



DOSSIER DE VALIDATION

DEVELOPPEUR FULL STACK (DFS)

Titre de niveau 6 inscrit au RNCP

Titre :

Démontrer l’acquisition des compétences décrites dans le référentiel de la certification visée

Marc DE LARREA

24/11/2023

Tuteur : Cédric NOTIN

# Remerciements

Je tiens à exprimer ma gratitude envers toutes les personnes qui m'ont soutenu et aidé tout au long de mon parcours en tant que développeur Full Stack. Leur soutien et leurs précieux conseils ont été essentiels pour mon apprentissage et ma réussite.

Tout d'abord, je souhaite remercier chaleureusement **Baptiste Coulaud**, mon mentor, qui m'a initié à ce vaste monde du développement et m'a transmis ses connaissances avec passion. Sa patience et sa disponibilité ont été d'une aide précieuse, et je lui suis reconnaissant de m'avoir encouragé à me dépasser constamment. C’est un honneur pour moi que d’avoir été l’élève d’un développeur aussi talentueux que lui.

Un grand merci également à **mes collègues de travail en formation**, qui m'ont accompagné jour après jour dans cette aventure. Leur expertise, leur soutien et leur esprit d'équipe ont créé un environnement propice à l'apprentissage et à l'épanouissement professionnel. Je suis reconnaissant d'avoir eu la chance de travailler avec des personnes aussi talentueuses et bienveillantes.

Je tiens également à exprimer ma reconnaissance envers l**'IT-Akademy**, mon organisme de formation. Leurs programmes d'apprentissage complets et leurs formateurs compétents m'ont permis de devenir un meilleur développeur jour après jour. Je suis reconnaissant envers toute l'équipe pédagogique pour leur engagement et leur dévouement.

Enfin, je souhaite remercier mon entreprise, **ARC Europe France**, qui m'a offert l'opportunité de mettre en pratique mes compétences et d'approfondir mes connaissances. Leur confiance en moi et leur soutien constant ont été déterminants dans ma progression professionnelle. Je suis reconnaissant envers toute l'équipe d'ARC France pour leur accompagnement et leur encouragement. Un grand merci tout particulier à **Aurélien Vaudan** pour toutes les explications et aides qu’il m’a apporté jour après jour dans la société. C’est un plaisir de travailler avec des personnes si bienveillantes et attachantes que cet homme.

# Avant-Propos

Ce rapport a été réalisé dans le cadre de ma formation en tant que développeur Full Stack à l' IT-Akademy, du 17 novembre 2022 au 24 novembre 2023. Son objectif est de valider les différents blocs de compétences requis pour l'obtention du diplôme.

Mon parcours personnel est marqué par une diversité de métiers exercés, où la minutie et la bienveillance ont toujours été les maîtres-mots. J'ai eu l'opportunité d'explorer plusieurs domaines et de devenir compétent dans ceux-ci. Cependant, ma passion pour l'apprentissage et mon désir d'aider les autres ont continué de grandir.

C'est pourquoi j'ai décidé de me lancer dans un nouveau défi, un défi de taille : devenir un développeur accompli. J'ai observé les incroyables possibilités offertes par le monde numérique et j'ai décidé de m'immerger dans cet univers en constante évolution.

Au cours de cette formation, j'ai eu l'opportunité d'apprendre les bases fondamentales du développement, de me familiariser avec les langages de programmation et les technologies les plus couramment utilisés. J'ai également acquis une compréhension globale de la conception et du déploiement d'applications web, ainsi que de la gestion des bases de données.

Je tiens à remercier chaleureusement l' IT-Akademy pour la qualité de son programme de formation et l'encadrement pédagogique attentif de ses formateurs. Leur expertise et leur passion pour le développement m'ont inspiré et stimulé tout au long de ce parcours.

J'aborde ce projet de soutenance avec enthousiasme et détermination.

Je suis fier du chemin parcouru jusqu'à présent et je suis convaincu que cette expérience en tant que développeur Full Stack me permettra d'ouvrir de nouvelles perspectives professionnelles passionnantes.

Ce rapport témoigne de mes compétences, de mes réalisations et de mon engagement dans cette formation.

Je tiens à exprimer ma gratitude envers tous ceux qui m'ont soutenu et encouragé dans cette aventure, ainsi qu'envers mes collègues de classe qui ont partagé cette expérience avec moi.

Table des matières

Pages :

[Remerciements 3](#_Toc149900337)

[Avant-Propos 4](#_Toc149900338)

[Table des matières 5](#_Toc149900339)

[Correspondance Référentiel 7](#_Toc149900340)

[Introduction 8](#_Toc149900341)

[Le projet IOT (Purpaws) de mon école IT-Akademy 8](#_Toc149900342)

[Le stage en entreprise (Arc Europe France) 8](#_Toc149900343)

[PROJET IOT (Internet des objets) 9](#_Toc149900344)

[Scrum 14](#_Toc149900345)

[Canva 15](#_Toc149900346)

[Les débuts du travail d’équipe 16](#_Toc149900347)

[1.Notion : 16](#_Toc149900348)

[2.Wireframes : 17](#_Toc149900349)

[3. Maquettes : 18](#_Toc149900350)

[4. UML du projet : 19](#_Toc149900351)

[5. Risques et Criticité 22](#_Toc149900352)

[Chef de projet 24](#_Toc149900353)

[Cahier des charges 25](#_Toc149900354)

[GitHub 25](#_Toc149900355)

[- DevOps : 27](#_Toc149900356)

[- CI/CD : 28](#_Toc149900357)

[- Création de branches localisées 28](#_Toc149900358)

[Maintien de Scrum 29](#_Toc149900359)

[Modéliser une maquette 30](#_Toc149900360)

[Composant Header.js 31](#_Toc149900361)

[Header.scss 32](#_Toc149900362)

[Visuels de l’application 33](#_Toc149900363)

[API 34](#_Toc149900364)

[Api.json 35](#_Toc149900365)

[Imprévu 37](#_Toc149900366)

[Etude de marché 38](#_Toc149900367)

[IA 38](#_Toc149900368)

[Présentation 39](#_Toc149900369)

[UX 40](#_Toc149900370)

[NotificationButton.js 41](#_Toc149900371)

[Mockoon 42](#_Toc149900372)

[Postman 43](#_Toc149900373)

[.env 44](#_Toc149900374)

[.gitignore 44](#_Toc149900375)

[SEO 45](#_Toc149900376)

[Accessibilité web et son importance pour notre site 47](#_Toc149900377)

[L’entreprise 48](#_Toc149900378)

[Déroulé du stage 49](#_Toc149900379)

[Cartographie Applicative Exacte 50](#_Toc149900380)

[Cartographie Applicative localisée 51](#_Toc149900381)

[Compléments 54](#_Toc149900382)

[Back API node express 54](#_Toc149900383)

[RGPD 56](#_Toc149900384)

[Plateforme d’hébergement 56](#_Toc149900385)

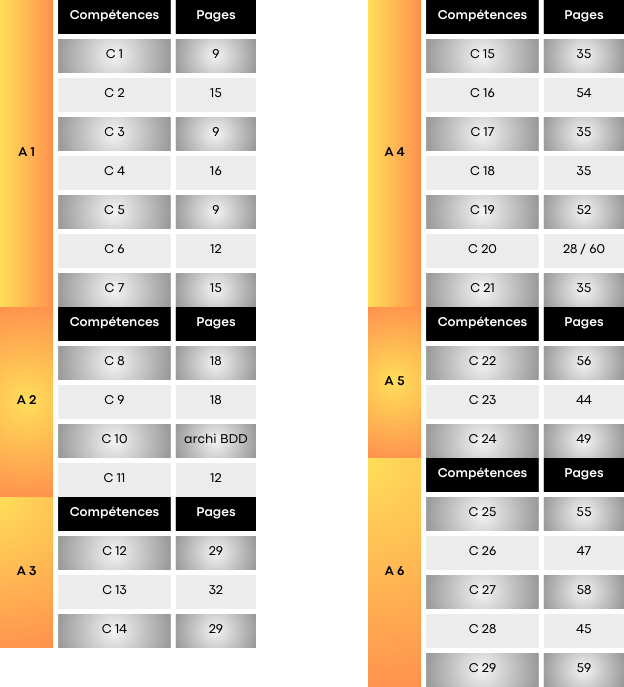
[SSIS 58](#_Toc149900386)

[Transactions PayPal 59](#_Toc149900387)

[Conclusion 62](#_Toc149900388)

[CV 63](#_Toc149900389)

# Correspondance Référentiel



# Introduction

## Le projet IOT (Purpaws) de mon école IT-Akademy

Au cours de ma formation à l’IT-Akademy, j'ai eu l'opportunité de participer à un projet passionnant axé sur l'IoT (Internet des Objets).

En tant que chef de projet, j'ai été chargé de superviser l'avancement et de gérer de manière Agile/Scrum le travail d'équipe.

Dès le premier jour, mes collègues de projet (Mel, Ryan, Yannis) m'ont désigné pour ce rôle en raison de mon expérience professionnelle, estimant que j'étais le mieux qualifié pour assumer cette responsabilité.

Dans le cadre de ce projet, nous avons dû identifier les technologies adaptées et concevoir une gamelle connectée pour chat. Grâce à cet exercice, nous avons pu mettre en pratique de nombreux concepts et compétences requis pour la validation de notre formation.

L’école se situe à Charbonnières-les-Bains, j’y étais de décembre 2022 au juillet 2023

## Le stage en entreprise (Arc Europe France)

Parallèlement à ce projet, j'ai également eu la chance d'effectuer un stage au sein de l'entreprise ARC.

ARC Europe France est une entreprise spécialisée dans le dépannage automobile. Mon travail au sein de cette société a été axé sur le développement et maintien d'une application destinée à aider les conducteurs en détresse (Cactus). J'ai eu l'occasion de travailler en équipe sur ce projet, contribuant ainsi à son développement et à son amélioration.

Mon stage s'est déroulé du 17 juillet au 24 novembre 2023.

# PROJET IOT (Internet des objets)

L' IT-Akademy nous a demandé de réaliser un projet IOT, nous avons donc (toute la classe) écrit sur des petits papiers des idées.

Après avoir lu et noté la pertinence des idées, nous avons voté pour celles qui nous intéressaient le plus et ensuite créé des équipes de 4.

J'ai personnellement choisi de participer à la création d'une gamelle pour chat connectée à son application web et mobile ainsi qu'à son site marchand. J'ai fait ce choix car, en tant que propriétaire d'un chat, j'ai pu facilement m'identifier aux problèmes potentiels des propriétaires qui ont des chats qui ne savent pas réguler leur appétit. Cette gamelle gère l'approvisionnement en croquettes des chats selon les paramètres souhaités par leurs maîtres. Le site marchand vend la gamelle, service tout en un.

