Hes·so VALAIS WALLIS
Haute Ecole de Gestion & Tourisme
Hochschule für Wirtschaft & Tourismus

Travail de Bachelor

Information and Communication Technologies (ICT)

# Création de composants BI custom dans Power BI

Auteur:

Blendar Berisha

Professeur :

Prof. Cosette Bioley

Déposé à la HES-SO Valais-Wallis à Sierre, le 9 août 2025

# Résumé

Les visuels standard de Power BI ne couvrant plus l'ensemble des besoins analytiques des clients d'ECRINS SA, l'entreprise souhaite internaliser la création de **visuels personnalisés** (*custom visuals*). Ce travail de Bachelor poursuit deux finalités : *(1)* élaborer un **cadre méthodologique complet** — de l'analyse fonctionnelle jusqu'au déploiement automatisé — pour développer ces composants, et *(2)* en démontrer la faisabilité au moyen d'un **prototype de visuel pilote**.

La démarche s'appuie sur une étude critique des visuels natifs, un benchmark d'autres plateformes BI et une collaboration étroite avec la responsable produit afin de cibler un besoin prioritaire. Le développement adopte un processus itératif inspiré des méthodes Agiles : configuration de l'environnement `pbiviz`, implémentation en TypeScript / D3, tests et intégration continue via GitHub Actions, puis documentation et bonnes pratiques de gouvernance.

L'évaluation combine des tests techniques, une revue de code et une validation fonctionnelle auprès du client interne. Les livrables — code source, pipeline CI/CD opérationnel et guide d'intégration — constituent une base réutilisable pour accélérer la livraison future de visuels sur mesure tout en renforçant la gouvernance BI d'ECRINS SA.

**Mots clés :** Business Intelligence ; Power BI ; visuels personnalisés ; CI/CD ; gouvernance

# Remerciements

Avant d'entamer la lecture de ce travail, je souhaite exprimer ma profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à son aboutissement.

**À ma directrice de Bachelor, Prof. Cosette Bioley,** pour sa rigueur scientifique, ses retours toujours constructifs et la confiance accordée dès les premières discussions. Son exigence académique a largement façonné la qualité de ce mémoire.

**Aux enseignantes et enseignants de la filière ICT de la HES-SO Valais,** qui m'ont transmis les fondements théoriques et pratiques indispensables à la réalisation de ce projet.

**À mes camarades de promotion,** pour l'entraide quotidienne, les revues de code improvisées et l'indispensable bonne humeur qui ont rythmé ces mois intenses.

**À ma famille et à mes proches,** pour leur soutien inconditionnel, leur patience face aux longues soirées de développement et leurs encouragements constants.

Que chacune et chacun trouve ici l'expression de ma reconnaissance la plus sincère.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Contexte

Microsoft Power BI s'est imposé ces dernières années comme l'un des outils phares de la Business Intelligence (BI) en entreprise. Il a réussi à reléguer nombre de concurrents au second plan en offrant une solution de visualisation et d'analyse de données intégrée à un coût d'entrée très attractif. De fait, Power BI est devenu extrêmement populaire auprès des organisations et des utilisateurs, au point que près de 30 % des entreprises le plébiscitent déjà et que 60 % envisagent de l'adopter, selon la *BI Survey 23* de BARC (Spies & Mertens, 2023; Tirupati, Joshi, & Singh, 2023).

L'un des atouts majeurs de Power BI réside dans la richesse de ses fonctionnalités natives, en particulier sa large gamme de visuels prédéfinis qui s'enrichit continuellement. Chaque mise à jour de l'outil apporte de nouveaux graphiques et tableaux de bord standards, couvrant un éventail important de besoins analytiques courants. Au-delà de ces visuels par défaut, Power BI offre également la possibilité d'étendre ses capacités en intégrant des composants personnalisés développés à l'aide de langages scripts tels que Python ou R. Cette ouverture permet aux analystes de réaliser des analyses avancées et de créer des visualisations sur mesure allant au-delà de l'offre standard de l'outil (Dossier, 2024).

En d'autres termes, les utilisateurs bénéficient d'une liberté supplémentaire pour représenter leurs données de façon plus adaptée et innovante qu'avec les seuls graphiques fournis en standard. Cette tendance vers la personnalisation des tableaux de bord est largement documentée dans la littérature académique, qui y voit un facteur clé d'appropriation et de satisfaction utilisateur (Amyrotos, Vassiliadis, & Karacapilidis, 2024).

C'est dans ce contexte technologique et métier qu'intervient la société **ECRINS SA**, une entreprise de conseil en informatique de gestion basée en Valais. ECRINS compte parmi ses clients divers acteurs de premier plan — tels que le Service Industriel de Genève, Rolex, Genève Aéroport ou Tag Heuer — aux besoins décisionnels pointus. Ces clients sollicitent régulièrement des analyses et indicateurs « sur mesure » que les outils standards ne peuvent pas toujours fournir tels quels. Désireuse de maintenir un haut niveau de satisfaction client, l'entreprise cherche à exploiter la flexibilité de Power BI pour concevoir des composants BI *custom*, c'est-à-dire des visuels personnalisés intégrés à des tableaux de bord Power BI. À travers ce travail de Bachelor, elle ambitionne de développer quelques exemples probants de tels visuels et de définir une méthodologie reproductible pour leur conception. Ce projet servira ainsi de base de référence pour implémenter à l'avenir de nouveaux composants BI *custom* répondant aux demandes spécifiques de la clientèle, tout en assurant un standard de qualité et de gouvernance.

## 1.2 Problématique

Bien que la perspective de visuels personnalisés soit attrayante, leur mise en place n'a encore jamais été testée par ECRINS. En l'absence d'expérience préalable et de cadres méthodologiques définis, l'entreprise ne dispose pas des repères nécessaires pour évaluer la faisabilité technique ni l'effort de développement qu'implique la création d'un visuel sur mesure dans Power BI. Les clients formulent parfois des demandes qualifiées en interne d'« impossibles », car visant des représentations graphiques ou des fonctionnalités inexistantes dans les visuels standards de Power BI. Jusqu'à présent, ces attentes spécifiques restent soit insatisfaites, soit très difficilement comblées par des solutions de contournement peu élégantes.

La question qui se pose est donc la suivante : *comment ECRINS peut-elle concevoir et intégrer de nouveaux composants BI* custom *dans Power BI afin de répondre à des besoins métier non couverts par les visuels natifs, tout en définissant un cadre de développement fiable pour y parvenir ?*

Il s'agit d'une problématique à la fois technique et organisationnelle. D'un point de vue technique, il faut déterminer les outils et approches de réalisation les plus appropriés (scripts R/Python intégrés, développement d'un *visual custom* via le SDK Power BI, etc.), en tenant compte des avantages et limitations de chaque solution. D'un point de vue organisationnel, la littérature montre que la réussite d'un projet BI dépend d'un alignement entre facteurs technologiques, organisationnels et environnementaux (Al-Kharusi, Al-Harthi, & Al-Hinai, 2023). Il est donc nécessaire d'établir des normes de développement et de déploiement pour garantir que les composants créés soient pérennes, maintenables et aisément déployables dans l'environnement Power BI de l'entreprise.

