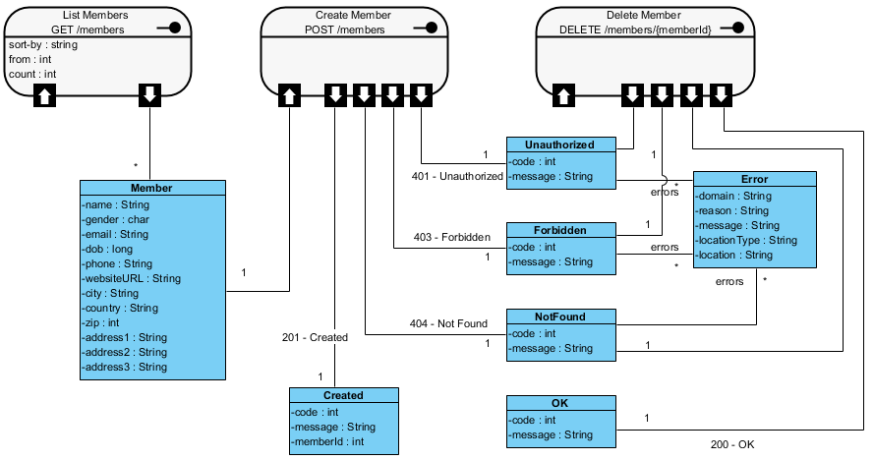


FIGURE 9 – Exemple d'un schéma visuel d'une API REST.

Les visualisations doivent être accompagnées d'un texte décrivant bien la façon d'interagir avec l'application afin de réaliser les fonctionnalités.

**Phase 2 (en Q2) : Projet de développement logiciel (S-INFO-106)**
Cette phase concerne l'implémentation, les tests et le déploiement du projet logiciel.

**Code source.** L'application serveur sera implémenté en utilisant le langage de programmation Java 17. L'implémentation doit suivre les principes de l'*orienté objet*, en utilisant de manière correcte le mécanisme de l'héritage, et en utilisant des design patterns. L'implémentation doit également suivre une approche de *programmation défensive* en utilisant correctement le mécanisme de gestion d'exceptions de Java. Le code Java doit être documenté avec JavaDoc. [10]

**Interface graphique.** L'interface graphique (GUI) des applications clients web doit être réalisée avec de la technologie JavaScript récente, compatible avec la plupart des navigateurs web récents.

**Tests unitaires.** Nous insistons sur l'utilisation d'un processus de *test-driven development* [11], en utilisant obligatoirement les *tests unitaires* tout au long du développement. L'utilisation d'un framework de test unitaires (pour Java nous imposons JUnit) est obligatoire, et l'utilisation d'un framework de *mock objects* pour compléter les tests unitaires est fortement recommandée. (Voir la section 1.4.3.)

## 2.2 Organisation et conditions du travail

**Gestion de versions.** Les groupes doivent obligatoirement utiliser un système collaboratif de contrôle de versions (git) pour gérer l'évolution du code source de leur application. Les enseignants doivent avoir accès en lecture aux dépôts des étudiants tout au long du projet, pour surveiller le travail et les progrès de chaque groupe et de ses membres. Dans le cas où, pour une raison quelconque, un étudiant venait à se retrouver seul dans son groupe, l'utilisation du système de contrôle de versions resterait néanmoins obligatoire. Des "commits" réguliers doivent être effectués.

**Travail en groupe.** Chaque groupe travaillera **séparément** à l'analyse, à la conception et au développement du projet. Chaque membre du groupe travaillera de manière individuelle à son extension. Toute collaboration entre les groupes est interdite sauf pour des discussions techniques spécifiques, si approuvées et modérées par les enseignants.

**Contraintes de temps.** Les livrables à rendre pour chaque étape clé doivent être déposés sur Moodle avant chaque échéance imposée. Puisque l'un des objectifs du projet consiste à apprendre à respecter les échéances imposées lors d'un projet de développement logiciel, *aucun retard ne sera toléré*.

**Exigences de qualité.** La fonctionnalité, la complétude et la qualité des livrables seront évaluées par les enseignants après l'échéance imposée pour chaque étape clé.

## 2.3 Outils

**Outil de modélisation.** Vous avez le libre choix des outils de modélisation. Beaucoup d'outils commerciaux ou open source sont disponibles sous la forme de stand-alone ou de plug-in pour des plateformes de développement intégré.