Dans ce projet, la première étape a été la création de l'équipe. J'ai eu la chance de travailler avec d'excellents coéquipiers :

- **Ryan**, qui possède une logique impressionnante et une aisance remarquable en algorithmie,

- **Mel**, qui a un goût prononcé pour le front-end et une réflexion hors du commun,

- Enfin, **Yannis**, qui possède une expérience solide en programmation, un grand sens de l'humour, ainsi qu'une source de connaissances et de motivation très appréciable pour l'apprentissage.



Lorsque nous nous sommes concertés pour déterminer comment le projet allait se dérouler, il est apparu évident à mes coéquipiers que ma place serait celle de chef de projet.

En effet, mes expériences passées m'ont fait apparaître comme un leader à leurs yeux, et ils m'ont donc confié cette responsabilité sans même me laisser d'autre choix.

Cela m'a permis de remplir plusieurs tâches cruciales du projet, notamment :

- Arbitrer les décisions en cas de litiges ou d'indécisions,

- Collaborer avec l'équipe pour définir l'organisation et les objectifs (utilisation de Scrum et définition de l'objectif final), **(A1 C3)**

- Gérer les petits conflits qui peuvent survenir,

- Organiser le groupe afin de minimiser les éventuels ralentissements, **(A1 C5)**

- Évaluer les forces et les faiblesses de chacun afin d'équilibrer la répartition du travail et de garantir au maximum que le projet puisse être mené à bien dans les délais impartis.

La deuxième étape a été de déterminer comment nous allions réaliser une gamelle connectée pour chat intégrée à une application web. Cette phase a soulevé plusieurs questions essentielles : **(A1 C1)**

**1.** Quel langage de programmation devons-nous utiliser ?

**2.** Quelle architecture convient le mieux ?

**3.** Quelles technologies devons-nous intégrer ?

**4.** Comment construire le dispositif en lui-même ?

**5.** Quelles sont les exigences de l'école à ce sujet ?

**6.** Comment pouvons-nous travailler de manière efficace en équipe ?

**7.** Qui sera responsable de quelles tâches ?

Nous avons abordé chacune de ces questions en groupe, et j'ai suggéré une idée supplémentaire : la nécessité de documenter l'ensemble du processus, en créant une présentation.

En fin de compte, nous avons pris les décisions suivantes : **(A3 C12)**

**1. Langages de programmation :** **Python** pour la partie embarquée, **React** pour le front-end, et **PHP** pour le back-end.

Et l'interaction entre ces langages nécessitait une vue claire :



Le langage **python** sera utilisé pour faire le lien entre la gamelle et le serveur,

**PHP symfony** sera le langage serveur (**Backend**) et fera le lien entre la gamelle et l’application web(**Frontend**)

**React JS** sera le langage de l’application web et fera le lien entre le serveur et l’appli.

Nous avons préféré rester sur des langages de programmation que nous avions vus en cours, le temps nous faisant défaut. Connaître même un peu le langage étant un gain de temps, les meilleures options que nous avions étaient donc **PHP**, **React** et **Python**.

**2. Architecture MVC (Modèle-Vue-Contrôleur) : (C11)**

C'est un modèle de conception logicielle qui divise une application en trois composants interconnectés : le modèle, la vue et le contrôleur;

**- Le modèle** représente la logique de l'application et les données. Dans notre projet, le modèle est responsable de la gestion des données de la gamelle connectée, telles que les informations sur la nourriture, les horaires d'alimentation, le nom du chat, etc.

**- La vue** est responsable de l'interface utilisateur. Dans notre cas, il s'agit de l'interface web que les utilisateurs utiliseront pour interagir avec la gamelle connectée.

**- Le contrôleur** agit comme un intermédiaire entre le modèle et la vue. Il traite les demandes de l'utilisateur et met à jour le modèle en conséquence.

Par exemple, lorsqu'un utilisateur planifie l'alimentation de son chat via l'application, le contrôleur s'assure que ces informations sont enregistrées dans le modèle et que la vue est mise à jour pour refléter les changements.

L'architecture MVC offre une séparation claire des fichiers du code ainsi qu'une logique plus facile à entrevoir, ce qui facilite la maintenance et l'évolutivité de l'application.

**3. Utilisation d'un Raspberry Pi ainsi qu'une puce pour le suivi du chat :**

Le Raspberry Pi est un petit ordinateur monocarte abordable qui peut être utilisé dans une variété d'applications, y compris les projets IoT.

Dans notre projet, nous avons choisi d'utiliser un Raspberry Pi pour plusieurs raisons :

- Il est compact et économe en énergie, ce qui en fait un choix idéal pour un dispositif embarqué.

- Il dispose de ports d'entrée/sortie permettant de connecter des capteurs et des actionneurs.

- Il peut exécuter une version de Python, ce qui facilite la programmation de la partie embarquée de notre gamelle connectée.

Le Raspberry Pi sera responsable de la collecte de données à partir de capteurs sur la gamelle, comme le poids de la nourriture, et de l'envoi de ces données au serveur back-end via une connexion Internet. Cela permet aux utilisateurs d'accéder à ces informations via l'application web.

**4. Nous avons élaboré plusieurs plans** individuellement pour ne pas influencer les idées des autres membres, puis nous avons fusionné nos meilleures idées pour créer un prototype.



**5. L'école nous avait imposé certains langages**, mais nous avons pu justifier notre choix d'utiliser Python. Nos recherches sur les interactions entre les objets et les différentes nécessitées des connexions (serveurs, applications) pointaient vers ce langage.

L’école est également de fait notre client. L'IT-Akademy nous avait demandé de réaliser un projet IoT (Internet des objets) ainsi qu'un "Fil Rouge" qui consistait initialement en la création séparée d'un site marchand et d'un objet IoT.

Cependant, nous avons négocié avec l'école la possibilité de fusionner ces deux objectifs en un seul projet, en développant la gamelle connectée ainsi que son site marchand associé.

**6. Nous avons adopté la méthode agile Scrum** pour travailler ensemble de manière efficace tout au long de notre aventure.

# Scrum

La méthodologie Scrum est un cadre de travail agile qui favorise la collaboration, la transparence et l’adaptabilité dans le développement de logiciels. Dans notre projet, nous avons choisi d’adopter Scrum pour plusieurs raisons :

- Scrum nous permet de travailler de manière itérative et incrémentale, ce qui signifie que nous développons notre projet par étapes successives et fonctionnelles.

- Les "sprints" de Scrum sont des périodes de développement définies, généralement de deux à quatre semaines (dans notre cas, nous faisions un nouveau sprint toutes les deux semaines), au cours desquelles nous nous concentrons sur la réalisation d'objectifs spécifiques.

- Scrum encourage la communication régulière au sein de l'équipe, ce qui est essentiel pour résoudre rapidement les problèmes et s'assurer que le projet avance sans problème.

Au fil des jours, il m'est apparu évident que notre rythme et notre manière de travailler ensemble pouvaient encore être optimisés. J'ai donc décidé de tenir une brève réunion tous les soirs, d'une durée de 15 à 20 minutes, pour résumer la journée en trois points : **(A1 C6)**

- Qu'as-tu accompli aujourd'hui ?

- Quels problèmes as-tu rencontrés ?

- Qu'as-tu prévu de faire demain ?

Cela présentait plusieurs avantages :

- Chacun d'entre nous savait où en étaient les autres,

- Nous échangions sur nos problèmes, et si l'un d'entre nous avait une proposition, le problème était résolu facilement le lendemain matin, ce qui représentait un gain de temps et d'énergie,

- Le maintien des échanges contribuait à maintenir une atmosphère positive au sein de l'équipe.

**7. L’équilibre.** Nous avons constaté que nous étions parfaitement équilibrés avec deux membres spécialisés en front-end et deux en back-end, avec Ryan et Yannis pour le back-end, et Mel et moi pour le front-end.

## Canva

En parallèle, j'ai commencé sur Canva une présentation. Je me suis dit que garder une trace écrite de l’avancée du projet pourrait être une bonne idée.

Mes collègues ont débattu sur l’utilité de tout noter, tout enregistrer, coucher sur papier ou Canva tous les détails de notre périple. Ils ont pensé à l’unanimité que cela n’était pas nécessaire au vu des demandes clients actuelles.

Mais de mon point de vue, il était important de noter nos difficultés, nos choix, nos créations diverses.

Dans un premier temps, parce que j'étais fier de créer quelque chose en partant de rien, et ensuite pour garder une trace des différentes options qui ont été choisies, tout simplement pour ne pas oublier par où nous étions passés pour arriver à ces résultats.

J’ai donc préféré être prudent et continuer malgré tout à mettre une quantité non exagérée de données de côté (pour le cas où).

J’en ai également profité pour jeter un œil sur le travail de mes collègues, et parfaire ma compréhension des langages divers utilisés. **(C21)**



# Les débuts du travail d’équipe

Nous nous sommes répartis les tâches de la manière suivante, en vue de réaliser notre projet : **(A1 C2, C7)**

## 1.Notion :

Nous avons créé une section sur Notion. Notion est une plateforme de gestion de projet et de collaboration en ligne qui permet aux utilisateurs de créer, organiser et partager des informations de manière flexible.

Elle combine des fonctionnalités de prise de notes, de gestion de tâches, de création de bases de données et de collaboration en temps réel au sein d'une seule application **(A4 C21.2)**.

Nous avons réparti les tâches en fonction de nos forces et faiblesses. Étant le membre le moins difficile de l'équipe Front (le "Front-end" est, pour simplifier, la partie visible de l'application), j'ai laissé à ma collègue le soin d'attribuer les tâches et d'approuver ou de rejeter ses choix. J'ai finalement pris en charge les sujets les plus long à réaliser, tandis que Mel a pris en charge ceux qui nécessitaient le plus de liens de redirection et de travail sur le CSS (le CSS est ce qui permet de modifier l'aspect visuel des différents éléments de l'application).



## 2.Wireframes :

Les wireframes sont des schémas de conception initiale de l'interface de l'application. **(A1 C4)**

Ils nous ont permis de visualiser à quoi ressemblerait l'application avant de passer à la phase de développement.

## 3. Maquettes :

Les maquettes sont des représentations plus détaillées de l'interface, montrant les éléments de design, les couleurs et les emplacements des éléments de l'application.

Elles nous ont permis d'avoir une idée précise du rendu final de l'application.

Elles ont également soulevées une question essentielle, la charte graphique.

Nous avons choisi en couleur principale le bleu, et en couleur secondaire l’orange. Le bleu a été choisi d’un commun accord, la couleur du ciel, de la liberté.

Pour ce qui est de l’orange, Mel nous a indiqué que c’était une couleur chaleureuse qui répondait bien avec le bleu, nous avons donc tous accepté son choix sans plus de questions.

