

Conception d'une application modulaire pour une passerelle de visualisation intégrant des données issues de divers dispositifs

Travail de Bachelor

Non confidentiel

**Département :** TIC

**Filière :** Informatique et systèmes de communication

**Orientation :** Informatique logicielle

Nicolas Crausaz

11 avril 2023

Travail proposé par :

Loris Gavillet

YALK

Rue Basse 43, 1422 Grandson

Supervisé par :

Patrick Lachaize

# Préambule

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d’études, en vue de l’obtention du titre de Bachelor of Science HES-SO en Ingénierie.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d’auteur.

HEIG-VD

Le Chef du Département

Yverdon-les-Bains, le 20 avril 2023

# Authentification



Le soussigné, Nicolas Crausaz, atteste par la présente avoir réalisé seul ce travail et n’avoir utilisé aucune autre source que celles expressément mentionnées.

Nicolas Crausaz

Yverdon-les-Bains, le 20 avril 2023

# Résumé

Cliquez ou appuyez ici pour entrer du texte.

**Table des matières**

[Préambule I](#_Toc133238231)

[Authentification III](#_Toc133238232)

[Résumé V](#_Toc133238233)

[Chapitre 1 Introduction 1](#_Toc133238234)

[1.1 Contexte 1](#_Toc133238235)

[1.2 Cahier des charges 2](#_Toc133238236)

[**1.2.1** **Besoins fonctionnels** 2](#_Toc133238237)

[**1.2.2** **Besoins non fonctionnels** 2](#_Toc133238238)

[**1.2.3** **Extensions** 3](#_Toc133238239)

[1.3 Contraintes client 4](#_Toc133238240)

[1.4 Tableaux 5](#_Toc133238241)

[1.5 Notes de bas de page 5](#_Toc133238242)

[1.6 Citations et bibliographie 5](#_Toc133238243)

[Chapitre 2 Analyse 6](#_Toc133238244)

[2.1 Division des objectifs 6](#_Toc133238245)

[**2.1.1** **Modules et gestionnaire** 6](#_Toc133238246)

[**2.1.2** **Application Web** 8](#_Toc133238247)

[**2.1.3** **Serveur et API** 8](#_Toc133238248)

[**2.1.4** **Base de données** 9](#_Toc133238249)

[2.2 Module « Proof of Concept » 10](#_Toc133238250)

[2.3 Planification 11](#_Toc133238251)

[**2.3.1** **Livrables** 11](#_Toc133238252)

[**2.3.2** **Sprints** 11](#_Toc133238253)

[2.4 Prototypes et essais effectués 13](#_Toc133238254)

[**2.4.1** **Architecture serveur HTTP** 13](#_Toc133238255)

[**2.4.2** **Architecture Electron.js** 14](#_Toc133238256)

[**2.4.3** **Choix** 14](#_Toc133238257)

[2.5 Choix technologiques 15](#_Toc133238258)

[**2.5.1** **Choix généraux** 15](#_Toc133238259)

[**2.5.2** **Backend** 15](#_Toc133238260)

[**2.5.3** **Frontend** 15](#_Toc133238261)

[Chapitre 3 Modélisation 18](#_Toc133238262)

[Chapitre 4 Architecture 21](#_Toc133238263)

[Chapitre 5 ?? 21](#_Toc133238264)

[Chapitre 6 Conclusion 22](#_Toc133238265)

[Bibliographie 23](#_Toc133238266)

[Annexes 24](#_Toc133238267)

**Table des figures**

[**Figure 1** - Exemple de graphique 1](#_Toc129262255)

**Liste des tableaux**

[**Tableau 1** - Liste des cantons 2](#_Toc129262231)

**Liste des codes sources**

[**Listing 1** - Exemple de code Java 1](#_Toc129262239)

## Introduction

**Ce travail de Bachelor consiste en la mise en place d’une application Web, destinée à s’exécuter sur une passerelle physique dans le réseau local d’un utilisateur. L’intérêt de cette passerelle est d’offrir un outil central pour mettre en place des interactions avec des services externes et dispositifs réseau, sous forme de modules. Cette solution permet à des utilisateurs d’automatiser des actions en utilisant / développant des modules interagissant avec leurs propres services existants. La passerelle reliée à un ou plusieurs moniteurs permettra l’affichage d’informations issues des interactions grâce à un tableau de bord modulable.**