Pour concevoir des maquettes graphiques il existe plusieurs éditeurs graphiques.

L'UMONS possède une licence académique pour Visual Paradigm [12]. Cet outil supporte la modélisation et la conception des systèmes logiciels, incluant à la fois la modélisation UML et la modélisation des données avec les diagrammes d'entité-relation [13]. Il y a même du support pour créer des maquettes de l'interface utilisateur de votre système, et des visualisations pour faire le design d'un API REST. [14]

10. `https://en.wikipedia.org/wiki/Javadoc`
11. `http://www.agiledata.org/essays/tdd.html`
12. https ://ap.visual-paradigm.com/university-of-mons
13. `https://www.visual-paradigm.com/solution/dbdesign/best-er-diagram-tool/`
14. `https://www.visual-paradigm.com/solution/rest-api-design-tool/`

**Outils de développement.** L'application serveur doit **obligatoirement** être implémentée en Java 17, la dernière version de type long-term support (LTS) pour Java. [15] Vous pouvez librement choisir votre *environnement de développement* (par exemple IntelliJ IDEA, Eclipse ou NetBeans) à condition que les enseignants qui évalueront l'application puissent facilement compiler et exécuter votre logiciel sans devoir installer cet environnement.

Pour réaliser les *tests unitaires* de l'application serveur et ses extensions, le framework de test JUnit 5 est obligatoire. [16] Pour faciliter l'écriture des tests unitaires, il est fortement recommandé d'utiliser une librairie de mocking. Nous recommandons l'utilisation de Mockito [17] mais d'autres librairies sont envisageables (par exemple, EasyMock [18]).

☞ L'utilisation de librairies tierces supplémentaires peut être envisagée. Vous devez cependant demander et obtenir l'accord explicite des enseignants avant tout usage d'une telle librairie.

**Moteur de production automatique.** L'utilisation du moteur de production gradle [19], dans sa version 7.5, est obligatoire. C'est un outil en logiciel libre facilitant la gestion et l'automatisation de cycles de production de projets logiciels. Il permet aux étudiants et enseignants la compilation du code source et l'exécution de votre projet, sa documentation et ses tests unitaires à partir de la ligne de commande sans avoir à installer un environnement de développement Java quelconque. Vous devrez utiliser cet outil pour gérer vos dépendances logicielles, de sorte qu'un appel par ligne de commande suffise à valider, compiler, tester, et packager l'application de base ainsi que chaque extension séparément depuis un nouvel environnement de travail.

On peut trouver des tutoriels sur gradle ici : `https://gradle.org/guides/`
Et voici un tutoriel spécifiquement destiné à la production des applications Java :
`https://guides.gradle.org/building-java-applications/`

Il est fortement conseillé d'également utiliser le "Gradle wrapper" (gradlew : `https://docs.gradle.org/current/userguide/gradle_wrapper.html`) pour gagner encore en robustesse dans le processus de déploiement.

**Outil de gestion de versions.** L'utilisation de git [20] est imposée pour la gestion des versions et le travail collaboratif. Chaque groupe devra créer un dépôt **privé** sur la plateforme de son choix (exemples : GitHub, BitBucket, GitLab) et donner aux enseignants l'accès à celui-ci suivant les consignes qu'ils auront données. Un groupe n'aura pas accès aux dépôts des autres groupes. Les enseignants auront constamment accès aux dépôts fournis. Utilisez-les intelligemment, en respectant les bonnes pratiques. Au sein d'une branche, le code source sera placé dans un répertoire, dans le respect des conventions proposées par gradle. Toutes vos classes et autres ressources devront appartenir (directement ou indirectement) au package `be.ac.umons.gXX` où XX est le numéro de groupe.

**Système d'exploitation.** Vous pouvez utiliser n'importe quel système d'exploitation pour développer l'application. La seule contrainte est que tous les livrables de la phase d'implémentation soient indépendants de la plateforme choisie. Le code exécutable sera testé sur trois systèmes d'exploitation différents : macOS X, Linux et Windows.

Une attention particulière doit être portée aux problèmes d'encodage des caractères qui rendent les accents illisibles sur certains systèmes d'exploitation. Un autre problème récurrent est l'utilisation des chemins représentant des fichiers : Windows utilise une barre oblique inversée (backslash) tandis que les systèmes dérivés d'Unix utilisent une barre oblique (slash). La constante `File. separator` donne une représentation abstraite du caractère de séparation.