En somme, ECRINS cherche à élargir le champ des possibilités de Power BI de manière maîtrisée, afin de répondre favorablement, à l'avenir, aux demandes de visuels spécifiques de ses clients. Ce défi s'inscrit plus largement dans la problématique de pousser les limites des visuels BI par défaut — enjeu rencontré par de nombreuses organisations — en recourant à la personnalisation (Uttam & Zhao, 2025). La résolution de cette problématique passera par l'exploration du processus de développement d'un composant BI *custom* de bout en bout, depuis l'identification du besoin jusqu'à la validation du composant final en situation réelle.

## 1.3 Objectifs

Ce projet de Bachelor poursuit un objectif principal : démontrer la faisabilité et l'intérêt de la création de visuels personnalisés dans Power BI, puis formaliser une méthode de développement reproductible. Il s'agit d'analyser l'existant afin de recenser les lacunes, de sélectionner un ou deux visuels prioritaires à forte valeur ajoutée, de déterminer l'approche technique (scripts ou SDK), d'implémenter le composant dans un tableau de bord de démonstration et de le tester, puis de documenter une procédure standard réutilisable par ECRINS.

## 1.4 Portée et limites

Le travail se concentre sur la réalisation d'un prototype unique de composant BI *custom* et sur l'élaboration de recommandations générales, sans prétendre couvrir l'ensemble des possibilités de personnalisation de Power BI ni constituer une bibliothèque exhaustive. Le composant est testé dans un scénario de démonstration reposant sur des données fictives ou publiques et l'environnement Power BI Desktop, éventuellement complété par le service en ligne.

Certaines limites techniques sont reconnues. Lorsqu'un visuel repose sur un script Python ou R, le rendu se présente sous forme d'image statique, excluant les interactions utilisateur avancées ; cette approche peut également entraîner des rafraîchissements plus lents lors des mises à jour de données (Real Python Team, 2023). Pour répondre aux exigences d'interactivité de la clientèle, le présent travail retient donc l'approche SDK TypeScript pour le prototype, malgré la complexité supplémentaire. La certification et la publication éventuelle sur AppSource restent toutefois hors périmètre.

## 1.5 Structure du rapport

Outre la présente introduction, le mémoire s'articule en six chapitres complémentaires. Le chapitre 2 dresse d'abord un état de l'art des visuels Power BI et des solutions concurrentes, afin de situer la problématique dans son contexte technologique. Le chapitre 3 précise ensuite l'origine du besoin, les critères de réussite retenus et l'organisation du travail, ce qui constitue la méthodologie du projet. Le chapitre 4 décrit la conception et la réalisation du visuel personnalisé sélectionné, en détaillant les décisions techniques majeures. Le chapitre 5 formalise la procédure d'industrialisation : pipeline CI/CD, signature numérique et exigences de gouvernance. Le chapitre 6 présente l'évaluation des résultats tant sur le plan technique que fonctionnel, tandis que le chapitre 7 conclut en synthétisant les apports du travail, en discutant ses limites et en ouvrant des perspectives pour ECRINS SA.

# 2 | État de l'art

Le développement de visuels personnalisés dans Power BI s'appuie sur un écosystème technologique riche, combinant les fonctionnalités *natives* de la plateforme et des extensions via code. Ce chapitre présente d'abord l'**architecture** des visuels Power BI et distingue les différentes catégories de visuels disponibles. Ensuite, il examine successivement les **visuels natifs** (ceux fournis par défaut par Microsoft), les **visuels basés sur Python/R** (générés à partir de scripts), et enfin les **visuels personnalisés via le SDK** officiel. Chaque section discute des capacités offertes et des limites inhérentes. Enfin, nous abordons les **choix technologiques** pour la création de nouveaux visuels (langage TypeScript, bibliothèque D3.js, usage éventuel de React), en justifiant ces choix dans le contexte actuel. L'objectif est d'établir l'état de l'art des technologies de visualisation de données dans Power BI, tout en adoptant un regard critique sur leurs forces et faiblesses respectives.

## 2.1 Concepts BI et datavisualisation

La *Business Intelligence* (**BI**) peut se définir comme l'ensemble des méthodes et technologies visant à transformer des données brutes en connaissances utiles pour la prise de décision organisationnelle. Chen, Chiang et Storey (CHEN, CHIANG & STOREY, 2012) rappellent que la valeur de la BI réside moins dans l'acquisition massive de données que dans la capacité à les modéliser, les analyser et les représenter de manière intelligible pour l'humain. L'étape de visualisation constitue ainsi le dernier maillon du pipeline « ingestion–modélisation–analyse–présentation », mais elle s'avère décisive pour convertir des métriques abstraites en informations actionnables. C'est précisément à ce niveau que se situent les visuels Power BI, natifs ou personnalisés, objets d'étude du présent travail.

Dans le domaine de la *datavisualisation*, les sciences cognitives ont montré que l'œil humain perçoit rapidement certains attributs dits *préattentifs* : position, longueur, orientation ou couleur, entre autres. Ware (WARE, 2019) démontre que l'exploitation adéquate de ces attributs maximise la vitesse et la justesse de la lecture visuelle. Tufte (TUFTE, 1983) a, pour sa part, popularisé l'idée de *data–ink ratio*, soulignant que le graphisme ne doit conserver que l'encre strictement indispensable au message ; tout élément décoratif superflu — le *chart-junk* — nuit à la clarté. Few (FEW, 2009) prolonge cette perspective en montrant que la cohérence des encodages visuels (axes, couleurs, échelles) constitue une condition essentielle pour comparer de façon fiable plusieurs séries de données.

L'unification théorique de ces principes a été proposée par Wilkinson (WILKINSON, 2005) puis formalisée dans la communauté statistique par Wickham sous le nom de *Grammaire des graphiques*. Le modèle décrit chaque graphique comme la combinaison déclarative de couches : données, transformations, géométries, échelles, systèmes de coordonnées et facettage. Ce

cadre a influencé la plupart des bibliothèques modernes — notamment D3.js, Vega ou ggplot2 — et se retrouve implicitement dans l'API de Power BI ; chaque visuel y spécifie ses champs (data roles), ses encodages (capabilities) et son canevas de rendu.

Au-delà des principes, la BI professionnelle introduit des impératifs supplémentaires : performance d'affichage, accessibilité numérique, conformité réglementaire (RGPD) et gouvernance des artefacts analytiques. Les visuels standards de Power BI satisfont ces exigences pour des cas courants, mais ils se heurtent aux demandes spécifiques de certains métiers ; c'est autour de ces limites qu'émerge le besoin de visuels personnalisés. Comprendre les fondements de la BI et de la datavisualisation éclaire ainsi la double problématique du mémoire : confirmer la pertinence d'enrichir Power BI par de nouveaux composants, puis garantir que ces composants respectent les bonnes pratiques cognitives tout en s'intégrant dans un environnement d'entreprise contrôlé.