**Ce travail revient à effectuer un « Proof of Concept » permettant de mettre en place une architecture de base et son implémentation, ainsi qu’un module, pour démontrer les possibilités et les limites d’une telle solution.**

**TODO…**

### Contexte

**TODO**

Une image contenant diagramme

Description générée automatiquement

### Cahier des charges

#### **Besoins fonctionnels**

**Les besoins fonctionnels suivants sont considérés comme des éléments de priorité, composant les objectifs fondamentaux du travail.**

*****Modules*****

**Un module est un composant logiciel destiné à offrir des fonctionnalités à la passerelle, principalement de gérer les interactions avec un dispositif précis : une API externe, un dispositif USB, Wifi etc. Les besoins liés aux modules sont :**

1. **Gestionnaire : Permettant l’activation / désactivation et la configuration du comportement d’un module.**
2. **Conception module « Proof of Concept » : Développement d’au moins un module spécifique permettant de gérer l’interaction entre un dispositif local et un service externe, permettant de démontrer l’intérêt de la solution.**

*****Application Web*****

1. **Structure : L’application offre deux interfaces : administration et visualisation. L’interface d’administration nécessite une authentification de l’utilisateur, la visualisation est accessible sans authentification, car elle ne propose pas de fonctionnalité d’édition. Les deux interfaces sont accessibles uniquement dans le réseau local.**
2. **Interface d’administration : permet l’activation / désactivation et la configuration des modules par l’utilisateur. Permet aussi la configuration du tableau de bord modulable, en laissant le choix à l’utilisateur de disposer les affichages des modules selon ses préférences.**
3. **Interface de visualisation : L’interface de visualisation affiche le tableau de bord selon la configuration effectuée dans l’administration. Lors d’une modification de la disposition des modules depuis l’administration, le tableau de bord devra se mettre à jour sans nécessiter d’interaction supplémentaire de l’utilisateur.**

#### **Besoins non fonctionnels**

*****Support physique et système d’exploitation*****

**La passerelle sera un Raspberry Pi relié au minimum à une source de courant, une connectivité réseau (avec ou sans fil) et pouvant être reliée à un ou deux moniteurs externes. Le système d’exploitation contiendra le nécessaire pour faire fonctionner l’application Web.**

*****Architecture locale*****

**Tout le nécessaire au bon fonctionnement de l’application s’exécutera sur la passerelle. La passerelle et son application web ne seront pas exposée en dehors du réseau local, mais pourront interagir avec des services externes (http).**

*****Performance*****

**Du fait de l’architecture (ressources) de la passerelle relativement limitée, l’application s’exécutera sans consommation trop excessive de ressources et répondra dans un temps acceptable de manière à offrir une utilisation fluide.**

*****Interface intuitive*****

**L’interface de l’application web sera intuitive à utiliser pour un utilisateur ayant un bagage technique lui permettant de comprendre et configurer les modules et interactions.**

*****Application fiable*****

**L’application doit être capable de gérer des erreurs pouvant survenir et de pouvoir continuer à fonctionner. En cas de perte de connectivité puis de reconnexion réseau, l’application devra pouvoir refonctionner correctement sans interaction utilisateur.**

**En cas d’erreur non récupérable, il sera acceptable dans ce projet de simplement redémarrer l’application ou la passerelle.**

*****Sécurité*****

**L’application n’est pas conçue pour gérer des interactions critiques pouvant avoir des aspects de sécurité importants (transactions financières, etc.). Dans le cadre de ce travail, il n’est pas non plus nécessaire d’implémenter un système de sauvegarde de la configuration de la passerelle ou de sa réplication.**

#### **Extensions**

**Les extensions ci-dessous sont des besoins fonctionnels complémentaires faisant partie de la solution complète souhaitée par l’entreprise. Ces dernières ne seront pas réalisées dans le cadre de ce projet, sauf si le temps le permet, mais qu’il faut garder en tête lors de la conception des autres besoins pour prévoir leur ajout ultérieur.**

*****Modules*****

**Dans la version souhaitée par les besoins fonctionnels, tous les modules sont stockés localement sur la passerelle. Il serait intéressant de mettre en place un système d’import de nouveaux modules par l’utilisateur, afin de mettre en avant l’aspect modulaire de l’application.**