15. `https://en.wikipedia.org/wiki/Java_version_history`
16. `https://junit.org/junit5`
17. `https://site.mockito.org`
18. http ://easymock.org
19. `https://gradle.org`
20. `http://git-scm.com`

# 3 Livrables du projet

Chaque phase du projet aura ses propres livrables. Si le livrable est un rapport, nous conseillons vivement d'utiliser LaTeXpour sa rédaction. [21]

## 3.1 Phase d'analyse et de conception (S-INFO-015)

La conception (*design*) de l'application sera modélisée de manière individuelle par chaque groupe en utilisant le langage de modélisation UML 2.5 et en proposant une maquette de l'interface utilisateur.

**Livrable 1 : les maquettes de l'interface utilisateur.** Une maquette présente de manière visuelle l'interface utilisateur de l'application à réaliser, et la façon d'interagir avec cette interface. Elle sera réalisée au moyen d'une application dédiée choisie par le groupe. La maquette visuelle doit être accompagnée d'une description textuelle décrivant la façon d'utiliser le système logiciel.

Plusieurs maquettes doivent être rendues : une pour la fonctionnalité de base de chaque application web ; et une pour chaque extension individuelle à réaliser.

**Livrable 2 : le rapport de modélisation.** Ce rapport contient les différents modèles de conception. Ceci inclut les diagrammes UML, ainsi qu'un modèle de données relationnelles et un schéma de l'API REST. Chaque modèle doit être accompagné par une description textuelle justifiant les choix de conception qui ont été faits pour chaque diagramme :

- Un ou plusieurs diagrammes d'entité-relation, décrivant le(s) modèle(s) de données relationnelles.
- Un schéma ou modèle décrivant le design de l'API REST qui sera proposé.
- Un ou plusieurs diagrammes de cas d'utilisation ainsi qu'un ou plusieurs interaction overview diagram présentant la vue d'ensemble des interactions entre les cas d'utilisation.
- Les diagrammes de classes et de paquetage présentant le design et la structure de l'application.
- Les diagrammes de séquences représentant le comportement de l'application.

☞ Pour faciliter la lecture, les images des diagrammes UML dans le rapport doivent utiliser un fond blanc ou transparent. Le contenu des diagrammes doit rester lisible sans soucis après impression du rapport en noir et blanc.

## 3.2 Phase de développement logiciel (S-INFO-106)

Chaque groupe doit travailler de manière séparée lors de la phase de développement, en respectant toutes les licences (open source ou autres) des classes et librairies tierces utilisées.

☞ **Le plagiat** (du code venant des autres groupes, d'Internet, ou de n'importe quelle autre source) **sera lourdement pénalisé** et peut mener à un échec total pour le projet ou même pour l'année d'études (cf. règlement des études).

Si vous désirez réutiliser du code (p. ex., des librairies) open source dans le cadre de votre projet, vous devez obtenir l'accord préalable des enseignants, et vous devez indiquer cela clairement dans les commentaires et la documentation de votre code source ainsi que dans vos rapports.

**Livrable 1 : le code source.** Une version complète du code source (des applications clients et serveur) et de l'interface graphique de la fonctionnalité de base et des extensions.

21. L'outil LaTeX sera également utilisé pour la rédaction du projet et du mémoire du master en sciences informatiques.

**Livrable 2 : le code compilé.** Une configuration gradle permettant de compiler automatiquement, et sans erreurs, le code exécutable de la fonctionnalité de base de l'application, ainsi que de chaque extension individuellement. Des versions précompilées (complètes, auto-exécutables et indépendantes de la plateforme) sous la forme d'archives .jar, de la fonctionnalité de base, ainsi que de chaque extension individuellement. Attention à bien inclure les dépendances éventuelles dans vos archives .jar (via gradle). N'hésitez pas à tester l'exécution de vos archives sur des machines "neutres" autres que celles que vous utilisez pour vos développements.

**Livrable 3 : la documentation du code.** Le code Java doit être documenté avec JavaDoc. Celle-ci doit permettre au lecteur une compréhension d'ensemble du système développé. La JavaDoc doit contenir une documentation complète pour le projet ; les packages ; les classes, interfaces et énumérations ; les méthodes principales. La configuration gradle doit permettre de générer automatiquement cette documentation.

**Livrable 4 : les tests unitaires.** Les tests unitaires en JUnit doivent être présents, et la suite de tests doit être exécutable automatiquement avec gradle. Cette exécution ne peut pas donner lieu à des échecs ou des erreurs.