## 2.2 Architecture des visuels Power BI

Power BI est conçu autour d'une architecture de visualisation ouverte et extensible. Chaque élément visuel (graphique, carte, jauge, etc.) est rendu côté client à partir des données du modèle, via du code JavaScript/TypeScript exécuté dans Power BI Desktop ou dans le service web (MICROSOFT, 2015). Depuis 2015, Microsoft propose non seulement une panoplie de visuels « *core* » (natifs), mais permet aussi l'importation de visuels additionnels développés par la communauté ou des éditeurs tiers (MICROSOFT, 2016). Cette ouverture repose sur des standards web : *« en s'appuyant sur des standards ouverts d'Internet et des bibliothèques open-source comme D3.js »*, la création de visuels personnalisés a été grandement simplifiée (MICROSOFT, 2017). Microsoft publie d'ailleurs le code source de nombreux visuels natifs sur GitHub, attestant de sa volonté d'encourager un écosystème ouvert (MICROSOFT, 2024h). L'API publique est passée en version 5.10, puis en préversion 6 (22 juillet 2024), introduisant un DOM sécurisé et la *Rendering Events API*, ce qui renforce la cohérence entre ouverture et sécurité (MICROSOFT, 2024g).

### 2.2.1 Visual container et bac à sable

Qu'il soit natif ou personnalisé, un visuel s'insère dans le *canevas* du rapport et interagit avec le modèle de données via des rôles prédéfinis. Chaque visuel reçoit, du moteur Power BI, les données filtrées qui lui sont attribuées (colonnes, mesures, hiérarchies), puis exécute son propre code de rendu. Pour les visuels *custom*, ce code est empaqueté dans un fichier `.pbiviz` contenant les scripts, les styles et le manifeste (MICROSOFT, 2023). Power BI exécute alors le visuel dans un bac à sable sécurisé (*sandbox*) : une `iframe` isolée du reste du rapport (OKVIZ, 2022a). Le visuel n'accède ni aux autres visuels ni au modèle global ; il ne « voit » que les champs que l'utilisateur lui a explicitement liés. Depuis la mise à jour 2.140 (février 2024), tout visuel — qu'il soit privé (*organizational visual*) ou destiné à AppSource — est audité par l'option `pbiviz package -certification-audit` de *powerbi-visuals-tools* $\geq$ 6.1 : les appels réseau (`fetch`, `XMLHttpRequest`, WebSockets) et l'évaluation dynamique de code

(`eval`, `Function`) sont bloqués, et l'exécution est interrompue au-delà d'environ 120 s de CPU ou de 230 Mio de mémoire(MICROSOFT, 2025g). Ces garde-fous empêchent qu'un code malveillant puisse lire ou exfiltrer des données sans autorisation (**MediumSecurityPBI2023**).

### 2.2.2 Interactions et intégration

Malgré cet isolement technique, les visuels s'intègrent pleinement dans l'expérience interactive globale. Un visuel personnalisé correctement développé se comporte *exactement comme un visuel natif* : il réagit aux filtres, autorise le *cross-highlight* (mise en surbrillance croisée) et expose des options de mise en forme dans le panneau *Format* (MICROSOFT, 2024b). Lorsqu'un utilisateur clique, par exemple, sur une barre d'histogramme, le moteur Power BI propage l'événement de sélection aux autres visuels. Si le développeur a implémenté l'API `ISelectionManager`, son visuel peut émettre et recevoir ces événements ; il peut également recourir à `ITooltipService` pour les infobulles contextuelles ou à `ILocalizationManager` pour l'internationalisation, de sorte que l'intégration fonctionnelle et linguistique demeure uniforme (MICROSOFT, 2024c, 2024d).

La différence fondamentale reste donc interne : les visuels natifs font partie du produit et peuvent exploiter des API internes non exposées, tandis que les visuels personnalisés s'appuient uniquement sur l'API publique du SDK, avec les restrictions de sécurité détaillées en section 2.4. La section suivante examinera d'abord les capacités et limites de ces visuels natifs avant de traiter, en 2.3, l'approche Python/R, puis, en 2.4, le développement complet via le SDK.

## 2.3 Visuels Python / R : usages, atouts, limites

Outre les visuels préfabriqués, Power BI autorise l'exécution de scripts Python ou R pour produire des visuels sur mesure. L'utilisateur place un composant “Python visual” (ou “R visual”) dans le rapport, saisit son code dans l'éditeur, puis Power BI exécute ce script en tâche de fond : les données liées sont transmises sous forme de *dataframe* et le résultat retourné est une image statique (PNG) affichée dans le canevas.

Cette fonctionnalité exploite l'écosystème analytique des deux langages : bibliothèques *Matplotlib*, *Seaborn* ou *Plotly* côté Python ; *ggplot2* ou *plotly R* côté R. Un data-scientist peut ainsi tracer dans Power BI un nuage de points avec régression LOESS (R) ou un diagramme de réseau (Python) en quelques lignes, visuels impossibles à obtenir via les graphes natifs.

**Atouts.** La puissance réside dans la bibliothèque de packages open-source : statistiques avancées, machine learning, heatmaps, dendrogrammes, etc. Les scripts peuvent en outre pré-traiter les données (agrégation, calcul d'indicateurs, entraînement d'un modèle) avant de dessiner le graphique ; l'opération se relance automatiquement lorsque le visuel est rafraîchi, offrant au passage des capacités qu'un simple DAX ne couvre pas (analyse de texte, séries temporelles complexes).

**Limites techniques et fonctionnelles.** Le rendu reste une image statique : “les visualisations Python dans Power BI sont des bitmaps 72 DPI, sans interactivité directe” (MICROSOFT, 2025c). L’utilisateur ne peut donc pas cliquer dans le graphique pour filtrer les autres visuels ; seul un nouveau filtrage externe provoque la ré-exécution du script, qui demeure plus lente qu’un visuel natif.

Power BI encadre par ailleurs la quantité de données : *(i)* 150 000 lignes et 100 colonnes au maximum sont transmises au script ; au-delà, seules les premières 150 000 lignes sont prises en compte, un avertissement s’affichant sur l’image ; *(ii)* la taille totale ne doit pas excéder 250 MB en mémoire ; *(iii)* un *timeout* de cinq minutes interrompt tout script trop long(MICROSOFT, 2025c)*, mais ce délai est ramené à une minute lorsque l’exécution se fait dans le service Power BI* (MICROSOFT, 2025e) ; *(iv)* pour les *R visuals*, la taille du PNG de sortie est désormais plafonnée à 2 MB(MICROSOFT, 2025d). Ces garde-fous rendent les visuels Python/R adaptés à des échantillons ou à des agrégats, mais inadaptés à un traitement massif ou à un apprentissage profond.

D’autres restrictions s’appliquent : les chaînes de plus de 32 766 caractères sont tronquées ; les visuels ne s’affichent ni dans *Publish to Web* ni dans certains environnements mobiles pour raisons de sécurité ; **le runtime R 4.3.3 dans le service limite en outre le payload compressé à 30 MB**(MICROSOFT, 2025e) ; enfin, l’interpréteur Python ou R et les packages requis doivent être installés localement pour Power BI Desktop, tandis que le service en ligne n’exécute les scripts que sur un workspace Premium ou pour des comptes Pro, **;**ce qui signifie qu’un compte Free n’affichera pas ces visuels (MICROSOFT, 2025c).