Et donc, voici nos maquettes :



## 4. UML du projet :

Nous avons utilisé l'**UML** (Unified Modeling Language) pour modéliser différents aspects du projet, notamment les diagrammes de cas d’utilisations, les diagrammes de séquences, la spécification des besoins, la Méthode d'Organisation et d'Analyse **(MOA),** le modèle conceptuel de données **(MCD)** et le modèle physique de données **(MPD).** **(A2 C8, C9, C10, C11)**

Petite explication :

**Diagrammes de Cas d'Utilisation** : Les diagrammes de cas d'utilisation sont des outils de modélisation UML qui permettent de décrire les interactions entre un système (comme une application ou un site web) et ses utilisateurs. Ils montrent les différentes actions ou fonctionnalités que les utilisateurs peuvent effectuer dans le système et comment ces actions sont liées les unes aux autres. Ces diagrammes nous ont été utiles pour comprendre comment le système se comporte du point de vue de l'utilisateur.

**Diagrammes de Séquences** : Les diagrammes de séquences se concentrent sur la manière dont les objets interagissent dans un système et la chronologie de ces interactions. Ils montrent comment les différentes parties du système communiquent entre elles pour accomplir une tâche spécifique. Les diagrammes de séquences sont particulièrement utiles pour représenter les processus complexes requis par cette application.

**Le diagramme de classe**: C’est un élément clé de la modélisation UML qui représente la structure statique d'un système logiciel, en mettant en évidence les classes, les attributs, les méthodes et les relations entre les classes. Il aide à visualiser comment les objets seront organisés et interconnectés dans le système. Le diagramme de classe est particulièrement utile pour les développeurs, car ils nous ont permis de planifier la conception du code et de s'assurer que toutes les classes et leurs relations sont bien définies avant de commencer la programmation.

**Méthode d'Organisation et d'Analyse (MOA)** : La MOA est une approche méthodologique qui vise à organiser et analyser de manière structurée les processus, les données et les besoins au sein d'une organisation ou d'un projet. Elle aide à améliorer l'efficacité, la compréhension et la gestion des activités en utilisant des techniques de modélisation, de documentation et d'analyse pour prendre des décisions éclairées et optimiser les opérations. Cette vision simpliste nous a aidé à y voir plus clair sur qui fait quoi.

**Modèle Conceptuel de Données (MCD)** : Le MCD est une représentation abstraite des données et de leurs relations dans une base de données. Il se concentre sur la structure logique des informations, sans se soucier des détails techniques de stockage ou d'implémentation.

**Modèle Physique de Données (MPD)** : Le MPD, quant à lui, représente la façon dont les données seront réellement stockées et organisées dans une base de données, en prenant en compte les contraintes techniques, les types de données, les index, les clés étrangères, etc. Le MPD traduit le MCD en une structure concrète qui peut être mise en œuvre dans une base de données spécifique.

**Diagrammes de cas d’utilisations (Use Case)**



**Diagramme de séquences**



**Diagramme de classe**



**MOA**





Ces différentes étapes de planification et de documentation ont contribué à assurer la cohérence et la clarté de notre projet tout au long de son développement.

## 5. Risques et Criticité



Un document "Risque et Criticité" est un outil de gestion de projet qui vise à identifier, évaluer et classer les risques potentiels associés à un projet. Voici une explication détaillée de ce qu'est ce document :

**1. Identification des risques :** Dans ce document, on recense tous les risques possibles qui pourraient affecter la réussite du projet. Ces risques peuvent être de différentes natures, comme des retards, des problèmes techniques, des contraintes budgétaires, des conflits internes, etc.

**2. Évaluation des risques :** Chaque risque est ensuite évalué en termes de probabilité de survenue (du pessimiste au plus optimiste) et d'impact potentiel sur le projet en cas de réalisation. On utilise souvent une échelle de notation pour ces deux critères.

**3. Calcul de la criticité :** Pour déterminer la criticité de chaque risque, on applique généralement une formule mathématique, qui combine les estimations pessimistes et optimistes. Cette criticité permet de hiérarchiser les risques en fonction de leur importance pour le projet.

Pour ce faire, nous avons donc appliqué cette formule mathématique simple :

FormuleRisqueEtCriticite

J'explique :

- P représente l'estimation "Pessimiste", c'est-à-dire la probabilité que le risque se matérialise de la pire manière possible.

- R signifie "Réaliste", et il s'agit de l'estimation moyenne ou d'une évaluation basée sur une perspective pragmatique.

- O correspond à l'estimation "Optimiste", décrivant la probabilité que le risque se matérialise de la meilleure manière possible.

En utilisant cette formule, en prenant en compte à la fois l'avis pessimiste et optimiste, puis en ajoutant le résultat à deux estimations réalistes, nous obtenons un indicateur qui permet d'évaluer la probabilité globale d'occurrence d'un risque.

Nous divisons ensuite ce résultat par quatre pour normaliser l'évaluation et ainsi obtenir une mesure plus représentative de la probabilité du risque.

Cette méthode permet d'obtenir une estimation de la probabilité de manière équilibrée en tenant compte des perspectives pessimistes, réalistes et optimistes. Elle nous a donc aidés à mieux évaluer et gérer les risques dans notre projet.

**4. Plan d'action:** Une fois les risques évalués et classés, on élabore des plans d'action pour chaque risque critique. Ces plans décrivent les mesures préventives et correctives à prendre en cas de réalisation du risque, afin de minimiser son impact sur le projet.

**5. Suivi et mise à jour :** Le document "Risque et Criticité" n'est pas statique. Il est régulièrement mis à jour tout au long du projet pour refléter l'évolution des risques. On peut ainsi anticiper les problèmes potentiels et prendre des mesures proactives pour les gérer.

Pour résumer, le document "Risque et Criticité" que nous avons réalisé tous ensemble est un outil de gestion de projet essentiel qui permet d'identifier, évaluer et classer les risques potentiels, tout en élaborant des plans d'action pour minimiser leur impact sur la réussite du projet.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Chef de projet

Personnellement, ce fut un moment véritablement agréable d'échange entre coéquipiers.

J'ai ainsi pu évaluer la psychologie de mes collègues de travail, ce qui m'a beaucoup aidé dans la gestion du projet.

J'ai pu voir où étaient les points bloquants pour eux, où étaient les incertitudes/doutes, et où se situaient leurs points forts, là où ils se sentaient le mieux ou ce qu'ils étaient sûrs de savoir/pouvoir faire.

C’est également à ce moment que je me suis rendu compte que, comparé à eux, j’étais très souvent le pessimiste du groupe (je préfère penser le plus expérimenté, mais bon).

C'est en partie grâce à cet échange que j'ai pu établir une stratégie à court et moyen terme pour atteindre notre objectif.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Nous avons aussi fait le **cahier des charges**, pour expliquer très simplement, qui décrit de manière détaillée les besoins, les objectifs, les spécifications et les contraintes de notre projet IOT. Il définit clairement ce qui doit être fait, mais une image vaut mille mots donc :

# Cahier des charges



# GitHub



Nous avons donc créé le GitHub du projet IoT, ajouté les fichiers `**. Git ignore**` et `**README.md**`, ainsi que configuré le fichier `**package.json**` avec certaines dépendances, comme par exemple :

"**axios": "^1.4.0**" ===> Bibliothèque HTTP pour les requêtes réseau.

"**dotenv": "^16.3.1**" ===> Gestion des variables d'environnement.

"**expo": "^48.0.9**" ===> Permet de voir via l’application « expo go » sur notre tel mobile le résultat du code.

"**path": "^0.12.7**" ===> Utilitaire pour la gestion des chemins de fichiers.

"**react": "^18.2.0**" ===> Langage React avec sa dernière version (à l’initiation du projet).

**- .gitignore** : Un fichier .gitignore est utilisé dans les projets Git pour spécifier les fichiers et répertoires qui doivent être ignorés par le système de contrôle de version Git. Cela permet d'exclure des fichiers temporaires, des fichiers générés automatiquement, des fichiers de configuration locaux, ou tout autre contenu qui ne devrait pas être suivi par Git. Il aide à garder le référentiel propre en évitant d'inclure des fichiers inutiles ou sensibles.

**- README** : Le fichier README est un document texte placé à la racine d'un projet logiciel. Il sert à fournir des informations essentielles sur le projet, telles que son but, son fonctionnement, les étapes d'installation, l'utilisation de base, et d'autres détails pertinents. Les README sont couramment utilisés pour informer les utilisateurs et les collaborateurs sur la manière de travailler avec le projet et sont souvent affichés sur la page d'accueil du référentiel sur des plateformes comme GitHub.

**- package.json** : Le fichier package.json est un fichier de configuration utilisé dans les projets Node.js. Il contient des métadonnées sur le projet, telles que son nom, sa version, ses dépendances (bibliothèques externes nécessaires au projet), des scripts personnalisés, et d'autres informations liées au projet. Ce fichier est essentiel pour la gestion des dépendances, la construction du projet, l'exécution de scripts, et facilite la distribution et la gestion des packages Node.js.

Ensuite, nous avons commencé à coder nos premières lignes dans `App.js`, qui est la pierre angulaire de notre application.

Nous importons dans un premier temps tous les langages et dépendances nécessaires pour que le code ne manque de rien. Nous importons également toutes les routes qui redirigeront l’utilisateur à l’endroit souhaité, et donc les pages principales de l’application ainsi que les pages de toutes les micro-applications.



Nous avons également créé plusieurs règles en suivant au mieux ce qui nous a été enseigné, notamment :

- DevOps : Le DevOps est une méthode de développement logiciel qui vise à améliorer la collaboration et la communication entre les équipes de développement **(Dev)** et les équipes d'exploitation **(Ops).**

Cette approche met l'accent sur l'automatisation, la mesure et le partage des informations pour atteindre un déploiement plus rapide et plus fiable des logiciels. Yannis s’occupait de toute la partie infra de la gamelle, j’ai donc tout fait pour que son travail soit cohérent avec nos fonctionnalités et que, de notre côté, nos fonctions s’adaptent le plus possible au matériel utilisé.

## CI/CD

La CI **(Intégration Continue)** est une pratique de développement où les développeurs intègrent fréquemment leur code dans un référentiel partagé. **(C 20)**

Le CD **(Livraison Continue)** va plus loin en automatisant la mise en production du code intégré et testé.

Une fois que le code a passé les tests automatisés, il est prêt à être déployé en production sans étapes manuelles supplémentaires. Pour notre projet, nous n’avons pas eu le temps de l’appliquer comme nous le voulions mais j’ai eu l’occasion d’en faire en entreprise (Voir plus loin dans le dossier de soutenance **page 60**).