Les *tests unitaires* doivent être utilisés tout au long de la phase d'implémentation. Une approche de développement dirigé par les tests est imposée : l'écriture du code doit être alternée avec celle des tests, et tous les tests doivent réussir avant de progresser dans l'écriture du code. Une autre bonne pratique qui sera évaluée est l'utilisation de *tests de régression*. Pour chaque défaut (erreur, bug, mauvais comportement) rencontré, un test unitaire doit être écrit qui met en évidence ce défaut. Vous n'aurez ainsi pas à comprendre et résoudre deux fois le même problème. Les tests doivent être fournis pour les fonctionnalités de base ainsi que pour chaque extension réalisée.

**Livrable 5 : le rapport d'implémentation.** Le rapport d'implémentation doit contenir une motivation des choix d'implémentation par rapport à la phase d'analyse et de conception. Si le design établi lors de cette phase précédente a été revu, les modifications proposées doivent être présentées et dûment justifiées.

Le rapport d'implémentation doit aussi inclure : une description et une motivation des design patterns utilisés dans le code ; une description des algorithmes importants ; les commandes gradle pour générer le code compilé et la documentation, ainsi que pour exécuter le code et les tests. Si des données spécifiques (par exemple, des logins ou mots de passe) sont requises pour exécuter ou tester le code, le rapport de modélisation doit le préciser.

**Livrable 6 : une vidéo du manuel d'utilisation.** Une **vidéo** d'environ 10 à 15 minutes, accessible sur internet par un lien URL précisé dans le rapport d'implémentation (et accessible aux enseignants sans installation de logiciel ou inscription à un quelconque service). La vidéo doit être accompagnée par une explication orale, et doit illustrer *toutes les fonctionnalités de base ainsi que les fonctionnalités de chaque extension.* Cette vidéo doit également servir comme mode d'emploi expliquant le fonctionnement de l'application et des extensions. Chaque utilisateur doit être capable d'utiliser l'application après avoir vu la vidéo. Les éventuels problèmes, bugs, ou fonctionnalités manquantes doivent être mentionnés dans la vidéo.

**Défense orale de l'implémentation.** Le projet se termine par une défense orale durant laquelle le code exécutable sera présenté aux enseignants, et le code source ainsi que les tests unitaires seront défendus. Les enseignants donneront un feedback oral et écrit sur le contenu et la qualité du projet.

# 4 Critères de recevabilité

☞ Cette check-list reprend l'ensemble des consignes à respecter pour la remise du projet.
Le non-respect des critères implique la non-recevabilité du projet. Celui-ci sera dès lors considéré comme non rendu et l'étudiant sera sanctionné par une note de 0/20 à la partie rendue !

**Respect des échéances** La date et heure de limite de remise est indiquée sur Moodle. Aucun retard ne sera toléré. Il vous est conseillé d'uploader des versions intermédiaires avant la version définitive. Seule la dernière version rendue avant l'échéance sera évaluée.

**Absence de plagiat** Toute présence de plagiat implique la non-recevabilité du projet. Un outil automatisé sera utilisé afin de vérifier l'absence du code dupliqué entre les différents groupes.

**Formats d'archive et de fichier** Lors de chaque phase du projet, le travail devra être remis sous forme d'une seule archive **.zip** dont l'intitulé aura le format suivant :
— Pour la phase d'analyse et de conception : PAC -<numéro de groupe>. zip
— Pour la phase de développement logiciel : PDL -<numéro de groupe>. zip
Les rapports contenus dans chaque archive doivent être en format **PDF**. Chaque rapport doit commencer par une page de garde indiquant l'intitulé du rapport, l'année académique, le numéro du groupe, et pour chaque étudiant membre du groupe le nom, le matricule et l'extension choisie.

☞ Les images intégrées dans les rapports de chaque phase doivent être bien lisibles, en format A4, même après impression en noir et blanc. (Vous pouvez toujours joindre les images originales et leurs sources dans le dossier contenant le rapport.)

**Contenu d'archive pour la phase d'analyse et de conception** Votre archive devra obligatoirement contenir les éléments suivants :
— Le rapport de *modélisation* incluant toutes les images et portant le nom
PAC<numéro de groupe>-rapport.pdf
— Les *maquettes de l'interface utilisateur* portant le nom
PAC<numéro de groupe>-maquette<numéro de la maquette >.pdf

☞ Veillez à bien séparer la partie commune (fonctionnalité de base réalisée en groupe) et la partie individuelle (extensions réalisées individuellement par chaque étudiant).