**Sécurité et maintenance.** Lors de l’insertion d’un premier visuel Python / R, Power BI émet un avertissement invitant à exécuter uniquement un code de source fiable (MICROSOFT, 2025c). Dans un contexte d’entreprise, la gouvernance du code devient critique : le script est embarqué dans le fichier `.pbix`, hors de tout versionnage Git natif ; il faut donc organiser une validation et un cycle de mise à jour des dépendances.

**Bilan.** Les visuels Python et R constituent une solution d’appoint puissante pour des analyses ad hoc ou spécialisées, en comblant certaines lacunes des visuels natifs grâce à l’arsenal *data-science*. Ils doivent toutefois être utilisés en connaissance de leurs limites : image statique, performance conditionnée par les seuils de lignes et colonne, exigences Premium dans le service, et effort de maintenance du code. Dès qu’un visuel doit être interactif, largement partagé ou intégré au catalogue interne, le développement d’un visuel personnalisé via le SDK — thème de la section 2.4 — devient souvent la voie la plus pérenne.

## 2.4 SDK Custom Visuals : principes, sécurité, pipeline

Lorsque les visuels natifs ne suffisent pas et qu'un script Python/R est trop limité, la solution la plus aboutie est de créer un visuel personnalisé complet en utilisant le Software Development Kit (SDK) de Power BI. Microsoft fournit un SDK officiel (sous forme de Node Package `powerbi-visuals-tools` **(v 6.3, mai 2025)**) qui permet aux développeurs de coder, tester et empaqueter de nouveaux visuels sous forme d'un fichier `.pbiviz`(MICROSOFT, 2025i).

Le principe fondamental est qu'un visuel Power BI n'est rien d'autre qu'une application web encapsulée : il contient du code HTML/JavaScript (plus précisément TypeScript, transcompilé en JavaScript) qui reçoit des données et génère du DOM (SVG, Canvas, etc.) pour afficher un graphique. Le SDK abstrait la communication avec Power BI : le développeur définit, via un fichier `capabilities.json`, quels champs et paramètres de formatage son visuel acceptera, puis implémente une classe TypeScript qui reprend l'interface `IVisual`. Cette classe comporte notamment une méthode clé `update(options)` qui est appelée par Power BI à chaque rafraîchissement des données ou interaction, fournissant les données filtrées à représenter. Le code du développeur doit alors traduire ces données en éléments visuels, en utilisant éventuellement des bibliothèques de visualisation comme D3.js (voir section 2.5).

### Cycle de développement (pipeline)

Le développement d'un custom visual suit généralement ce pipeline :

1. Initialisation du projet via l'outil en ligne de commande (CLI) du SDK – par ex. `pbiviz new MyVisual` génère une structure de projet avec les fichiers de base (manifest, capabilities, code TS, etc.)(MICROSOFT, 2024a).

2. Développement et test en local : le SDK permet de lancer un serveur local et d'attacher le visuel en développement à Power BI Desktop (en mode Developer). On peut ainsi tester le visuel dans un rapport en temps réel pendant qu'on code.

3. Compilation et packaging : une fois le développement achevé, la commande `pbiviz package -certification-audit` produit le fichier `.pbiviz` final et exécute un contrôle automatique des appels réseau(MICROSOFT, 2025j).

4. Distribution : ce package peut être importé manuellement dans n'importe quel rapport Power BI (option *Importer un visuel à partir d'un fichier*), ou déployé de façon plus gérée. Pour une utilisation interne à une organisation, l'administrateur tenant peut le publier dans le magasin organisationnel de Power BI(MICROSOFT, 2025b).

Pour une distribution publique à l'écosystème Power BI, le développeur peut soumettre son visuel à AppSource, la place de marché Microsoft. Une étape de certification est alors nécessaire (voir plus loin). Microsoft indique que « n'importe quel développeur web peut créer un visuel

personnalisé et le packager en un fichier .pbiviz unique à importer dans Power BI »(MICROSOFT, 2024e), et propose même un concours et une galerie communautaire pour encourager ces contributions(MICROSOFT, 2025h).

Depuis l'introduction des custom visuals, le catalogue AppSource s'est ainsi énormément étoffé : on y recense aujourd'hui plusieurs centaines de visuels disponibles (gratuits ou commerciaux) couvrant des usages variés (graphes, chronologies, infographies, etc.).

### Capacités et restrictions

Un visuel personnalisé bien conçu permet d'aller au-delà des limitations des visuels natifs. On peut par exemple introduire de nouveaux types de graphiques, des interactions innovantes, ou des designs sur mesure. Cependant, il est important de souligner que les visuels custom opèrent dans un cadre réglementé par Microsoft pour assurer la sécurité et la performance. D'une part, comme mentionné, ils tournent dans un *sandbox* isolé. Cela a plusieurs implications :

(a) le visuel ne peut pas accéder aux données du modèle qui ne lui ont pas été explicitement fournies via les champs liés par l'utilisateur(OKVIZ, 2022b) ;

(b) il ne peut pas non plus lire ou modifier l'état d'un autre visuel ou élément du rapport (pas d'accès au DOM global ou aux variables globales de l'hôte) ;

(c) le visuel ne peut dessiner qu'à l'intérieur de sa surface allouée : il lui est interdit, par exemple, de faire apparaître une fenêtre pop-up ou un élément HTML en dehors de son cadre *bounding box*(OKVIZ, 2023).

**Pas de communication externe non approuvée.** Par mesure de sécurité, les visuels custom n'ont pas le droit d'envoyer des données vers un service externe ou d'en charger, sauf approbation explicite. Les règles de certification 2025 interdisent toute requête sortante pour un visuel certifié(MICROSOFT, 2025b). Même pour les visuels non certifiés, le sandbox applique une Content Security Policy très stricte (`default-src 'none'`) documentée dans les notes de version API v 6(MICROSOFT, 2025a).

### Certification et confiance

Microsoft a mis en place un programme de certification des visuels. Un visuel certifié est un visuel custom qui a passé avec succès un processus de validation par Microsoft, incluant des vérifications de sécurité (absence de code malveillant ou de fuite de données), de performance et de conformité aux bonnes pratiques(MICROSOFT, 2025b). Les visuels certifiés sont signalés par une icône spéciale et certaines fonctionnalités de Power BI (export PDF/PPT, abonnements email) ne rendent que ces visuels(OKVIZ, 2024).

### Synthèse

En résumé, le développement de visuels custom via le SDK offre une flexibilité maximale pour répondre à des besoins spécifiques, au prix d'un effort de développement et du respect de contraintes de sécurité. Dans la section suivante, nous discutons des choix technologiques concrets (TypeScript, D3.js et éventuellement React) pour implémenter un visuel custom.

## 2.5 Choix technologiques (TypeScript, D3, React optionnel)

Le développement d'un visuel personnalisé Power BI repose sur un *stack* web moderne articulé autour de trois briques : TypeScript pour le langage, D3.js pour le rendu vectoriel et, à titre optionnel, React pour la structuration de l'interface. Ce choix résulte d'une analyse des alternatives en termes de maintenabilité, performance, sécurité et accessibilité.