*****Autonomie*****

**Mise en place et configuration automatique / assistée de la passerelle dans le réseau local de l’utilisateur. Ceci permet de s’assurer une configuration correcte et connue pour l’exécution de l’application.**

*****Gestion des écrans*****

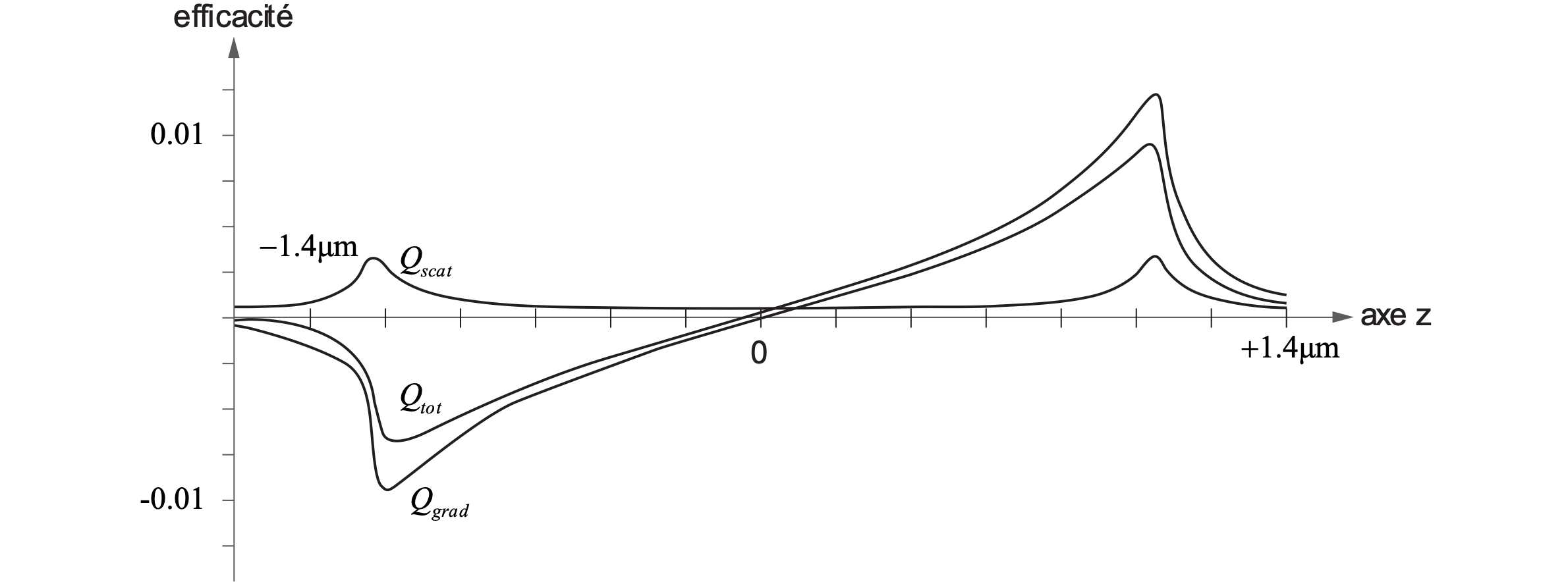
**Détection des écrans reliés à la passerelle pour pouvoir adapter le contenu du tableau de bord, selon l’orientation, la taille, la disposition relatives des écrans. Ceci permettrait d’ajouter un indicateur visuel et une aide à l’utilisateur configurant le tableau de bord.**

*****Système d’exploitation*****

**Il serait imaginable de créer une image d’un système d’exploitation contenant le strict nécessaire au fonctionnement de l’application. Ceci permettrait de limiter les interactions possibles de l’utilisateur final avec le logiciel, améliorant la sécurité et la stabilité de l’application.**

### Contraintes client

Pour présenter vos résultats d’expérience, vous pouvez soit dessiner des graphiques manuellement en utilisant des outils de dessin vectoriel comme Inkscape ou Adobe Illustrator, comme illustré à la Figure 1.



**Figure 1** - Exemple de graphique

Notez que la coloration syntaxe est conservé lorsque vous copier-coller du code depuis Intellij IDEA. Le Listing 1 en est un exemple.