**Contenu d'archive pour la phase de développement logiciel** Votre archive doit obligatoirement contenir les éléments suivants :
— Le rapport d'*implémentation* incluant toutes les images et portant le nom
PDL<numéro de groupe>-rapport.pdf
— Une (ou plusieurs) vidéos, accessible(s) aux enseignants par un lien internet (dont l'URL est précisée dans le rapport d'implémentation), présentant *toutes les fonctionnalités de base et de chaque extension*, et illustrant comment l'application peut être utilisée, tout en mentionnant les éventuels problèmes ou manquements restants.
— Le code source et la documentation JavaDoc contenant tous les fichiers nécessaires à sa compilation ainsi qu'à son exécution.
— Les tests unitaires contenant tous les fichiers nécessaires à sa compilation ainsi qu'à son exécution.

☞ La compilation du code source, de la documentation, de l'archive auto-exécutable et des tests unitaires doit être réalisable au moyen d'une commande gradle ou gradlew sans que cela ne génère d'erreur. Le rapport doit préciser les commandes à utiliser.
☞ Veillez à bien séparer la partie commune (fonctionnalité de base réalisée en groupe) et la partie individuelle (extensions réalisées individuellement par chaque étudiant).

# 5 Critères d'évaluation

Tous les étudiants du même groupe obtiendront une note qui tient compte du travail individuel (l'extension réalisée par chaque étudiant) et le travail en commun (les fonctionnalités de base). Les livrables rendus seront évalués selon les critères suivants :

## 5.1 Pour la phase d'analyse et de conception

### 5.1.1 Maquette de l'interface utilisateur

La maquette couvre-t-elle tous les aspects de l'énoncé ? Toutes les fonctionnalités sont-elles prises en compte ? La maquette de l'interface graphique est-elle conviviale, intuitive et ergonomique ?

Une maquette n'est pas juste un ensemble d'images représentant comment on propose de visualiser l'interface graphique. Les images doivent obligatoirement être accompagnées du texte expliquant comment l'utilisateur est censé utiliser l'interface graphique pour réaliser les fonctionnalités de base et des extensions fournies par les applications clients.

### 5.1.2 Rapport de modélisation

**Modèle de données**

Le modèle de données couvre-t-il tous les aspects de l'énoncé ? Toutes les exigences sont-elles prises en compte ?

Les *diagrammes d'entité-relation* (ERD) sont-ils syntaxiquement et sémantiquement cohérents ?

Les ERD présentent-ils toutes les entités, leurs attributs et relations ? Les multiplicités, clés primaires et étrangères sont-elles bien représentées ? Les tables incluses dans le diagramme couvrent-elles toutes les fonctionnalités envisagées ? Observe-t-on de mauvaises pratiques ? Des optimisations sont-elles possibles ?

**Design de l'API REST**

Le schéma ou modèle décrivant le design de l'API REST est-il complet et compatible avec les fonctionnalités demandées ?

**Modèles UML**

**Conformité au cahier des charges / Complétude** Les diagrammes UML utilisés sont-ils complets ? Couvrent-ils tous les aspects de l'énoncé ? Toutes les exigences (fonctionnelles et non fonctionnelles) sont-elles prises en compte ?

**Style** Les diagrammes sont-ils bien structurés ? Les diagrammes UML respectent-ils la version de la norme imposée ? Suivent-ils un style de conception orientée objet ? (Par exemple, une bonne utilisation de la généralisation et de l'association entre les classes, utilisation d'interfaces et de classes abstraites, une description des attributs et des opérations pour chaque classe.)

**Cohérence** Les diagrammes UML sont-ils syntaxiquement et sémantiquement cohérents ? N'y a-t-il pas d'incohérences : (i) dans les diagrammes (p. ex. : opérations et attributs dans les classes, multiplicités sur les associations, ...) ; (ii) entre les différents diagrammes ? (p. ex. Les objets et messages dans un diagramme de séquences correspondent-ils aux classes et opérations dans le diagramme de classes ?)