- Création de branches localisées en fonction du travail à faire, selon la logique suivante :

- Actions à faire, par exemple :

**feature**: Ajout d’une nouvelle fonctionnalité

**Bugfix**: Correction d’un bug

- Travail ciblé, par exemple :

**ModificationsBoutons**

**AjoutHeader**

Cela donne un résultat qui ressemble à :

- **feature/ModificationsBoutons**

- **bugfix/header**

Avant de faire une pull-request, il fallait que le/la coéquipier/ère accepte le push de l'autre.

Nous avons donc ajouté la règle de faire une revue de code ensemble (**code review**) et de vérifier si tout était exactement comme il fallait, tant au niveau de la propreté du code, de l'efficacité énergétique (green IT), de la sécurité, des commentaires insuffisants, etc.

Et c'est seulement une fois validé que nous pouvions pull le code et continuer à avancer.

De plus, nous nous sommes efforcés de ne pas toucher aux codes de l'autre, isoler nos travaux est donc devenu une pierre angulaire de notre travail d’équipe. Si une retouche devait être faite, c’est celui ou celle qui l’a fait qui devra s’en occuper.

# Maintien de Scrum

J'ai essayé de pratiquer le Scrum en répartissant le travail de la manière la plus efficace possible au sein de l'équipe,

- en déléguant certaines tâches,

- en définissant l'objectif à atteindre,

- en maintenant les sprints agiles des deux équipes (l'une front, l'autre back).

Je tiens à préciser que je ne suis pas Scrum Master, mais plutôt, pour le moment, un Scrum But (dit autrement, un débutant). Mes compétences dans ce domaine ont été apprises lors de mes anciennes responsabilités et consolidés par les cours que nous avons eu au cours de l’année.

La distinction entre Scrum Master, Chef de projet et Chef d'équipe n'est toujours pas très évidente à mettre en pratique à ce stade, mais je tiens à remercier toute l'équipe pour son sérieux, son dévouement et sa bienveillance au quotidien, des aspects qui ont été hautement appréciés par chacun d'entre nous.

Pour la répartition, **Yannis** s'occupait de la partie "industrielle" de l'objet,

**Ryan** a configuré le Raspberry Pi pour qu'il serve d'intermédiaire entre la machine et le back-end (le back-end, pour simplifier comme le Front-end, gère le traitement des données recueillies par la gamelle ou indiquées par l'utilisateur via l'application mobile).

**Mel** a organisé, en tant que frontLead, les composants de l'application et ajouté les tâches à faire sur Notion.

Quant à moi, je me suis concentré sur les demandes des clients telles que :

* Restrictions budgétaires pour l'objet,
* Vérification des obligations linguistiques,
* Vérification des contraintes matérielles,
* Faire le lien entre les demandes client et nos capacités actuelle.

Une fois ces échanges conclus sur des accords mutuels, j’ai suivi les tâches qui m'ont été attribuées via Notion, j'ai commencé à réaliser certains composants essentiels, tels que le header et les différentes pages de connexion. Pour ce faire, nous nous sommes basés sur les maquettes que nous avions faites au préalable.

# Modéliser une maquette



J’ai commencé par le composant Header.js. Cette partie de l’application contient une cloche de notification et un bouton de connexion à l’état fermé de base.

Il y avait plusieurs détails à prendre en compte, le premier étant l'emplacement (en haut), le deuxième étant la couleur. Nous avons donc appliqué notre charte graphique.

Les couleurs ont été ajoutées en tant que variables dans le SCSS, que nous avons nommées "**PrimaryColor**" (pour le bleu) et "**SecondaryColor"** (pour l'orange).

Si le bouton du header est cliqué et/ou actif, alors il devient orange. Sinon, son état "naturel" est le bleu.

La cloche des notifications, si elle est cliquée, déroule une liste de notifications et s'ouvre lentement de la gauche vers le centre de l'écran.

Le bouton burger, lui, doit afficher un menu avec à l'intérieur des textes cliquables qui redirigent vers la page voulue.

Nous avons donc créé des composants React qui sont, par nature, réutilisables. Un seul composant de l'en-tête (**header**) sera donc utilisé sur toutes les pages de l'application. Nous avons également réutilisé le composant "button" dans toutes les pages, mais l'un s'appelle "Retour" tandis que le même bouton à un autre endroit s’appelle "Accepter". Il s'agit toujours du même bouton, mais réutilisé de manière différente. Nous avons appliqué ce principe en respectant ainsi le Green IT imposé par l'école et en automatisant le code. **(A3, C12, C14)**

## Composant Header.js



## Header.scss



## Visuels de l’application

J’ai commencé à faire un formulaire sur l’application en respectant, bien sûr, les couleurs, le logo et les liens de redirection.

Il faut bien comprendre que dès que l’on clique sur un simple bouton, au niveau du code qui va interpréter cette action, il y’a toute une logique qui s’exécute en arrière pour faire en sorte de, par exemple, changer de pages, ou alors de pouvoir enregistrer son mail.

Sur l’image suivante, j’ai fait la page de connexion. Elle se compose de deux éléments principaux, le logo en haut et le formulaire au centre.



Comme expliqué plus haut, le formulaire est un composant, les boutons sont des composants et les champs cliquable ou l’utilisateur entrera ses informations personnelles sont également des composants.

J’ai rencontré plusieurs problèmes inattendus au niveau du **CSS** et des placements des différents composants, mais avec quelques paramètres modifiés, le rendu visuel était impeccable.

En revanche, pour les liens de redirection (connexion/inscription) j’ai eu du mal à envoyer les informations reçues dans les champs au Back End. Nous avions décidé d’utiliser une API pour l’échange de données.

## API

En parallèle, Ryan et Yannis ont commencé le Back-end, et il nous fallait une liaison entre le Back et le Front. Nous avons donc décidé de créer une **API REST** qui enverra les données reçues de l’utilisateur via le Front-end de l’application vers le Back-end, qui lui traitera ou sauvegardera ces données.

Je vais tenter de vous expliquer ce qu’est, selon moi, une API. J’utiliserai donc une analogie. Il faut voir une API comme un bus qui transporte des gens. Sa mission est d’emmener des personnes d’un endroit « A » à un endroit « B »:

1. **Le Bus** : Une API est comme un bus qui transporte des données, mais il suit un ensemble spécifique de règles et de conventions pour gérer ces données.
2. **Les Passagers** : Les données transportées par une API sont comme les passagers dans le bus. Chacune d’entre elles sont identifiées par une adresse (URI), et ces données peuvent être de différents types, tels que des textes (txt), des images (JPEG), des vidéos (MPEG).
3. **L'Arrêt** : L'arrêt du bus représente une ressource sur le serveur web qui peut être consultée via une adresse URL. Par exemple, une API pourrait avoir un arrêt (ressource) appelé "/users" qui représente une liste d'utilisateurs.
4. **Le Conducteur** : L'API RESTful est comme le conducteur du bus, son comportement peut être régi par les méthodes HTTP standard, notamment GET (pour la lecture), POST (pour la création), PUT (pour la mise à jour), DELETE (pour la suppression). Chacune de ces méthodes correspond à une action spécifique que l'API peut effectuer.
5. **Le Passage des Passagers** : Les passagers (les données) montent et descendent du bus en utilisant les méthodes HTTP appropriées. Par exemple, pour obtenir la liste des utilisateurs, on peut envoyer une requête GET à l'adresse "/users".
6. **Les Actions de l'Agence de Voyage (Back-end)** : L'agence de voyage (Back-end) comprend les demandes des passagers (les requêtes HTTP) et réagit en conséquence. Par exemple, si une requête demande l'ajout d’un nouvel utilisateur, l'API effectuera l'action appropriée pour créer cet utilisateur selon le traitement du Back-end.
7. **Le Choix du Passager** : Les passagers (les clients) peuvent choisir de demander des informations spécifiques auxquelles ils sont intéressés. Par exemple, ils peuvent demander de filtrer une liste d’utilisateur (/users) en fonction de certains critères (seuls ceux qui sont majeurs peuvent passer).
8. **La Destination** : Le service qui reçoit les données est la destination finale où les données sont traitées. Le service qui aura besoin de recevoir les données de notre API peut s’il le souhaite choisir de traiter toutes les données ou seulement celles qui l'intéressent en fonction des requêtes (si les données ont plus de 6 mois, alors je ne les prends pas).

Donc chaque fois que vous entendrez parler d’une API, pensez à un bus rempli d’informations.

## Api.json



**"uuid":** Ceci définit un identifiant unique pour l'API, qui est utilisé pour l'identifier.

**"name":** Il s'agit du nom de l'API, qui est simplement défini comme "Api", nous nous sommes dit que s’il fallait préciser plus tard, nous le ferions mais au final, « Api » pour un nom d’API reste cohérent. Simple mais cohérent. Mais simple, mais …. Bref vous avez saisi l’idée.

**"endpointPrefix":** C’est un préfixe d'URL pour les points de terminaison de l'API. Pour l'instant, il est défini comme une chaîne vide, ce qui signifie que les points de terminaison utiliseront la racine de l'URL.

**"latency":** La latence, définie à 0, indique qu'il n'y a pas de délai artificiel spécifique défini pour l'API. Cela signifie que les réponses devraient être aussi rapides que possible. Si d’aventure nous voulions faire patienter la réponse, on modifierait cette donnée au-dessus de 1.

**"port":** C'est le numéro de port sur lequel l'API écoutera les requêtes HTTP. Dans ce cas, il est configuré pour écouter sur le port 3001 en Localhost, ce qui signifie qu’il est en local sur nos postes de travail mais pas en ligne.

**"hostname":** Le nom d'hôte auquel l'API sera liée. S'il est défini comme une chaîne vide, cela signifie qu'il écoutera sur toutes les interfaces disponibles de la machine.

**"routes" :** Ceci est une liste de routes ou de points de terminaison pour l'API. Chaque élément de la liste décrit une route spécifique avec ses propriétés. Il y a plusieurs routes configurées dedans mais par soucis de place (une photo de 8 pages aurait été un poil excessif je pense) je vais résumer la première d’entre elles.

**"type":** Le type de la route, qui est HTTP dans ce cas. Les programmes utilisent des protocoles pour discuter entre eux. Il existe par exemple HTTP (World Wide Web), FTP (transfert de fichiers), SMTP (messagerie), SSH (connexion à distance sécurisée) et pleins d’autres.

**"method":** La méthode HTTP associée à cette route, qui est "POST" dans ce cas, cela veut dire que cette route va s’occuper des **ajouts** en base de données.

**"endpoint":** L'URL ou le point de terminaison de cette route, qui est "auth/login", donc en suivant les informations vues avant, on va **ajouter** grâce à cette route un **login** en base de données.

**"headers" :** Une liste de paires **key = value** (clé = valeur) pour les en-têtes de la réponse.

**"key":** La clé de l'en-tête, qui est "Content-Type".

**"value":** La valeur de l'en-tête, qui est "application/json", indiquant que le contenu de la réponse est au format JSON.