**Listing 1** - Exemple de code Java

### Tableaux

Todo : Définir vocabulaire

Concernant les tableaux un seul conseil : restez simple et minimaliste, n’ajoutez des séparateurs que là ou c’est nécessaire pour améliorer la lisibilité. Une liste de quelques cantons suisses est donnée à titre d’exemple dans le Tableau 1.

|  |  |  |
| --- | --- | --- |
| Abréviation | Nom du canton | Depuis |
| ZH | Zürich | 1er mai 1351 |
| BE | Berne | 6 mars 1353 |
| FR | Fribourg | 22 décembre 1481 |
| VD | Vaud | 19 février 1815 |
| VS | Valais | 4 août 1815 |
| NE | Neuchâtel | 19 mai 1815 |
| GE | Genève | 19 mai 1815 |

**Tableau 1** - Liste des cantons

### Notes de bas de page

Parfois, il est plus élégant d’annoter une définition en utilisant une note de bas de page [[1]](#footnote-1). Alternativement il est possible d’annoter un paragraphe avec une note marginale.

### Citations et bibliographie

Citer vos sources est essentiel. Avec **Microsoft Word** vous pouvez facilement citer des articles, des livres ou des sites internet. Toutes les citations dans le texte seront automatiquement regroupées en fin de document dans la section « Bibliographie ». Par exemple, citons un article d’Einstein (EINSTEIN 1905) ou le livre de Dirac (DIRAC 1981).

Parfois il peut être utile d’utiliser un gestionnaire de bibliographie. La communauté académique recommande l’outil [Zotero](https://www.zotero.org/) qui permet de gérer une bibliothèque numérique d’ouvrages et de références numériques. Il permet également de générer une bibliographie compatible avec **Microsoft Word**. Il suffira d’installer au préalable le plugin, pour pouvoir ajouter l’onglet *Zotero*.

## Analyse

### Division des objectifs

**La première partie de l’analyse a consisté en la définition des fonctionnalités selon les besoins explicités au chapitre précédent. Cette division a permis de décomposer les bases de l’architecture de l’application, en séparant les fonctionnalités dans plusieurs composants ayant chaque leur rôle respectif.**

#### **Modules et gestionnaire**

**Les modules sont l’aspect central de l’application, ils permettent d’offrir des fonctionnalités à la passerelle. Leur rôle est de fournir le comportement nécessaire afin de permettre l’interaction entre deux services ou dispositifs n’ayant pas initialement été conçu ou ne pouvant pas nativement interagir entre eux.**

**Un module est composé de plusieurs parties :**

* ***Un comportement*: la logique et les interactions offertes par le module y sont définie, sous forme de code.**
* ***Une configuration* : Afin de rendre le comportement d’un module personnalisable par l’utilisateur final, une configuration du module pourrait être plus ou moins spécifique dans l’objectif d’offrir une plus grande portée au comportement du module**
* ***Un affichage* : il s’agit du rendu du module pouvant être affiché sur l’interface de l’application (voir point 2.1.x)**

**Ainsi, plusieurs fonctionnalités sont à prévoir :**

* **Lors de sa première installation, un module peut exécuter une série d’instruction spécifique au module pour assurer son bon fonctionnement.**
* **La configuration d’un module sera composée d’informations obligatoire (par exemple le nom du module) et de valeurs personnalisées dépendant du module. Ces valeurs seront définies par le développeur du module et pourront être modifiées. Cette configuration devra être accessible dans le comportement du module afin de pouvoir modifier ce comportement selon ces valeurs.**
* **Un module sera distribué sous la forme de paquet indépendant, contenant le nécessaire à son intégration à la passerelle. Il est donc nécessaire de prévoir cette une structure adaptée à cela. Il serait également intéressant de fournir un module d’exemple documenté pouvant servir de base aux développeurs pour faciliter la création de modules. De ce fait il est donc nécessaire de concevoir l’architecture logicielle d’un module de manière à la rendre configurable et de permettre facilement la création de nouveaux modules.**
* **Un module est destiné à pouvoir être ajouté, supprimé, activé et désactivé à la demande de l’utilisateur, ces actions ne doivent ainsi ne pas générer d’erreurs et gérer la libération des ressources liées aux modules.**
* **Le comportement d’un module doit pouvoir recevoir des données et en envoyer, notamment pour mettre à jour son affichage.**