**TypeScript vs JavaScript.** Le SDK Power BI est conçu nativement pour TypeScript, surensemble typé de JavaScript que Microsoft recommande pour les visuels personnalisés (MICROSOFT, 2025f). Le typage statique détecte précocement les incohérences et réduit les bogues en production : une étude empirique portant sur plus de 400 projets GitHub montre une diminution moyenne de 15 % des défauts après migration vers TypeScript (BEYER, FARIA & ZIMMERMANN, 2023). Les annotations rendent le code plus explicite, facilitant lecture, revue et refactorisation. Par ailleurs, TypeScript apporte des abstractions modernes — interfaces, classes, *generics* — qui encouragent une architecture modulaire et extensible. Le code est ensuite transcompilé en JavaScript ES 2019, sans impact mesurable sur les performances d'exécution (ECMA INTERNATIONAL, 2024). Ne pas passer par cette couche (écrire directement en JavaScript ES6+) aurait simplifié la phase de build, mais au prix d'une dette technique accrue et d'un risque de régression plus élevé, notamment pour un composant destiné à évoluer avec l'API Power BI.

**D3.js pour le rendu SVG.** D3 — « Data-Driven Documents » — est la librairie de référence pour manipuler le DOM SVG et créer des visualisations sur mesure. Elle établit un lien direct entre données et éléments graphiques, autorisant des transformations déclaratives efficientes (BOSTOCK, 2019). Cette approche bas niveau donne un contrôle total sur chaque attribut visuel (couleur, position, animation) : condition nécessaire pour implémenter un graphique non standard conforme aux exigences métier. D3 offre en outre un large éventail de modules (générateurs de formes, projections cartographiques, échelles, *layouts* hiérarchiques) et bénéficie d'un écosystème mature d'exemples réutilisables. Les alternatives « haut niveau » (Chart.js, Plotly) accélèrent le prototypage mais atteignent vite leurs limites lorsqu'il s'agit d'un design original ; quant au *canvas* ou à WebGL, ils complexifient l'accessibilité et la netteté du rendu zoomé. Le choix du SVG produit par D3 facilite la mise à l'échelle et l'ajout d'attributs ARIA ou de balises `<title>`, répondant aux recommandations WCAG 2.2 pour les rapports Power BI (W3C, 2023). En pratique, les tests préliminaires sur un échantillon de 50 000 points affichés montrent des temps de rendu inférieurs à 250 ms sur un poste standard, confirmant la pertinence de D3 dans ce contexte (DUPONT & NGUYEN, 2025).

**React (optionnel) pour l'UI.** React n'est pas requis par le SDK, mais devient pertinent dès lors que le visuel embarque une interface utilisateur complexe : sélecteurs, menus contextuels, légende cliquable. Sa philosophie *component-based* et son *virtual DOM* optimisent les mises à jour d'interface en réduisant les re-rendus coûteux (META OPEN SOURCE, 2024). L'association « React pilote la structure, D3 gère les calculs et applique les transformations » est désormais un *pattern* reconnu ; plusieurs visuels open-source l'utilisent déjà dans AppSource (MICROSOFT, 2024f). L'empreinte ajoutée ($\approx$ 40 kB minifiés) reste compatible avec la limite de 2 Mo du package `pbiviz`. Pour les projets très légers, on peut encore préférer Preact, clone allégé compatible avec l'API React. Le coût cognitif — JSX, gestion d'état — est maîtrisé par l'équipe et amorti par la facilité de test unitaire des composants. Si le visuel n'exige qu'un rendu statique ou des animations D3 simples, il est cohérent de se passer de React ; c'est pourquoi l'usage reste qualifié d'« optionnel ».

**Synthèse.** Le triptyque TypeScript + D3 (+ React) offre un compromis robuste : rigueur logicielle, expressivité graphique, performances maîtrisées et accessibilité native. Il s'inscrit dans les standards de l'écosystème Power BI, maximise la réutilisabilité du savoir-faire front-end de l'équipe et minimise la dette technique à long terme. Les alternatives évaluées (JavaScript pur, bibliothèques charting « tout-en-un », canvas/WebGL) ont été jugées moins compatibles avec les objectifs de maintenance, de qualité et de gouvernance fixés dans le présent travail.

## 2.6 Solutions concurrentes (Tableau, Qlik, Looker)

Les principales plateformes BI concurrentes — Tableau, Qlik Sense et Looker — proposent chacune des mécanismes de personnalisation de visuels qui diffèrent de ceux de Power BI, tant sur le plan technique que sur les implications métier. Comprendre ces approches permet de situer les *visuels custom* Power BI dans un paysage concurrentiel plus large.

**Tableau.** Tableau étend ses capacités via les *Tableau Extensions*, des applications Web encapsulées dans les tableaux de bord grâce à l'Extension API(TABLEAU SOFTWARE, 2024b). Un développeur crée, en JavaScript, un composant (souvent dans une `iframe`) qui ajoute un graphique personnalisé ou une interaction spécifique, par exemple du *write-back* ou l'intégration d'un service tiers(TABLEAU SOFTWARE, 2024a). Depuis la v2019.4, les extensions sont exécutées par défaut en *sandbox*, sans accès réseau, et toute extension nécessitant un accès externe doit être explicitement inscrite sur liste blanche par l'administrateur serveur ou site(TABLEAU SOFTWARE, 2025a). Tableau n'assure pas le support technique des extensions ; cette responsabilité incombe à leur éditeur, si bien que les entreprises doivent évaluer la fiabilité de la source avant déploiement. Sur le plan financier, la licence Creator avoisine 70 USD utilisateur/mois(TABLEAU SOFTWARE, 2025b), ce qui fait de la personnalisation une option coûteuse pour les organisations sensibles au budget, à la différence d'un environnement Power BI où le coût d'entrée est plus faible.

**Qlik Sense.** L'extensibilité de Qlik repose sur les *Visualization Extensions* : objets personnalisés écrits en HTML 5 / JavaScript / CSS et déclarés par un manifeste `.qext`(QLIK, 2024a). Une extension Qlik, lorsqu'elle respecte l'API Qlik, s'intègre comme un objet natif : on la fait glisser dans la feuille, on ajuste ses propriétés, et elle réagit aux sélections grâce au moteur associatif(QLIK, 2024b). Le développeur peut embarquer n'importe quelle bibliothèque (D3, three.js, etc.) pour façonner un rendu spécifique. Qlik privilégie ici une approche communautaire : de nombreuses extensions sont partagées via *Qlik Branch* sans validation officielle. L'administrateur doit donc examiner puis installer manuellement l'extension sur le serveur ou le tenant SaaS avant qu'elle soit disponible aux créateurs de contenu. Commercialement, Qlik Sense Business se situe autour de 30 USD par utilisateur/mois, et l'édition Enterprise encore au-dessus (QLIK, 2025). Les clients attendent donc qu'une capacité de personnalisation avancée soit incluse, mais ils doivent disposer en interne de compétences front-end JS pour en profiter pleinement.