**Exactitude** Les différents éléments d'un diagramme sont-ils utilisés correctement ? Par exemple :

1. Dans un *diagramme de cas d'utilisation*, les acteurs sont-ils correctement définis ? Les notions de généralisation, d'extension et d'inclusion sont-elles correctement mises en œuvre ? Fait-on appel aux points d'extension lorsqu'ils sont nécessaires ?
2. Dans un *diagramme de classes*, les multiplicités sur les associations sont-elles judicieusement utilisées ? L'utilisation de la composition, de l'agrégation et de l'association est-elle pertinente ? Les responsabilités (fonctionnalités) sont-elles bien distribuées aux différentes classes ? Observe-t-on

la présence de *design patterns* et l'absence d'*antipatterns* (par exemple des "god class") ? Les opérations et attributs définis dans chaque classe sont-ils suffisants pour permettre à l'application de fonctionner correctement ?

3. Dans un *diagramme de séquence*, les opérations appelées correspondent-elles à celles décrites dans le diagramme de classes ? Les objets commencent-ils et finissent-ils leur vie au bon moment ? Les objets communiquant entre eux sont-ils connectés ensemble ? Les fragments combinés sont-ils utilisés correctement pour représenter des boucles, des conditions, des exceptions, du parallélisme ?

**Compréhensibilité** Les diagrammes sont-ils faciles à comprendre ? Ont-ils le bon niveau de détail ? Pas trop abstraits, pas trop détaillés ?

**Lisibilité** Les diagrammes ne peuvent pas être dessinés à la main et doivent être lisibles sans devoir agrandir le document.

## 5.2 Pour la phase de développement logiciel

### 5.2.1 Qualité du code source

**Conformité au cahier des charges** Toutes les fonctionnalités requises sont-elles implémentées ?

**Conformité au design** Le code source est-il cohérent avec les modèles UML ?

**Lisibilité** Le code est-il facile à lire et comprendre ?

**Style** Le code Java respecte-t-il la version Java imposée ?
Le code est-il bien structuré (en packages, classes, méthodes, ...) ?
Le code suit-il les principes de la *programmation orientée objet*, en utilisant le mécanisme de typage, héritage, polymorphisme, liaison tardive, interfaces, classes et méthodes abstraites... ? Le code n'est-il pas inutilement complexe ? À tout moment, il faut éviter un style procédural avec des méthodes complexes et beaucoup d'instructions conditionnelles.
Le code suit-il un *style de programmation défensive ?* (Afin de réduire les erreurs lors de l'exécution du programme, le programme doit utiliser au maximum les mécanismes de typage, d'exceptions et d'assertions.)
Les *design patterns* sont-ils utilisés lors de l'implémentation ?
Les *conventions de nommage* sont-elles respectées ? [22]
La *duplication de code* et la présence de *code mort* sont-elles évitées ?

### 5.2.2 Testabilité

Les tests unitaires accompagnant le code source seront évalués selon les critères suivants :

**Processus** Le processus de test-driven development a-t-il été suivi ?

**Couverture** Les tests unitaires couvrent-ils toute la fonctionnalité requise ? Y a-t-il des tests superflus ? Toutes les classes et méthodes importantes sont-elles couvertes par des tests ?

**Exactitude** Les tests sont-ils corrects ? Tous les tests unitaires fonctionnent-ils sans échecs ni erreurs ?

**Qualité** Les tests sont-ils de bonne qualité (pas de tests trop simples, des tests avec un comportement trivial, ...) ? Y a-t-il une librairie de *mocking* qui a été utilisée pour la création des *mock objets* lors des tests comportementaux ?

**Automatisation** Est-il possible, grâce à gradle, d'exécuter tous les tests unitaires en une seule fois ?

22. `https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html`

### 5.2.3 Exécutabilité et fonctionnalité

**Compilation** Y a-t-il des problèmes ou warnings lors de la compilation ? Est-il possible de compiler et exécuter le code (et les tests et le JavaDoc) avec gradle ?

**Ergonomie et convivialité** L'application (et son interface graphique) est-elle conviviale, intuitive et facile à utiliser ?

**Fluidité et efficacité** L'utilisation de l'application est-elle fluide et performante ? N'y a-t-il pas de temps d'attente inutile ?

**Complétude** Toutes les exigences du cahier des charges sont-elles prises en compte par l'application ?

**Exactitude et fiabilité** Le programme ne doit pas contenir de bugs, et ne doit pas planter.
L'application fonctionne-t-elle correctement dans des circonstances normales ?
L'application fonctionne-t-elle correctement dans des circonstances exceptionnelles (p. ex., données erronées, format de données incorrect, problème de réseau, problème de sécurité, ...) ?