**Les différents besoins liés aux modules nécessiteront donc une entité qui permet leur gestion, notamment pour s’assurer de la cohérence de leurs états, ceci afin de faciliter leur ajout à la plateforme et d’éviter les erreurs. Ainsi, un gestionnaire de module doit être mis en place.**

**Le gestionnaire de module permet d’assurer le bon fonctionnement et la cohabitation des différents modules, il est notamment chargé d’offrir les fonctionnalités suivantes :**

* **Collecter les modules et maintenir leur état de manière centralisée**
* **Permettre l’ajout et la suppression de modules, en s’assurant de les initialiser correctement et de charger leur configuration.**
* **Permettre l’activation et la désactivation de modules**
* **Permettre le chargement dynamique des modules, sans devoir effectuer quelconque compilation ou redémarrage de l’application.**

**Le besoin d’ajout et de suppression des modules étant des extensions du cahier des charges, leur modélisation sera mise en place pour permettre leur implémentation si le temps le permet.**

#### **Application Web**

**L’objectif de ce travail est de fournir une interface web, accessible dans le réseau local de la passerelle, offrants à l’utilisateur les diverses fonctionnalités liées aux modules cités dans les points précédents et plus encore. L’interface web de l’application est séparée conceptuellement en deux parties, chacune jouant un rôle distinct :**

* ***Interface d’administration* : permet toutes les opérations liées aux modules citées dans les points précédents ainsi que la configuration du tableau de bord. Le tableau de bord permet de disposer les différents affichages spécifiques aux modules, sous forme d’une grille personnalisable par l’utilisateur. Pour accéder à cette interface, l’authentification de l’utilisateur est nécessaire.**
* **Interface de visualisation : permet l’affichage en lecture seule du tableau de bord configuré dans l’interface d’administration. Cette interface est destinée à être affichée sur l’écran directement relié à la passerelle, elle ne nécessite donc pas d’authentification utilisateur et ne propose pas d’interactions directe avec l’utilisateur.**

**L’interface complète sera donc composée des pages suivantes :**

* ***Connexion* : Authentification de l’utilisateur par nom d’utilisateur et mot de passe.**
* ***Modules* : Listes des modules disponibles localement sur la passerelle, possibilités d’activation, désactivation, ajout (extension) et suppression (extension) des modules.**
* ***Configuration de module* : Affichage d’un unique module, modification de sa configuration, activation et désactivation**
* ***Tableau de bord (édition)* : Configuration de la disposition des affichages des modules selon une grille. Possibilité d’avoir une disposition différente pour chaque écran.**
* ***Tableau de bord (affichage)* : Affichage du tableau de bord selon la configuration effectuée et selon l’écran choisi.**
* ***Paramètres* : Configuration générale des informations de la passerelle. Cette page n’est pas explicitée dans le cahier des charges mais permettrait dans le futur plusieurs ajouts (gestion utilisateurs …)**

#### **Serveur et API**

**L’application sera composée d’un serveur (backend) permettant de traiter les requêtes venant de l’interface, ainsi que de permettre la communication vers et depuis les modules.**

**Il sera donc nécessaire de définir les « specs » de l’API**

**Authentification (deux types ?)**

**Utile à l’interface**

**Permet des interactions LAN**

**Rest ?**

#### **Base de données**

**Afin d’assurer la persistance des données en cas d’arrêt de la passerelle, une base de données devra être mise en place. Elle permettra de stocker les données suivantes :**

* ***Utilisateurs* : comptes pouvant se connecter à l’application web**
* ***Modules* : État d’activation des modules et leur configuration actuelle.**
* ***Dashboard* : Configuration du tableau de bord crée par l’utilisateur, ainsi que la gestion des écrans multiples.**

Il sera nécessaire d’établir quel système de gestion de base de données utiliser..

#### **Module « Proof Of Concept »**

TODO : Choix de plusieurs idées de modules PoC

**Interactions entre les composants**

EventEmitter (Pattern Observer)

http (frontend – backend)

aspect evenement orienté serveur

Server Sent Events vs WebSockets

API keys

JWT

ServerSide rendering ?