**Looker.** Looker (aujourd'hui composant de Google Cloud) autorise des visuels personnalisés via le *Looker Custom Visualization SDK*. Le développeur code en JavaScript un composant qui implémente la fonction `updateAsync` afin de recevoir les données d'une *explore* Looker et de restituer un rendu SVG ou Canvas (GOOGLE CLOUD, 2025). Pour diffusion publique, Google impose un processus Marketplace : le code doit être hébergé sur GitHub et passer une revue de conformité avant publication(GOOGLE CLOUD, 2024). Il est toutefois possible de limiter le visuel à un usage interne en l'ajoutant directement via l'interface Admin. La philosophie lookerienne est donc plus centralisée : la personnalisation est permise, mais sous contrôle étroit. La tarification, négociée au cas par cas (souvent nettement au-delà des niveaux Power BI/Tableau), signifie que les clients Looker visent généralement une valeur élevée tirée soit du modèle LookML, soit d'un nombre réduit, mais hautement spécifique, de visuels custom.

**Lecture comparative.** Tableau et Qlik offrent une liberté initiale plus large — l'utilisateur avancé peut charger une extension localement — mais délèguent la gouvernance au client ; Power BI et Looker exigent un contrôle centralisé (certification ou validation administrateur) avant déploiement massif. Techniquement, toutes s'appuient sur le triptyque HTML/JS/CSS, mais Power BI se distingue par son SDK TypeScript/D3 structuré et sa marketplace AppSource très fournie, facteur d'adoption rapide. Sur le plan économique, Power BI reste l'option la plus abordable pour déployer des visuels custom à grande échelle, alors que Tableau, Qlik Sense et surtout Looker réservent la personnalisation intensive à des environnements dont le budget justifie l'investissement.

## 2.7 Synthèse des écarts & opportunités

À l'issue de cette analyse, il est possible de synthétiser les forces et limites des différentes approches de visualisation de données dans Power BI et ses alternatives, puis d'en déduire les cas d'usage privilégiés. Quatre grandes catégories de solutions ont été examinées :

1. les visuels natifs de Power BI ;

2. les visuels basés sur des scripts R / Python intégrés à Power BI ;
3. les solutions concurrentes (extensions Tableau, Qlik Sense, visuels Looker) ;
4. les visuels personnalisés développés via le SDK Power BI.

Chaque option présente des atouts spécifiques mais aussi des lacunes fonctionnelles, dont la compréhension permet d'identifier des opportunités d'amélioration ou d'utilisation optimale.

**Visuels natifs Power BI.** Les visuels fournis d'office constituent le socle de la plupart des rapports. Ils sont clefs en main, supportés par Microsoft, optimisés pour des performances élevées et parfaitement intégrés (sélections croisées, filtres, export PDF/PPT, affichage mobile). Leur fiabilité et leur sécurité sont maximales puisqu'aucun code utilisateur n'est exécuté. Leur limitation : une relative rigidité. Dès que l'on vise une représentation moins courante — diagramme de Sankey, bullet chart spécifique, réseau, alluvial, cartographie indoor — l'utilisateur reste tributaire des fonctionnalités prévues. Certains visuels natifs souffrent également de restrictions (impossibilité d'ajouter un second axe dans certaines combinaisons, personnalisation limitée des étiquettes, etc.). Ainsi, les visuels natifs excellent dans les usages courants avec un effort zéro, mais offrent peu de latitude pour répondre aux demandes hors norme.

**Visuels Python / R.** L'exécution d'un script Python ou R à l'intérieur de Power BI ouvre la porte à l'immense écosystème de visualisation de ces deux langages (matplotlib, seaborn, ggplot2, plotly offline, folium, networkx, etc.). Cet atout est la flexibilité : tout graphique réalisable dans Python ou R peut, en théorie, être intégré dans un rapport Power BI, ce qui est particulièrement utile pour des analyses statistiques avancées ou des bibliothèques très spécialisées. Limites : – rendu *statique* (image PNG 72 DPI) : aucune interaction directe ; – plafond de 150 000 lignes transmises au script ; – liste de packages autorisés restreinte dans Power BI Service ; – exécution plus lente qu'un visuel natif (script relancé à chaque rafraîchissement) ; – public restreint aux profils sachant coder. Les visuels R / Python sont donc idéaux pour un prototypage rapide ou l'exploration ad hoc, mais peu adaptés à un déploiement massif et interactif : ils servent souvent de tremplin vers un futur visuel SDK.

**Extensions Tableau / Qlik / Looker.** Tableau se distingue par une bibliothèque native déjà riche ; néanmoins, les *Tableau Extensions* permettent d'incorporer des applications web JS (dans une `iframe`) pour ajouter, par exemple, du *write-back* ou un composant interactif inédit. Depuis 2019.4, ces extensions tournent par défaut en sandbox sans accès réseau ; un administrateur doit inscrire sur liste blanche les extensions sortantes. Qlik Sense propose des *Visualization Extensions* qui se comportent comme des objets natifs ; elles participent aux sélections associatives et peuvent être partagées via la communauté Qlik Branch. L'entreprise doit toutefois valider le code avant déploiement sur le serveur. Looker, plus récent, autorise des visuels JS via le *Custom Visualization SDK*. Pour une diffusion Marketplace, Google exige une revue GitHub,

ce qui garantit qualité et sécurité mais freine la spontanéité. Toutes ces plateformes prouvent que l'extensibilité par code est devenue un incontournable ; cependant, elles entraînent un effort de développement et des coûts de licence souvent supérieurs à Power BI.

**Visuels SDK Power BI.** La solution la plus puissante et flexible dans l'écosystème Power BI repose sur le SDK (TypeScript, D3, éventuellement React). Elle permet de créer un composant répondant exactement aux spécifications métier, avec une intégration complète (filtres, mobile, export) et une possible diffusion AppSource ou Organizational Visuals. Contreparties : effort de développement élevé (semaines), maintenance continue (veille SDK, correctifs, adaptation thèmes) et besoin de gouvernance du code. La certification Microsoft atténue les risques en cas de diffusion publique.

**Vue d'ensemble comparative.**

| Critère | Natifs PBI | Python/R | SDK PBI | Ext. Tableau | Ext. Qlik | Visuels Looker |
|---|---|---|---|---|---|---|
| Interactivité | Excellente, intégrée | Statique, rafraîchie | Équivalente aux natifs | Bonne, API Tableau | Native, moteur associatif | Bonne, limitée au visuel |
| Extensibilité | Type figé | Très large graphique | Totale (TS/D3) | Large (widget web) | Large (JS libre) | Moyenne (API Looker) |
| Effort dev. | Nul | Faible–modéré | Élevé | Élevé | Élevé | Élevé |
| Maintenance | Microsoft | Auteur script | Équipe dev. | Auteur ext. | Auteur / communauté | Auteur / Google |
| Sécurité | Maximale | Sandbox packages | Sandbox + signature | Sandbox réseau | Contrôle admin | Marketplace / admin |

TABLE 2.1 *: Comparaison synthétique des approches de personnalisation*

**Conclusion.** Chaque approche comble des écarts laissés par les autres. Les visuels natifs offrent robustesse et simplicité, mais limitent la créativité. Les scripts R / Python ouvrent la voie à l'innovation visuelle rapide au prix de l'interactivité. Les solutions concurrentes montrent que l'ensemble du marché reconnaît l'importance de la personnalisation. Le SDK Power BI, soutenu par un écosystème open-source foisonnant, transforme ces écarts en opportunités : répondre à des besoins métier spécifiques, enrichir l'offre AppSource, ou différencier les rapports par un design inédit. Pour Ecrins SA, investir dans la maîtrise des visuels SDK apparaît donc comme une réponse stratégique aux limites identifiées, à condition de mettre en place une gouvernance technique et sécuritaire adaptée — ce à quoi le chapitre **??** est consacré.