Le **JSON (JavaScript Object Notation)** est un format de données léger utilisé pour structurer et échanger des données entre applications. Il est facile à lire pour les humains et à analyser pour les machines, ce qui en fait un choix populaire pour la communication de données sur le web.

Donc notre **API** utilise le format **JSON**.

Nous avons avancé ainsi pendant plusieurs jours, en nous adaptant aux différentes nécessités du moment. **(A3 C13 C15 C17 C18 C21)**

Les réunions devenaient de plus en plus précises et habituelles, le dialogue entre nous s'est installé de manière naturelle, et la progression était très satisfaisante.

Ryan était responsable d’un suivi pour conserver nos évolutions :



Sans jamais oublier de mettre à jour notre Notion.



Il nous fallait donc simuler notre API puis la tester. Mais un jour, un mardi matin (pour être précis), nos objectifs de la semaine furent chamboulés.

## Imprévu

L'école nous a envoyé un mail pour nous demander une .... Oui oui, une présentation !

Et là, je ne vous cache pas que je me suis retenu au maximum, mais la phrase

***"HAHA Je vous l'avais bien dit !!"*** a été sortie.

Il nous a donc fallu reporter l’avance de notre API pour nous consacrer à plein temps à cette présentation car il faut le savoir, la date limite pour la présentation était de seulement deux jours (présentation le vendredi même).

Et quel bonheur d'avoir préparé, noté et illustré cette présentation Canva tout au long du projet, à chaque nouvelle image, chaque nouvelle étape, tout était mis de côté, pour le cas où.

Ce qui s'est avéré être au final non pas du temps perdu mais bel et bien une étape du projet réglée avant même que le client (l’école) y est pensé.

Heureusement pour nous, environ 80% du travail avait déjà été réalisé, il nous restait principalement à définir le contenu verbal, à décider qui prononcerait quel discours, et à ajouter quelques slides supplémentaires pour plus de détails.

Exemple de slides ajoutées:

## Etude de marché



Pour compléter notre présentation, nous avons donc ajouté une étude de marché, la stratégie marketing de notre marque ***« Purrpaws »*** et quelques éléments supplémentaires.

Nous avons tous participé cette fois-ci et avons comblé certaines lacunes, clarifié des incompréhensions concernant l'ordre dans lequel nous avons réalisé les différentes tâches, ajouté des éléments tels que le MCD et le MPD, introduit un début et une fin, et inclus des photos de l'équipe retouchées par une IA, une Intelligence Artificielle (dans notre cas, elle a été utilisée pour légèrement modifier nos photos avec un style unique, donnant ainsi un aspect original à nos photos de présentation).

## IA

Je tiens à préciser que le terme IA n’est en fait en aucun cas de l’Intelligence. L’intelligence est, selon moi et dans le sens « humain » de la chose, la faculté d’adaptation. Certes l’IA s’améliore par la répétition (beaucoup, BEAUCOUP de répétitions) mais n’est en aucun cas de l’intelligence a proprement parlé.

L’IA ne comprend pas, elle doit être orientée vers ce qui est une bonne ou une mauvaise réponse (selon des critères définis par les responsables) aux demandes des utilisateurs.

Et enfin, le Canva fut prêt !

# Présentation



Pour ma part, une présentation orale devant des juges est une nouvelle expérience. Il nous a fallu expliquer de manière claire et précise, tout en restant simple, le commencement, le déroulement et l'objectif final du projet de la Gamelle connectée.

Ce qui, en fin de compte, s'est avéré bien plus complexe que ce à quoi je m'attendais. Écrire une expérience et la raconter sont totalement différents.

Un oubli ou une erreur dans un écrit se résout facilement avec une relecture et une réécriture. Perdre le fil ou oublier, même vaguement, un sujet à l'oral est bien plus complexe à rattraper.

Nous nous sommes réparti le plus équitablement possible les différentes slides comme ceci :

1 - Démarrage du PowerPoint et explication (Marc)

2 - Présentation (Tout le monde)

3 - Introduction (Mel)

4 - Présentation des catégories (Mel)

5 - Slide Analyse du besoin

6 - Cahier des charges (Yannis)

7 - Diagrammes UML (Yannis)

8 - Plan de coupe (Marc)

9 - Slide Gestion de Projet

10 - Gestion risque et criticité (Marc)

11 - Notion (Ryan)

12 - Slide conceptualisation

13 - MCD (Ryan)

14 - Wireframes (Mel)

15 - Maquettes (Mel)

16 - Slide solution technique

17 - Ingénierie (Yannis)

18 - Ingénierie (Yannis)

19 - Langages (Marc)

20 - Schéma Python <-> Symfony <-> React (Ryan)

21 - Marketing (Yannis)

22 - Marketing ((everyone)

23 - Questions (everyone)

Nous avons donc fait notre présentation. J’ai pour ma part un peu oublié certains détails mais dans l’ensemble, tout s’est bien passé.

Notre jury était composé d’un débutant en développement et d’un expert avec un certain bagage technique et de gestion. Les questions posées par eux avaient donc un écart significatif en terme de cible.

L’un nous demandait quels ont été les problèmes rencontrés et comment nous les avons fixés. Comment nous étions arrivés à telle ou telle conclusion ou qu’est ce qui nous a amené à choisir telle ou telle technologie ou langage informatique.

Tandis que l’autre chercher plus à comprendre comment faire pour utiliser l’application et faire ce dont il avait besoin pour nourrir le chat, se connecter, ou modifier ses choix. Pourquoi nous avons choisi la couleur bleue et orange, ou encore s’il avait besoin d’internet pour que la gamelle fonctionne.

Nous lui avons donc expliqué ce qu’est l’**UX**.

## UX

Ce que l’on appelle **l’Expérience Utilisateur** (**User Experiment**) consiste à rendre le plus agréable possible la sensation qu’aura l’utilisateur en consommant notre application. Au plus l’utilisateur prendra de plaisir à rester sur notre application ou notre site, le plus de chance nous aurons de pouvoir vendre notre produit.

Après plusieurs minutes, nous avons étanché leur soif de questions et fini notre présentation sur une note détendue et souriante.

Notre objectif étant atteint, nous avons pu reprendre la suite de notre périple.

Au cours des jours qui suivirent l’annonce de notre présentation et après concertation avec l’équipe, il fallait impérativement que je finisse un travail avant de reprendre l’API : le **bouton des notifications**. **(C26)**

## NotificationButton.js



Nous nous sommes rendus compte qu’il fallait que l’onglet des notifications se ferme si on clique à côté dudit onglet, (**handleClickOutside**), et en mettant un léger délai à l’ouverture/fermeture, cela rend le tout plus interactif, l’**UX** est beaucoup plus agréable donc, idée conservée (**UX** signifie Expérience utilisateur).

Puis j’ai attaqué les pages de connexions, le burger bouton, le CSS qui gère les composants React.

Mais qui dit « formulaire » dit « champ à remplir par l’utilisateur » et donc, données à sauvegarder (notamment le mail et le mot de passe).

Il nous a donc fallu simuler un échange « requête/réponse » entre le front, l’api et le back, c’est là qu’arrive Postman et Mockoon.

## Mockoon

**Mockoon** est un outil qui vous permet de créer des serveurs **simulés** (mocks) pour simuler le comportement d'une API. Il est principalement utilisé pour tester des clients d'API lorsque le serveur réel n'est pas encore disponible. On va l’utiliser pour optimiser la création de notre API et, au passage, apprendre cet outil.

Principaux avantages de Mockoon :

**Création de mocks d'API** : Mockoon vous permet de créer des points de terminaison (endpoints) simulés avec des réponses préconfigurées. Si vous voulez paramétrer votre API de différentes manières, vous pouvez le tester grâce à Mockoon.

**Simulation de scénarios** : Vous pouvez simuler différents scénarios en configurant des routes et des réponses différentes pour chaque route. Cela vous permet de tester comment votre application va réagir aux diverses réponses d'API.



Ici, nous envoyons les données de base de création d’un nouveau compte utilisateur.

J’ai donc commencé par :

* Un « userId » qui sera unique pour chaque utilisateur
* Un Nom
* Un Prénom
* Le champ « friends » sera le champ de l’animal (ou des animaux) et un identifiant a l’intérieur qui sera lui aussi unique.

Pour information, les « faker » qui définissent les valeurs dans les champs génèrent aléatoirement des noms, des dates, des chiffres, cet outil est très pratique pour remplir des champs vides.

Ces « faker » sont donc les premières données que je voulais envoyer pour voir ce que ça donnerait après réception dans Postman.

## Postman

**Postman** est essentiellement la même chose que Mockoon, mais de l’autre côté de la barrière. Comme écrit plus haut, toute réponse demande requête. Mockoon simule l’envoie de données API, Postman simule la réception de ces données et le traitement qui devra s’adapter à ces requêtes.

C’est également une application de développement API qui permet aux développeurs de tester, documenter et automatiser les interactions avec des API, facilitant ainsi le développement et le débogage des services web.



L’adresse écoutée est donc ici [**https://127.0.0.1;800/animals**](https://127.0.0.1;800/animals). Mockoon devra donc utiliser cette adresse pour envoyer la donnée, et Postman devra recevoir le JSON de l’API et afficher les données grâce au GET.

## .env

Un fichier `.env` (environnement) est un fichier de configuration utilisé pour stocker des variables d'environnement dans une application, y compris dans le contexte d'une API .

Ces variables d'environnement sont essentiellement des paramètres ou des valeurs de configuration que l’application peut utiliser pour fonctionner correctement. Ici, nous l’utilisons par exemple pour définir les endPoints (adresses) qui seront utilisées/écoutées par l’API. **(C23)**



Le .env de notre API permet de remplir quelques points importants : (sécurité référentiel)

- **Stockage des configurations sensibles** : Les fichiers `.env` sont généralement utilisés pour stocker des informations sensibles ou des configurations spécifiques à l'environnement, telles que des clés d'API, des identifiants de base de données, des jetons d'accès ou des URL de services tiers. Plutôt que d'inclure ces informations directement dans le code source de notre application, elles sont stockées de manière sécurisée dans un fichier `.env`.

- **Séparation des configurations et du code** : L'utilisation d'un fichier `.env` permet de séparer les configurations de votre code source. Donc on peut changer les paramètres de configuration sans avoir à modifier directement le code de l'API, ce qui la rend bien plus facile à gérer ou a moduler au fil du temps et des différentes mises à jour possible.

- **Protection des données sensibles** : En utilisant un fichier `.env`, on réduit le risque d'exposer des informations sensibles, car ce fichier est généralement ajouté à la liste des fichiers ignorés (exemple : « .gitignore » en bas de la page). En gros le fichier `.env` ne sera pas partagé avec le code source lorsque vous utilisez des systèmes de contrôle de version comme Git.