**Serveur**

API accessible au réseau local

Rest ?

Interface

Application web client

Listes des fonctionnalités par pages

L’analyse de tous les points précédents permettent de modéliser conceptuellement l’architecture complète.

TODO : ajouter DB au shéma

Une image contenant diagramme

Description générée automatiquement

### Planification

**Le projet se déroule du lundi 20 février au juillet 27 juillet 2023. Une présentation viendra finalement clôturer ce travail en septembre 2023.**

#### **Livrables**

**Plusieurs livrables sont attendus :**

* **Un rapport intermédiaire, détaillant le travail produit jusqu’à cette étape, au 26 mai.**
* **Un rapport final, détaillant le travail complet du projet, au 27 juillet.**
* **Un résumé publiable ainsi qu’une affiche, permettant de présenter et verbaliser le projet.**
* **L’application, selon les critères explicités précédemment.**

#### **Sprints**

**J’ai choisi de travailler sous forme de sprint, généralement d’une durée de 1 à 3 semaines tout au long du projet. Cela me permet de d’avancer rapidement en priorisant les fonctionnalités principales à chaque itération. C’est également un bon moyen de faire un retour concret au mandant et au professeur en fin de sprint. Après avoir isolé tous les besoins (point 2.1), j’ai choisi de réaliser les sprints suivants :**

*****Sprint 1*** *– 20.02 au 03.03 – Cahier des charges***

**Ce premier sprint consistera à établir les besoins avec le mandant, de manière à rédiger un cahier des charges du projet.**

*****Sprint 2*** *– 04.03 au 17.03 – Architecture et outils***

**L’objectif sera d’analyser les besoins pour en extraire les différents composants de l’application et de mettre en place une première architecture. Il s’agira ensuite de choisir les technologies et de créer la base de code utilisée pour le projet, ainsi que la mise en place des différents outils.**

*****Sprint 3*** *– 18.03 au 07.04 – Prototypages, test et analyse***

**Dans cette itération, l’objectif sera de d’effectuer différents prototypes, visant à évaluer la faisabilité et d’obtenir une base d’architecture exploitable. Il sera potentiellement nécessaire de comparer plusieurs approches et de comparer leur intérêt.**

*****Sprint 4*** *– 08.04 au 21.04 – Modules***

**Ce sprint portera sur la modélisation et l’implémentation des fonctionnalités relatives aux modules. Étant l’élément central du projet, la bonne conception de cet aspect est très critique. La création d’un premier module simple pour prouver le fonctionnement de l’intégration de modules est à prévoir.**

*****Sprint 5*** *– 22.04 au 14.05 – API et backend***

**L’accent sera mis sur les fonctionnalités côté serveur, telles que la définition de l’API, la modélisation de la base de données et l’authentification. Le développement d’un module « PoC » sera effectué.**

*****Sprint 6*** *– 15.05 au 26.05 – Refactoring et rapport intermédiaire***

**Ce court sprint permettra d’effectuer du refactoring du code existant, puis de terminer la rédaction du rapport intermédiaire de projet. Le rendu de ce rapport clôturera le sprint.**

*****Sprint 7*** *– 27.05 au 18.06 – Frontend***

**L’objectif de ce sprint est d’assembler l’interface de manière à ce qu’elle offre les fonctionnalités nécessaires qui n’ont pas encore été complètement implémentées dans les sprints précédents. Ceci comprendra entre autres la mise en place de l’authentification ainsi qu’en grande partie la configuration du tableau de bord et son affichage sur l’interface de visualisation. Ce sprint marque la fin du semestre académique, le temps attribué à la réalisation du projet sera désormais de 5 jours par semaine.**

*****Sprint 8*** *– 19.06 au 21.06 – Refactoring et documentation***

**Le début du travail à plein temps sur le projet sera une bonne occasion de prendre du recul sur le travail accompli et d’effectuer une session de refactoring et de documentation.**

*****Sprint 9*** *– 22.06 au 09.06 – Finalisation et tests***

**Ce sprint permettra de terminer les fonctionnalités si nécessaire, et d’effectuer des tests globaux sur l’application afin de pouvoir juger sa performance et sa capacité et s’exécuter de manière autonome. Ce sprint permettra également la rédaction de la documentation et du rapport de projet.**