# 3 | Contexte du pilote, méthodologie & organisation

## 3.1 Origine du besoin

Dans le cadre de son activité de conseil, la société ECRINS SA étudie la *faisabilité* de visuels Power BI dépassant les limites du catalogue natif et des extensions AppSource. Deux scénarios métiers proposés par des clients—l'un marketing, l'autre analytique—ont été retenus afin de démontrer, sous forme de **preuve de concept (PoC)**, la viabilité technique de visuels personnalisés susceptibles d'être industrialisés par la suite.

### 3.1.1 Besoin marketing : carte des flux passagers aéroportuaire

Un client gestionnaire d'un aéroport international souhaite **cartographier les flux de passagers sur l'ensemble du site** (parkings, halls d'enregistrement, points de contrôle, boutiques, portes d'embarquement) pour identifier les zones de plus forte affluence et optimiser l'emplacement des surfaces publicitaires. Le visuel attendu adopte le paradigme de la *flow map* : des arcs reliant les points d'intérêt, dont l'épaisseur matérialise le volume de voyageurs, révèlent immédiatement les axes de déplacement dominants (**GuoFlowMaps2011**). Une couche de *heat map* complétera la lecture en soulignant les « hot-spots » commerciaux. Les choix techniques (plan SVG, coordonnées, routage A*, calcul de densité) seront détaillés au chapitre **??**.

### 3.1.2 Besoin d'analyse hiérarchique : Sunburst de décomposition

Un autre client d'ECRINS SA exige une visualisation capable d'explorer *n'importe quelle hiérarchie de données* (structure organisationnelle, catalogue produits, processus métier, etc.). Le *Decomposition Tree* natif de Power BI, limité en personnalisation et en nombre de mesures, ne répond pas à cette exigence. Un *diagramme Sunburst* a donc été retenu : son organisation radiale affiche plusieurs niveaux en une seule vue et compare visuellement les contributions relatives de chaque branche (**StaskoSunburst2000**). L'objectif est de disposer d'un composant **personnalisable** et **réutilisable** pour tout scénario d'analyse hiérarchique ; les aspects d'implémentation (partition D3, drill, accessibilité) seront présentés également au chapitre **??**.

## 3.2 Critères de réussite

Les critères de réussite du prototype servent d'indicateurs de *faisabilité technique* et de *rigueur méthodologique*, plutôt que d'objectifs fonctionnels exhaustifs. Ils visent à vérifier que la preuve de concept respecte un socle minimal en matière de performance, d'ergonomie et de qualité.

1) **Performance** : le temps de rendu et de mise à jour doit rester inférieur à 300 ms, garantissant une interaction fluide pour l'utilisateur.

2) **Accessibilité (WCAG 2.2)** : le visuel doit se conformer aux recommandations WCAG 2.2 (contraste suffisant, navigation clavier, attributs `aria` pertinents) afin d'être utilisable par le plus grand nombre.
3) **Couverture fonctionnelle** : toutes les fonctionnalités attendues (détails au survol, sélection et filtrage, zoom, etc.) sont implémentées, assurant la pertinence métier du prototype.
4) **Intégration Power BI (cross-highlighting)** : le visuel prend en charge le filtrage et le surlignement croisés avec les autres visualisations du rapport, conformément au comportement natif de la plateforme.
5) **Qualité du code** : le code source est modulaire, documenté et couvert par des tests unitaires ($\geq$ 75 %), facilitant la maintenance.
6) **Sécurité (sandbox pbiviz)** : le prototype s'exécute dans l'*iframe* sandbox sans appel réseau externe ni usage d'API non autorisées, passant l'audit `pbiviz package -certification-audit`.
7) **Documentation** : un guide utilisateur et un guide technique détaillent l'installation, l'usage et l'architecture interne du visuel.

TABLE 3.1 *: Synthèse des critères de réussite*

| **Axe** | **Critère cible** |
|---|---|
| Performance | Temps de réponse du visuel < 300 ms |
| Accessibilité | Conformité complète aux WCAG 2.2 |
| Couverture fonctionnelle | Fonctionnalités attendues implémentées |
| Intégration Power BI | Cross-highlighting pleinement opérationnel |
| Qualité du code | Code modulaire, tests $\geq$ 75 %, commentaires |
| Sécurité | Exécution conforme aux règles du sandbox |
| Documentation | Guides utilisateur et développeur fournis |

Chaque critère énuméré ci-dessus constituera un point de contrôle lors de l'évaluation finale : leur satisfaction attestera de la faisabilité technique et de la qualité du travail réalisé dans le cadre de ce *proof of concept*.

## 3.3 Plan de tests (datasets démo, scénarios tableau de bord)

La phase de tests s'appuie sur des jeux de données de démonstration destinés à valider les fonctionnalités attendues. Ces jeux de données, bien qu'évolutifs, sont conçus pour couvrir l'ensemble des cas d'usage clés. Les paragraphes suivants détaillent pour chacun des deux

visuels (Carte de flux de passagers et Sunburst hiérarchique) la structure du dataset démo, les scénarios d'usage simulés et la manière dont ces scénarios valident la faisabilité fonctionnelle du visuel.

### 3.3.1 Carte de flux de passagers (Passenger - Flow Map)

**Structure du jeu de données :** Le dataset contient notamment, pour chaque segment de flux, l'identifiant du flux (EdgeId) reliant un point de départ à un point d'arrivée, les coordonnées de ces deux points (X1, Y1 pour le point d'origine et X2, Y2 pour la destination) en pixels, ainsi que le nombre de passagers (Passages), l'heure du passage (Heure) et le type de flux (par exemple « Départ » ou « Arrivée »). Ces champs permettent de cartographier spatialement les mouvements de passagers au sein de l'aéroport.

**Cas d'usage simulés :** Les cas d'utilisation incluent notamment la sélection par tranches horaires (filtrage sur le champ Heure), la distinction du type de flux (Départ/Arrivée) et l'activation d'un mode d'affichage en « heatmap » pour repérer les zones de forte affluence. Les interactions croisées sont également testées : par exemple, la sélection d'une tranche horaire ou d'une porte d'embarquement dans un autre graphique doit filtrer dynamiquement la carte de flux (et inversement). Par ailleurs, la visualisation affiche des info-bulles présentant les valeurs agrégées (par ex. nombre de passagers), qui se mettent à jour en fonction des filtres appliqués.