**Indépendance de la plate-forme** Le code produit est-il exécutable sans aucun souci sur trois systèmes d'exploitation différents (macOS X, Linux et Windows) ? N'y a-t-il pas de problème avec des caractères spéciaux, chemins de fichier, ... ?

### 5.2.4 Documentation

**Rapport d'implémentation** Le rapport d'implémentation du projet est-il bien rédigé (style d'écriture, facilité de lecture, structure, ...) ? Justifie-t-il tous les choix d'implémentation importants (p. ex. les librairies externes utilisées, les design patterns utilisés, les déviations éventuelles du cahier de charges, les fonctionnalités non implémentées, les problèmes rencontrés, les limitations de l'application réalisée) ?

**Vidéo** Les vidéos fournies présentent-elles toutes les fonctionnalités de base, ainsi que les fonctionnalités de chaque extension ? Sont-elles suffisamment complètes pour qu'un utilisateur puisse utiliser l'application sans problème ? Contiennent-elles une description orale du déroulement de l'application ? Les éventuels problèmes ou manquements restants sont-ils bien mentionnés ?

**JavaDoc** Le code Java est-il bien documenté avec JavaDoc ? La JavaDoc est-elle présente et est-elle suffisamment complète ? Est-elle de bonne qualité ? Les informations qu'on trouve dedans sont-elles pertinentes et utiles pour quelqu'un qui veut comprendre ou modifier le code ? Est-il possible, grâce à gradle, de générer la JavaDoc ?

# 6 Dates importantes

Le projet est jalonné de dates reprises ci-dessous. Dans le cas où la date est associée à la remise d'un livrable, elle constitue l'extrême limite de remise du livrable sur la plateforme Moodle. Aucun retard ne sera toléré.

**S-INFO-015 "Projet d'analyse et de conception"** (Bloc 2 Bachelier Sciences Informatiques)

**Le 26 septembre 2022** Présentation du projet et distribution de l'énoncé. Proposition de formation des groupes.

**Le 10 octobre 2022** Les groupes sont définitivement formés. Début de la phase d'analyse et de conception.

**Vendredi 16 décembre 2022** Remise des livrables pour la phase d'analyse et de conception :
Rapport de modélisation : UML + modèle de données + design de l'API REST
Maquette de l'interface utilisateur
Le livrable doit bien distinguer la maquette et les modèles pour la fonctionnalité de base (réalisée en groupe) et les extensions (réalisées de manière individuelle).

**Feedback** *Un feedback écrit sera donné aux étudiants concernant la note obtenue et les points à améliorer.*

**S-INFO-106 "Projet de développement logiciel"** (Bloc 2 Bachelier Sciences Informatiques) (Bloc complémentaire Master Sciences Informatiques)

**Lundi 6 février 2023** Début de la phase de développement logiciel.

**Lundi 20 mars 2023** Remise d'un prototype / proof-of-concept du système logiciel à réaliser. Ce prototype correspond à une pré-version qui doit inclure le bon fonctionnement et interaction entre toutes les briques technologiques (langages, outils, frameworks, libraries) qui ont été choisies et mises en place pour réaliser le système.

Pour illustrer cela, au moins une partie de la fonctionnalité de base doit être implémenté, permettant de montrer comment les applications web (clients frontend) fonctionnent et intéragissent par l'API REST avec l'application serveur (backend), et comment cette application serveur communique avec la base de données.

Lors de cette remise, nous vérifierons si votre travail rendu respecte plusieurs critères de recevabilité importants attendus :
Le prototype DOIT être installable, compilable, exécutable et testable avec gradle. Des tests unitaires DOIVENT déjà être présents pour tester le code (car vous devez suivre un processus de test-driven development). Du JavaDoc DOIT être présent pour documenter le code.

**Lundi 24 avril 2023** Remise finale des livrables pour la phase de développement logiciel :
Rapport d'implémentation
Vidéo de la fonctionnalité (manuel d'utilisation)
Scripts gradle, code source et exécutable, JavaDoc
Le livrable doit bien distinguer la fonctionnalité de base (réalisée en groupe) et les extensions (réalisées de manière individuelle).

**Mercredi 18 mai 2022** Défenses orales, tests d'acceptation par les enseignants.
**Votre présence est obligatoire ! (Absence injustifiée = 0/20)**