*****Sprint 10*** *– 10.06 au 27.07 – Livrables***

**L’objectif cet ultime sprint sera de clôturer le projet en retravaillant les éventuels derniers détails d’implémentation, mais principalement la finalisation de rédaction du rapport final et des autres livrables attendus.**

### Prototypes et essais effectués

**Durant cette phase initiale d’analyse, j’ai effectué une série d’essais d’architecture et de technologies relativement variés. Ces essais, plus au moins concluants, m’ont permis d’évaluer et d’assurer mes choix de technologies et d’architecture finaux. Ce chapitre détaille les différents prototypes réalisés pour analyser la faisabilité des différentes approches.**

**Deux architectures pouvant répondre aux différents besoins de l’application ont été prototypées afin de départager le choix final. Ces deux architectures répondent au besoin de fonctionnement de manière locale sur un réseau privé de l’utilisateur, sans dépendre de la conception de services externes à ce réseau et permettent de répondre aux besoins finaux de l’application.**

#### **Architecture serveur HTTP**

**La première approche envisagée consiste en une architecture relativement classique, composée d’un serveur HTTP, offrant une interface web ainsi qu’une API. Ce serveur, fonctionnant sur la passerelle, offre une API pour gérer les intégrations avec l’interface utilisateur et également de permettre à des dispositifs locaux (LAN) de communiquer avec la passerelle au travers du protocole HTTP.**

Une image contenant diagramme

Description générée automatiquement

Là où l’architecture devient

Définir les moyens de communications

#### **Architecture Electron.js**

TODO

Fonctionne avec les IPCs

Auto-Updater

Le problème d’electron.js :

Ecosystème assez restrictif et strict, risque de limitation quand au intégration faisable.

Les modules doivent s’exécuter dans des fenètre cachées -> pas vraiment moyen de les réafficher centralement

Accès au réseau. L’objectif d’élèctron est de concevoir des application client, destiné à s’exéctuter sur un poste client.

Dans mon cas, cette application serait sur le serveur mais embarquerai encore à l’interne un serveur http pour permettre la connexion par les utilisateurs. Le problème est que les vues devraient donc être servies par le serveur http, ce qui casse l’infrastructure Electron.

Utilisation

Une image contenant texte

Description générée automatiquement

Lister les tests d’architecture effectués et les technologies testées

Parler de l’archi API – App client, Electron

Parler des tests effectués pour le rendu

#### **Choix**

La première architecture a été retenue.