- **Lecture des variables d'environnement** : Dans notre code source, on peut lire les variables d'environnement à partir du fichier `.env` en utilisant une bibliothèque ou un package qui permet de charger ces variables. Elles peuvent ensuite être utilisées pour configurer l'API, y compris pour se connecter à des bases de données ou gérer l'authentification.

## .gitignore

Le fichier .gitignore est utilisé dans les systèmes de contrôle de version Git pour spécifier des fichiers et des répertoires qui ne doivent pas être suivis ni inclus dans le dépôt Git. Cela permet d'ignorer les fichiers temporaires, les fichiers de configuration locaux et d'autres fichiers non pertinents pour le développement.



L'avantage ?

- **Protection des données sensibles** : En spécifiant les fichiers et répertoires sensibles dans le fichier .gitignore, on les empêchent de se retrouver dans le dépôt Git.

Donc les fichiers contenant des clés d'API, des mots de passe ou d'autres données confidentielles ne seront pas visibles en dehors du code voulu (exemple : « .env »). Et donc on réduit le risque de divulgation accidentelle de ces données.

- **Exclusion des fichiers générés** : Les fichiers générés automatiquement (par exemple, les fichiers binaires, les fichiers de build ou les journaux d'erreurs) peuvent être exclus du suivi Git en les spécifiant dans .gitignore. Cela évite que des fichiers volumineux ou inutiles soient ajoutés au dépôt (exemple : node-modules, plusieurs milliers de fichiers inutiles à stocker dans Git).

- **Prévention d'inclusions malveillantes** : En ayant un fichier .gitignore bien configuré, on réduit le risque d'inclusion accidentelle de fichiers malveillants ou non autorisés dans le dépôt. Donc on peut se protéger contre des attaques potentielles liées à l'injection de code ou à l'ajout de fichiers malveillants.

## SEO

Avec les cours que nous avons suivi, il fallait nous occuper de notre **référencement** (C’est la visibilité sur internet et dans les recherches Google par exemple), nous avons donc chercher des noms percutants et ciblés pour nous donner de la visibilité. Mais d’abord, petite explication de ce qu’est le **SEO** :

Le **SEO**, ou **Search** **Engine** **Optimization** en anglais, est l'ensemble des pratiques visant à améliorer la visibilité et le classement d'un site web dans les résultats des moteurs de recherche. Nous avons choisi de répertorier un ensemble de mots clés que pourrait rechercher un client potentiel. Le but étant d’optimiser le contenu et la structure du site pour correspondre aux critères et aux algorithmes de recherche des moteurs. On aurait pu choisir des mots (pardonnez-moi l’expression) « **putaclick** », mais nous sommes restés sur des mots pertinents.

Je me permets une **parenthèse** ici car, les algorithmes de recherche sont les moyens les plus efficace d’avoir de la visibilité sur internet, le problème me demanderez-vous ? Et bien, si un jour, par exemple, un créateur créé le meilleur site internet ou la meilleure vidéo YouTube, mais que sont SEO n’est pas bon, alors ils ne seront jamais vu. Tout cela parce que les mots clés ou le titre de ceux-ci ne seront pas pertinents ou « putaclick ». Je trouve vraiment dommage que la qualité du contenu ne l’emporte pas sur le titre ou les mots clés associés à ce site ou vidéo mais bon … (Fin de la **parenthèse**)

Pour notre site internet, nous avons, sur Notion, mis en place notre stratégie SEO, voici les suggestions de l’équipe :



Et avec ces suggestions, voici le début de notre site :



Il y’a plusieurs manières d’optimiser le référencement, mais nous avons choisi celui-ci. Il existe aussi le choix des balises « **meta** » mais apparemment, elles sont moins pertinentes aujourd’hui, les moteurs de recherche se concentrent davantage sur le contenu et les balises donc, dans l’image « **gamelleV1** » et dans le « **alt** » nous irons.

## **Accessibilité web et son importance pour notre site**

**L'accessibilité web** se réfère à la pratique de concevoir et de développer des sites web de manière à ce qu'ils soient utilisables et accessibles par tous, y compris les personnes ayant des handicaps ou des limitations physiques, sensorielles ou cognitives.

Cela permet à un plus grand nombre de personnes de naviguer sur notre site web et d'y accéder facilement. Cela peut inclure des personnes atteintes de déficiences visuelles, auditives ou motrices, ainsi que des personnes ayant des troubles tels que la dyslexie. Le tout étant règlementé par le **RGAA**.

Le **RGAA** ou **Référentiel général d'amélioration de l’accessibilité** énonce des directives techniques et des recommandations pour garantir que les contenus numériques sont utilisables par un large public, y compris les personnes handicapées de différentes manières. Le **RGAA** a été élaboré en réponse à la loi pour l'égalité des droits et des chances, la participation et la citoyenneté des personnes handicapées de 2005.

Je vous montre une partie de notre **Notion** où nous avons déterminer comment nous allions gérer l’accessibilité à notre site :

**Méthodes pour rendre notre site web accessible pour les personnes ayant des handicaps :**

1. Visuels
   * *Utiliser des couleurs de texte et de fond suffisamment contrastées pour que le texte soit facile à lire.*
   * *Utiliser des tailles de police suffisamment grandes pour que le texte soit facile à lire, même pour les personnes ayant une déficience visuelle.*
   * *Utiliser des descriptions alternatives (alt-text) pour toutes les images afin que les personnes malvoyantes puissent comprendre leur contenu.*
2. Auditifs
   * *Utiliser des sous-titres pour les vidéos et les enregistrements audio.*
   * *Fournir des transcriptions textuelles pour les contenus audio et vidéo.*
   * *Éviter d'utiliser des contenus audio et vidéo qui dépendent uniquement du son pour transmettre l'information.*
3. Moteurs
   * *S’assurer que le site web peut être navigué à l'aide d'un clavier uniquement, sans nécessiter l'utilisation d'une souris ou d'un autre périphérique de pointage.*
   * *Utiliser des boutons et des liens suffisamment grands pour être facilement cliquables avec une souris ou un autre périphérique de pointage.*
   * *Éviter d'utiliser des contenus qui clignotent rapidement, car cela peut être dérangeant pour les personnes atteintes d'épilepsie ou d'autres troubles neurologiques.*
4. Cognitifs
   * *Utiliser un langage simple et clair pour que les informations soient faciles à comprendre.*
   * *Utiliser une structure de site web claire et facile à naviguer pour aider les utilisateurs à trouver rapidement l'information dont ils ont besoin.*
   * *Éviter d'utiliser des termes techniques ou des acronymes qui peuvent être difficiles à comprendre pour les personnes ayant des handicaps cognitifs.*

Et c’est à ce moment qu’arrive : **mes débuts de stage en entreprise !**

# L’entreprise

logo_acta

**ARC Europe France**, filiale du groupe ARC EUROPE est située à Limonest, à côté de Lyon (69). C’est un centre d’appels spécialisé dans le déploiement de prestations d’assistance automobile pour des clients du monde de l’automobile, du leasing et de l'assurance.

Ils interviennent pour des bénéficiaires en besoin d'assistance en bord de route en France et en Europe. Afin d’assurer un service de qualité et de proximité, ARC Europe s’est doté d’un réseau performant de prestataires (dépanneurs, loueurs de véhicules, taxis…) permettant une plus grande réactivité.

L’entreprise compte 450 salariés permanents environs.

**Cette expérience est une véritable aventure, où chaque élément appris renferme en lui-même des sujets qui pourraient être discutés bien plus longuement que le temps alloué à la présentation orale.**

Ainsi, le mot d'ordre sera la concision ! Mais sans précision, ça ne sera pas forcément facile à cerner donc, je vous explique le fonctionnement de la société Arc.

Imaginez que vous êtes en train de conduire votre voiture pour aller au travail, mais, pour une raison inconnue, votre voiture s'arrête. Vous êtes en panne au bord de la route ! Après avoir retrouvé votre calme (et cessé de proférer des jurons à la chance ou plutôt à l'infortune qui est la vôtre), vous évaluez la situation et déterminez ce que vous devez faire en premier. La première étape à suivre est de contacter votre assurance ! Vous composez le numéro et dites : « Allo, j'ai besoin d'aide, je suis en panne. » Vous êtes donc le **bénéficiaire**.

De l'autre côté de la ligne, la personne qui vous répond est située dans notre bureau à Arc, et elle s'appelle un **chargé** d'assistance. Il/elle s'assure que vous allez bien, que vous êtes en sécurité, puis recueille des informations vous concernant, comme votre nom ou votre marque de voiture. Le chargé vous demande l’autorisation de vous **géolocaliser** afin de dépêcher une dépanneuse exactement à votre emplacement. Si vous avez des questions, il ou elle est là pour vous fournir toutes les réponses nécessaires. La dépanneuse arrive, prend en charge votre voiture, et l'achemine vers un garage capable de réaliser les réparations nécessaires. Ces garages sont désignés comme **réparateurs agréés (RA)**. Le RA effectue une évaluation de votre véhicule, effectue les réparations requises, puis restitue votre voiture réparée en moins d’une semaine.

Enfin, la facture est émise par le service comptable, puis envoyée au bénéficiaire.

Ma reconversion professionnelle devait avoir du sens pour moi, et aider des personnes qui en ont besoin m’a tout de suite intéressé. Je me suis donc lancé dans l’aventure Arc Europe France.

## Déroulé du stage

Pendant les deux premières semaines de mon expérience, nous avons consacré du temps à la mise en place de mon environnement de travail et j'ai reçu une présentation approfondie du fonctionnement de l'entreprise. Cette période a également inclus des visites des locaux, des observations des appels avec les bénéficiaires pour comprendre le processus de traitement des appels, ainsi qu'une exploration détaillée des détails de mon futur espace de travail, tant sur le plan physique qu'environnemental.

Ensuite, nous avons plongé dans la résolution de problèmes, suivi d'un examen complet de tous les éléments interactifs de l'application **Cactus**. J'ai relevé plusieurs problèmes à ce stade :

- Certains formulaires étaient en réalité présentés sous forme de tableaux (utilisant les balises tr/td), dans lesquels du texte était inséré directement, sans recours à des variables ou des composants réutilisables.

- Certains éléments censés être des boutons étaient en fait des images (gif/jpg) cliquables agissant comme des liens de redirection.

- Une portion raisonnable du code datait de plus de 20 ans, ce que l'on appelle souvent du code "**legacy**".

Cette phase m'a permis de saisir une réalité manifeste mais parfois négligée : alors que nos formations nous enseignent à construire des solutions à partir de zéro, le monde professionnel peut nécessiter des compromis dans le code, que ce soit pour des raisons de durabilité environnementale (évoquée sous le terme "**Green IT**") ou pour faciliter les révisions du code par des pairs ("**code review**").