### Choix technologiques

#### **Choix généraux**

J’ai choisi un langage commun : TypeScript

TypeScript

Jest

#### **Backend**

L’environnement Node.js

Express

SQLite

Sql vs orienté document

#### **Frontend**

React

Tailwind, DaisyUI

Approches

Restrictions, limitations

Tests effectués

Choix des technologies

Gestion des erreurs

Support physique / déploiement sur Raspberry PI

### Méthodologies et outils

#### **CI / CD**

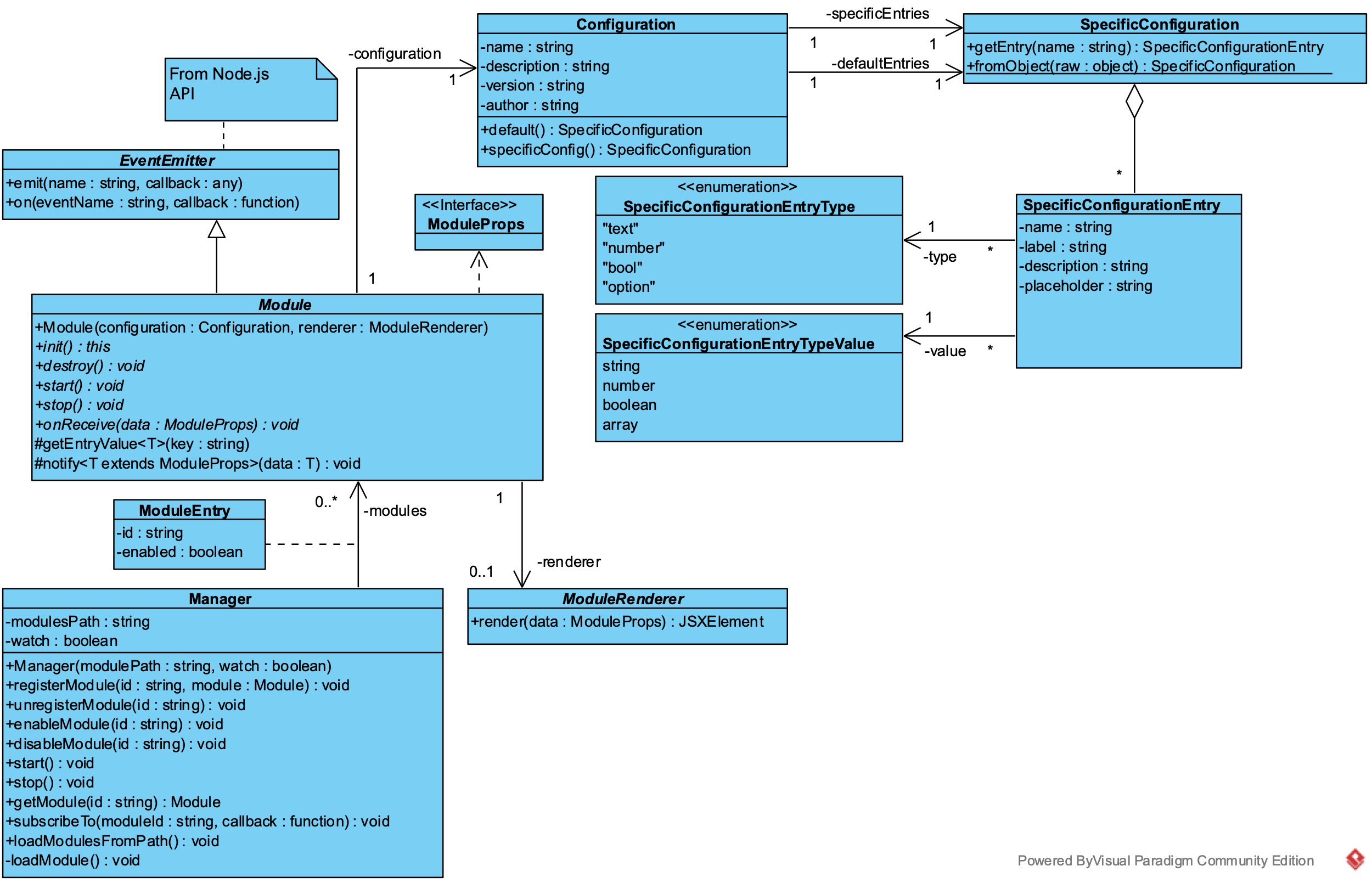
#### **Production**

Comment effectuer le déploiement sur le raspberry pi. Comment assurer le bon fonctionnement ? Docker ?

## Modélisation

La deuxième phase du travail a consisté en la modélisation logicielle des différentes fonctionnalités qui ont été determinée lors de la phase d’analyse

Modélisation du système de module



TODO : Diverses explication

TODO : zoom sur le module :

* Système d’event
* Code d’exemples
* Structure d’un module (package) et objectif de sa mise a disposition (-> communauté / devs)

TODO : zoom sur la configuration

* Interet et configurable par user
* Fichier JSON

TODO : zoom sur Manager

* Point central

UMLs, graphiques

CI/CD

Schéma DB (modélisation) et explications

Monorepo

Mockups

Structure d’un module : TS vers JS, 3 fichiers, fichier de type ?

**Mettre en avant l’aspect utilisateur, développeur / communauté**

Restrictions, limitations

## Architecture

Communication entre les elèments

Backend et frontend

## ??

## Conclusion

Cliquez ou appuyez ici pour entrer du texte.

**Nicolas Crausaz**

# Bibliographie

DIRAC, Paul Adrien Maurice. «The Principles of Quantum Mechanics.» *International series of monographs on physics.* Clarendon Press, 1981.

EINSTEIN, Albert. «Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies].» *Annalen der Physik* 322, n° 10 (1905): 891-921.

# Annexes

**Glossaire**

**Index**

Adobe Illustrator, 1

Inkscape, 1

1. La note en bas de page (ou note de bas de page) est une forme littéraire, consistant en une ou plusieurs lignes ne figurant pas dans le texte. [↑](#footnote-ref-1)