En collaboration avec Damien (collègue développeur), nous avons entrepris la transformation des 72 images agissant comme des boutons (une pratique discutable aujourd’hui mais rependue à l’époque où a été créer le code) en véritables boutons textuels (avec une balise HTML <img>). Cette transformation permettait une meilleure gestion des boutons, la suppression des images de la base de données et la création d'un style CSS approprié. **(Green It)**

Ensuite, nous avons exécuté une révision du code (code review), employant des pratiques similaires à celles utilisées avec Git, suivie de la création d'une "release note" détaillant nos modifications pour la version 2.26.

Nous avons également rédigé une **documentation** dans laquelle nous avons indiqué les changements effectués, en y associant une date. **(C24)**

Nous avons ensuite identifié des problèmes mineurs, que nous avons résolus rapidement, avant de les soumettre à un processus de validation (tests) en environnement de recette.

Cette séquence de développement suivait le parcours : développement (**dev**) => recette(**rec**) => production (**prod**). Ce qui veut dire que nous développons en DEV, nous testons le code en REC, et poussons le résultat validé par tous en PROD.

De plus, nous avons **automatisé** des requêtes SQL, comme l'exemple suivant :

INSERT INTO ATP\_Event VALUES (353233,'2015-01-5','14:28:43',NULL,'Renault',2,NULL,2,NULL,NULL,NULL);

Après mes vacances, Cédric m'a expliqué les "**nugets**", qui sont des bibliothèques de dépendances utilisées lors de la compilation. Suite à cela, notre mission a consisté à chiffrer le niveau de difficulté pour résoudre différents problèmes, en d'autres termes, à estimer la charge de travail associée à chaque ticket.

## Cartographie Applicative Exacte

Je ne vais pas vous mentir, il m’a fallu quelques semaines pour comprendre ou je mettais les pieds, voyez par vous-même :



Sachez que je ne touche pas à la totalité de ces applications, mais juste à quelques une d’entre elles (heureusement pour moi).

En tant que stagiaire, je me suis concentré sur les applications principales **: cACTus, Bridge, Resimage, Becall, Espace RA**. J’ai eu l'occasion de suivre les mises en recette du projet "**Bridge/cACTus**". Nous avons revu la procédure, car celle-ci était incomplète, puis nous avons soumis le projet "Bridge" à la phase de recette.

Pour ce travail, nous avons consacré un total de 6 heures avec Damien, et une assistance de 4 heures de la part d'Aurélien.

## Cartographie Applicative localisée



Pour résumer l’image ci-dessus, **cACTus** est l’application principale de la société. **Atlas** est la version 2022 Polonaise (AEF, AEP pour Arc Europe France / Pologne). **Bridge** fait le pont entre **Atlas** et **cACTus** pour le transfert de données, **Calypso** est la DataBase, le but de l’API est d’envoyer toutes les données de Ford (un de nos clients) vers la DataBase Polonaise. **VGF** (Volkswagen) et **JLR** (Jaguar, Land Rover) suivront une fois que tout sera stable.

On m’a confié le développement de l’API destinée à faciliter la migration entre les deux applications, qui sera précisément entre **l'Espace RA Ford** et **Calypso**. J'ai réussi à mettre en place une version fonctionnelle de cette API, avec des tests réalisés à l'aide de **Postman, Swagger, TFS et MVS**. Son nom ? **Promethee**.

Cependant, cette version n'a pas satisfait les exigences du client et mon responsable a décidé de la refondre entièrement.

Pour cette nouvelle version de l'API, nous avons repris le processus depuis le début, en utilisant MVS comme base. Nous avons créé le projet via TFS et l'avons ensuite cloné dans MVS.

Nous avons ensuite ajouté les différentes solutions nécessaires (Promethee/Promethee.Tests/Promethee.Business/Promethee.Business.Tests), ainsi qu'un répertoire dédié à la base de données (BDD) et un autre pour les librarys (adresse de jonction avec la BDD).

Les dépendances nécessaires ont également été intégrées à l'aide de nugets. Un **NuGet** dans Microsoft Visual Studio est un module de code préconstruit qui ajoute des fonctionnalités ou des bibliothèques externes à un projet, simplifiant ainsi le développement en réutilisant du code existant.

Exemple :

**Swagger** : Dans le domaine du développement web, est un outil essentiel qui permet de concevoir, documenter et tester des APIs de manière efficiente. Il propose une interface interactive pour interagir avec les points d'accès (endpoints) et génère automatiquement une documentation claire et compréhensible pour les développeurs.

**Newtonsoft.Json** : Souvent simplement désignée sous le nom de Json.NET, c’est une bibliothèque largement reconnue pour le traitement des données au format JSON (JavaScript Object Notation) dans le domaine du développement logiciel. Cette bibliothèque facilite la **sérialisation** (conversion d'objets en JSON) ainsi que la **désérialisation** (conversion de JSON en objets) grâce à une approche souple et pratique.

Une fois cette étape accomplie, nous avons construit les bases de notre future API. Dans le but de garantir une compatibilité optimale pour la migration (serveur de 2013, code de 2023), nous avons intégré **LLBLGen Pro**.

**LLBLGen Pro** se révèle être un outil puissant et flexible de mappage objet-relationnel (ORM) qui simplifie l'accès aux bases de données. En permettant aux développeurs de manipuler les données au moyen d'objets et de requêtes orientées objet plutôt que d'interactions directes avec les bases de données relationnelles, cet outil améliore grandement l'efficacité et la maintenabilité du code. Nous avons choisi la version 5.10.1, car elle est la seule à pouvoir prendre en charge **.NET7** en cas de migration vers un serveur plus récent, envisageant ainsi la pérennité de notre API. (Cette version a été obtenue via le chemin [\\srv-itt\ITT\_Sources\Outils\BDD\LLBLGen\Versions](file:///\\srv-itt\ITT_Sources\Outils\BDD\LLBLGen\Versions)).

Dans le cadre de la migration de **cACTus** vers **Atlas**, il a fallu faire une fonction qui va copier les données de l’atelier RA (garage) vers Atlas, et une autre qui va vérifier si l’atelier RA a déjà été migré ou non.

Arrive donc les premiers tests unitaires avec **Telerik** **JustMock** (c’est un outil de mocking pour les tests unitaires en développement logiciel, permettant de simuler le comportement des dépendances). Le but des tests unitaires est simple, on vérifie si la fonction fait ce qu’elle est censée faire. Telerik JustMock est un outil qui facilite la création de ces tests unitaires en permettant de "simuler" ou de "se moquer" de certaines parties du code. Pour ça, on va lui envoyer des données correctes, et de fausses données pour voir comment va réagir la fonction. Si le test est concluant, alors la fonction fonctionne bien, indépendamment du reste du code et peut donc partir en recette /prod. Sinon, cela veut dire que la fonction est incomplète/fausse et à revoir. **(C19)**

J’ai appris l'approche "**Arrange**, **Act**, **Assert**" (**AAA**). C’est une structure couramment utilisée dans la rédaction de tests unitaires pour maintenir une organisation claire et cohérente. J’explique :

1. **Arrange :** C'est la première étape où on configure l'environnement de test. Il faut mettre en place tous les prérequis et les conditions initiales nécessaires pour le test. Cela peut inclure la création d'objets, l'initialisation de variables, la configuration de l'état initial, ou la préparation de données de test.
2. **Act :** C'est la deuxième étape où on effectue l'action ou l'appel de la méthode que l’on souhaite tester. C'est le moment où on exécute le code à évaluer.
3. **Assert :** C'est la dernière étape où il faut vérifier les résultats de l'action entrepris à l'étape précédente. On compare les résultats obtenus avec les résultats attendus pour s’assurer que le comportement du code testé est correct. Si les résultats correspondent aux attentes, le test est considéré comme réussi.

**Sur l’image qui suit, nous avons testé la fonction MigrateWorkshopGroupInBDD qui va chercher si le regroupement de garage (ce sont des groupes de garages, je prends l’exemple de Ford, s’il y’a 5 garages sur Lyon, le groupe contiendra 5 garages Ford) à bien envoyé les données à Atlas.** **(3 réussites ici)**.

Dans les deux premiers tests, on envoie de bonnes réponses, dans le troisième, des fausses.



# Compléments

Dans un souci de vous montrer au maximum le fait que j’ai vu l’ensemble du référentiel DFS, je vais vous expliquer ce que j’ai fait sur différentes cellules du référentiel de l’école **(C XX)** sans qu’il y est forcément de suite logique, mais juste confirmer le fait que j’ai vu ou fait ces activités.

## Back API node express



Dans le cadre de notre apprentissage, nous avons vu les API et le backend qui les gère, il nous a donc fallu faire, grâce à **node.js** et **express**, faire les redirections, l’enregistrement des données et la logique qui gèrera les données. Petite explication de ce qu’est **Node.js** et **Express** : **(C16)**

**Node.js** : C'est un environnement d'exécution JavaScript côté serveur (Back) qui permet d'exécuter du code JavaScript en dehors d'un navigateur web (Chrome, Firefox …). Node.js est largement utilisé pour créer des applications serveur rapides et évolutives.

**Express** : C'est un **Framework** web pour Node.js. Il simplifie la création d'applications web en fournissant des fonctionnalités prêtes à l'emploi, comme la gestion des routes et des requêtes HTTP. Express est populaire pour le développement du backend d'API en raison de sa simplicité et de sa flexibilité.

**Framework** : Il faut voir les Framework comme des infrastructures prêtes à l'emploi pour le développement logiciel. Ils sont conçus pour simplifier et accélérer le processus de création d'applications en fournissant une structure et des outils standardisés. J'ai moi-même beaucoup utilisé **Bootstrap**, qui est très pratique pour le **responsive** design (l'adaptation de l'apparence en fonction de la taille de l'écran : le rendu visuel d’une application ne sera pas obligatoirement le même si on le regarde sur un téléphone portable ou sur un écran d'ordinateur).

Pour expliquer rapidement ce que fait ce code, en premier j’importe les modules nécessaires, ensuite je crée un objet Express en utilisant **express()**, et un routeur est créé en utilisant **express.Router()** pour gérer les différentes routes de l'API. Enfin, le code définit des routes pour différentes actions liées aux utilisateurs. J’ai suivi les conventions REST :

* Pour la route /users en méthode GET, il renvoie tous les utilisateurs stockés sous forme de JSON.
* Pour la route /users en méthode POST, il ajoute un nouvel utilisateur en extrayant les données du corps de la requête.
* Pour la route /user/:id en méthode GET, il récupère un utilisateur spécifique en fonction de l'ID fourni dans les paramètres de la requête.
* Pour la route /users/:id en méthode DELETE, il supprime un utilisateur spécifique en fonction de l'ID fourni dans les paramètres de la requête. Si l'utilisateur n'est pas trouvé, il renvoie un statut 404 avec un message "User not found".

On renvoie sous format JSON des réponses **res.status()** qui définissent le statut http approprié. Les codes HTTP 200 indiquent le succès, les codes 300 sont liés aux redirections, les codes 400 sont des erreurs attribuées au client (Front), et les codes 500 sont des erreurs attribuées au serveur (Back).

Vous devez certainement connaitre le plus connu d’entre eux :



Dans le cadre de notre formation, nous avons abordé un point extrêmement important concernant le droit et les lois qui entourent notre métier. Imaginons que l'on me demande de concevoir un algorithme destiné à piloter automatiquement des voitures électriques, et qu'un jour cette même voiture soit impliquée dans un accident. Qui en serait responsable ? La société qui a fabriqué la voiture ? Le passager ? Le développeur qui a écrit le code ? Toutes ces questions suscitent des débats complexes. Par conséquent, nous avons eu la chance de rencontrer une avocate au cours qui a résumé le contexte. **(C25)**

## RGPD

**RGPD :** Le Règlement Général sur la Protection des Données est une loi européenne qui concerne la manière dont les données personnelles sont traitées. En gros, il garantit que les entreprises doivent prendre des mesures pour protéger les informations personnelles de leurs utilisateurs et que donc elles doivent demander la permission de collecter ces données, les protéger de manière sécurisée et permettre aux gens de les supprimer s'ils le souhaitent. C'est la base de la protection de la vie privée des gens en ligne.

Dans notre application **Purrpaws**, nous avons donc ajouté un bouton d'acceptation des **CGU** (Conditions Générales d'Utilisation) qui nous autorise à agir conformément à ce qui est stipulé dans notre règlement. Tant que nous ne transgressons pas la loi, nous avons la possibilité (techniquement) de gérer les données enregistrées dans notre base de données comme bon nous semble. Il faut savoir que ces données ont une valeur, et que nous pouvons parfaitement les partager avec des organismes qui les utiliseront pour personnaliser leurs publicités de manière plus ciblée (chose très fréquente).

En partant de ce principe, nous avons décidé de ne pas être trop excessif et de nous catonner aux données qui nous intéresse à savoir :

* Le nom / prénom de l’utilisateur,
* Les coordonnées (mail, tel),
* Les données de l’animal (nom, poids, sexe, castration, photo, âge),
* Le lieu de résidence,
* Le type de croquette que va recevoir la machine

Sachant que tout ceci a une valeur, nous avons pris la décision de tout vendre. Après tout, pourquoi pas ?

## Plateforme d’hébergement

Toujours dans notre formation, on nous a montré **AWS** : **(C 22)**

**Amazon Web Services** est une plateforme de cloud computing. Elle permet de louer des ressources informatiques à la demande, comme des serveurs ou du stockage, via internet. AWS offre un large éventail de services pour héberger des applications et des données. Les utilisateurs paient en fonction de leur utilisation pour de nombreux besoins informatiques.

Je pensais que c’était vraiment génial comme plateforme, jusqu’à ce que je vois le système économique. Le meilleur avantage à mes yeux c’est la sécurité, ça évolue en permanence et c’est très complet. Par contre l’inconvenant principal, l’argent ! Si on ne fait pas attention à l’optimisation des couts de manière permanente, il peut y avoir des factures inattendues (vous remarquerez le pluriel). Si bien que quand on travaille avec eux, avant toute chose, il faut regarder combien ça va couter à la fin, et ensuite seulement de quoi a-t-on réellement besoin. Donc non, hors de question d’utiliser cette plateforme.

Pour rebondir sur ce sujet, mon entreprise **AEF** détient son propre data center. Le budget alloué cette année dépasse les six chiffres. Leur ancienne base de données était à 99,9 % de charge cette année en septembre. Il a fallu intervenir en supprimant des données stockées. Le choix s’est porté sur des données vieilles de plusieurs années et devenues obsolètes avec le temps, ce qui a permis de gagner quelques giga-octets. Précisons rapidement quand même ce qu’est un octet :

Un **bit** est l’unité de base de l’informatique, il peut avoir 2 valeurs : 0 ou 1. Il constitue le fondement du langage binaire.

Un **octet** est composé de 8 bits. C’est grâce à cette unité que nous mesurons la plupart des données de stockage. Et enfin, les multiples pour faciliter les quantités qui sont :

Le **Méga-octet** (1 million), le **Giga-octet** (1 milliard) et le **Téraoctet** (1000Go ou 1 trillion d’octet).

Mais quand le nouveau matériel est arrivé, tout a évolué très vite. Pour la même quantité de données, les serveurs sont passés de 98 % à 4 % !! Et voici une partie des éléments qui composent actuellement le Datacenter de mon entreprise.



Comme vous pouvez le voir, chaque partie peut contenir **24 To** de données, alignés sur plusieurs étages. Redoutable d’efficacité. Si j’avais le choix, je ferai exactement comme eux, pas de latence car proximité évidente, sécurité maximum, pas de surprises au niveau des factures, quantité de stockage suffisante pour plusieurs années. Je signale également le fait qu’il y’a un **backup** (si tout disparait, les données sont sauvegardées dans un autre Datacenter en toute sécurité).

## SSIS

**SSIS :** ou **SQL Server Intégration Services**, c’est un outil de **Microsoft SQL Server** qui permet de gérer et d'automatiser le processus **ETL**. Pour le résumé rapidement, c’est un script **SQL** qui va exécuter automatiquement une série d’action dans un système (Dans l’exemple suivant ce sera une Base de données) et rendre quelque chose en sortie (Ici un mail qui va faire une synthèse en pourcentage du nombre de dossiers traités ou pas, ainsi qu’un compteur de notifications) **(C27)**

**ETL :** ou **Extraction** (récupération des données), **Transformation** (modification des données si nécessaire) et **Load** (chargement des données dans une base de données). C’est un ensemble de processus utilisés pour collecter des données à partir de sources variées (essentiellement tout ce qui peut contenir de la donnée), les transformer et les charger dans une base de données.

Dans les screenshots suivant, je vous montre des SSIS qui vont rassembler, modifier et afficher, dans un mail, un suivi des performances de l’application « **Espace Commun »** chez AEF(C’est une application de mise à disposition des données d’incidents pour les clients).



Et donc le visuel final dans le mail (Les données sont réelles, merci de ne pas prendre de notes)



## Transactions PayPal

Le code qui suit est une base pour intégrer des paiements **PayPal** dans une application web en collectant des données du formulaire et en utilisant **l'API PayPal** pour effectuer la transaction. Le mode « **sandbox** » est utilisé pour effectuer des tests sans effectuer de transactions réelles (Le code réel est un peu trop long, je vais en expliquer le principal). **(C29)**

1. **Inclusion des bibliothèques** :
   * Le code commence par inclure les bibliothèques nécessaires pour appeler avec l'API PayPal.
2. **Configuration du contexte PayPal** :
   * On crée un objet **$apiContext** et on le configure avec les informations d'identification du client PayPal (Client ID et Client Secret).
   * Le mode "**sandbox**" est spécifié pour effectuer des tests.
3. **Définition du payeur** :
   * Un objet **$payer** est créé pour spécifier la méthode de paiement, qui est "**paypal**".
4. **Définition de l'article** :
   * On crée l’objet **$itemSold** en récupérant des données depuis un formulaire **POST** (nom de l'article et montant).
   * Cet article sera en dollars **(USD : Dollar United State)** et la quantité est fixée à 1 (ligne 31 => 33).
5. **Création d'une liste d'articles** :
   * Un objet **$itemList** est créé et contient l'article qu’on vient de créer.
6. **Définition du montant de la transaction** :
   * L’objet **$amount** est créé avec le montant total récupéré depuis le formulaire **POST** à la ligne 30 (USD)
7. **Création de la transaction** :
   * On rassemble tout ce qui a été créé plus tôt en le mettant dans l’objet **$transaction** et on attribue la description de la transaction (Le paiement), la liste d'articles et le montant.
8. **Définition des URL de redirection** :
   * On ne le voit pas sur le screenshot mais, on crée l’objet **$redirectUrls** pour spécifier les URL de retour après le paiement, comme la page de succès et d'annulation.

Ensuite, on fait la **Création du paiement** : **$payment** en définissant l'intention « vendu », le payeur, la transaction, et les URL de redirection.

Pour finir, on gère les cas **d'erreur** lors de la création du paiement, le code gère l'exception « **PayPalExceptionPayPalConnectionException $ex »** en affichant des informations de débogage. **(echo $ex->getData() ;**

## CI/CD (Partie 2)

Comme mentionné précédemment **(Page 28)**, le concept de **CI/CD (Continuous Integration/Continuous Deployment)** est une approche de développement logiciel visant à automatiser la construction, les tests et le déploiement des applications de manière continue. Cette automatisation permet de simplifier les mises en production ou d'optimiser les opérations répétitives, notamment en utilisant des scripts en ligne de commande pour accélérer le processus.

Imaginons que pour créer un nouveau client souhaitant acheter notre produit, la personne en charge doit accéder à la **base de données**, puis créer une nouvelle ligne contenant le nom, le prénom, l'ID, le produit acheté et le montant de la transaction manuellement. Une fois cela fait, nous devons accéder à **un tableau Excel** où nous conservons l'ID, les transactions et le prix. Ensuite, nous devons accéder à un **autre tableau** où nous enregistrons le nom, le prénom et l'ID. Enfin, nous devons accéder à un **dernier tableau** pour enregistrer toutes les données concernant la personne.

Mettre bout à bout toutes ces étapes impliquent plusieurs détails intéressants : le risque d'erreur de frappe est important, tout comme le risque d'oublier de remplir l'un des tableaux.

En automatisant l'ensemble de ce processus, nous pourrions créer un script qui récupérerait toutes ces données et les déploierait dans les bons tableaux et dans la base de données en une seule fois et de manière automatique. Pour illustrer cela, le code pourrait être contenu dans un fichier nommé « CDbash.sh » et ressemblerait à ceci :



Le fait de remplir ce « **bash »** fera gagner du temps à la personne qui gère les nouvelles inscriptions et évitera au passage les erreurs potentielles (Sans parler bien sûr du bien que cela fera à l’utilisateur).

## Architecture BDD

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RESUME DES CHOSES A FAIRE ET MONTRER

Résumé des abréviations (mvs, ide, api ectect )

CI/CD => sonarqube, CI fait / CD avec Damien

Monetisation de l’appli / site discord Batou

Observation/monitoring de la BDD avec Aurelien

Sécurité => voir les cours RootMe, owasp

# Conclusion

1 Page max, 400 / 450 mots Max

# CV