**Validation de la faisabilité fonctionnelle :** Ces scénarios confirment que le visuel réagit aux besoins métiers et techniques. Par exemple, l'application d'un filtre temporel permet de vérifier que le nombre de passagers représentés s'ajuste et que les info-bulles correspondent aux données filtrées *learn.microsoft.com*. L'activation du mode heatmap démontre la capacité du visuel à convertir les flux en zones d'intensité. Enfin, les tests d'interaction croisées valident que la carte de flux répond correctement aux sélections dans d'autres visuels, en filtrant les données selon les sélections effectuées, comme attendu *learn.microsoft.com*.

### 3.3.2 Sunburst hiérarchique (Analyse multi-niveaux)

**Structure du jeu de données :** Ce jeu de données de démonstration représente une hiérarchie multi-niveaux typique (p. ex. structure organisationnelle ou répartition budgétaire). Il contient en général plusieurs colonnes de catégories hiérarchiques (au moins deux niveaux, par ex. Département puis Sous-département), ainsi qu'une colonne de mesure (par exemple un montant). Chaque ligne associe ainsi un compte (p. ex. Marketing > Digital > SEO) à sa valeur numérique correspondante. Cette structure multi-niveaux est adaptée à un diagramme en rayons de soleil, chaque anneau correspondant à un niveau de la hiérarchie *datavizcatalogue.com*.

**Cas d'usage simulés :** Les tests incluent la sélection de différents niveaux hiérarchiques (par exemple filtrer un département), ainsi que des opérations de forage (drill-down et drill-up) pour naviguer entre niveaux généraux et détaillés. La mise en évidence de branches (sous-ensembles de catégories) est également simulée, ainsi que l'application de filtres numériques (pour ne

conserver que les valeurs supérieures à un seuil). De plus, l'interaction croisée est vérifiée : sélectionner un segment du sunburst (par ex. un département particulier) filtre en conséquence les autres visuels du rapport, et inversement.

**Validation de la faisabilité fonctionnelle :** La réussite de ces tests confirme que le Sunburst peut représenter fidèlement la hiérarchie et ses agrégations. Le test de forage garantit que le passage d'un niveau à l'autre s'effectue correctement et que les proportions sont mises à jour. Les filtres appliqués (catégoriels ou numériques) vérifient que le visuel réagit comme prévu, en ajustant les segments affichés. Enfin, les tests d'interaction croisées montrent que la communication entre le Sunburst et les autres éléments du rapport est opérationnelle, conformément aux pratiques standards de filtrage Power BI *learn.microsoft.com*.

## 3.4 Organisation du travail — cycle *Waterfall* en cinq phases

Le projet suit un cycle de développement structuré de type *Waterfall*, organisé en cinq phases successives couvrant la période du 12 mai au 14 août 2025. Ce modèle en cascade, bien que linéaire, offre une clarté dans le déroulement des étapes, permet de fixer des objectifs intermédiaires précis et facilite la validation progressive des livrables, en cohérence avec le cadre académique.

TABLE 3.2 *: Découpage du projet selon un cycle Waterfall*

| **Phase** | Période | Objectif principal |
|---|---|---|
| **Analyse & cadrage** | 12 mai – 31 mai | Formaliser les besoins, les contraintes Power BI, et produire le cahier des charges fonctionnel. |
| **Conception détaillée** | 1 juin – 14 juin | Définir l'architecture des visuels, produire les maquettes, planifier les tests et établir le dossier de conception. |
| **Implémentation** | 15 juin – 19 juillet | Développer les deux visuels (PassengerFlow Map, Sunburst) et les intégrer dans un rapport Power BI. |
| **Vérifications & optimisation** | 20 juillet – 4 août | Tester les composants (fonctionnalité, performance, accessibilité), corriger les anomalies et optimiser les performances. |
| **Livraison finale** | 5 août – 14 août | Rédiger la documentation, finaliser le packaging, préparer le rapport et la soutenance. |

La première phase, prévue entre le 12 et le 31 mai, est consacrée à l'analyse du besoin et au cadrage fonctionnel. Elle a pour but de formaliser les attentes des clients d'ECRINS SA, d'identifier les limites des visuels existants et de poser les fondations du projet à travers un cahier des charges. La seconde phase, qui s'étend du 1er au 14 juin, consiste à traduire ces exigences en une conception détaillée : choix techniques, structure des données, maquettes visuelles, et architecture du composant. Ces éléments constitueront le dossier de conception, servant de référence pour le développement.

La phase suivante, qui s'étale du 15 juin au 19 juillet, est dédiée à l'implémentation proprement dite des deux visuels (Passenger-Flow Map et Sunburst hiérarchique). Durant cette période, les premières versions fonctionnelles seront produites et intégrées dans un rapport Power BI de démonstration. Viendra ensuite la phase de tests et d'optimisation, entre le 20 juillet et le 4 août, qui vise à valider la conformité des composants par rapport aux critères de performance, d'accessibilité et d'intégration attendus. Les tests automatisés et les ajustements finaux permettront de consolider la qualité des livrables. Enfin, la période du 5 au 14 août est réservée à la livraison finale : rédaction de la documentation utilisateur et technique, préparation du guide d'intégration et formalisation du rapport académique.

Le pilotage du projet s'appuie sur un mode de gestion individuel allégé. Aucun outil de gestion de projet formel n'est imposé, mais un ensemble cohérent de pratiques permet d'en assurer le suivi rigoureux. Un journal de bord personnel consigne les tâches réalisées, les choix techniques, les obstacles rencontrés et les décisions prises. Ce journal, versionné via Git, constitue également une base factuelle pour le chapitre d'évaluation.

L'avancement est visualisé à l'aide d'un tableau de type *Kanban* dans GitHub Projects, organisé en trois colonnes (*À faire*, *En cours*, *Terminé*). Cette méthode offre une vue claire de la charge de travail et permet de prioriser les actions chaque semaine. Par ailleurs, des réunions de suivi hebdomadaires sont organisées avec le professeur référent. Elles servent à valider l'état d'avancement, à ajuster les priorités si nécessaire et à arbitrer d'éventuelles décisions techniques.

La qualité du livrable est suivie à trois niveaux. D'abord, la couverture des tests unitaires est mesurée tout au long du développement, avec un objectif minimal de 75 %. Ensuite, l'accessibilité est vérifiée selon les recommandations WCAG 2.2 à l'aide d'outils automatisés tels que `axe-core`. Enfin, un audit de conformité est réalisé en amont de chaque livraison majeure à l'aide de la commande `pbiviz package -certification-audit`, qui garantit le respect des exigences de sécurité et d'isolation imposées par Power BI.

Plusieurs risques ont été identifiés en amont. Une dérive du planning est anticipée par l'ajout de marges de sécurité entre certaines phases, notamment entre l'implémentation et la phase de test. Les blocages techniques liés au SDK Power BI feront l'objet d'un traitement rapide par l'analyse de la documentation officielle et le recours aux communautés techniques ; à défaut,

des simplifications fonctionnelles seront envisagées. Enfin, si les jeux de données venaient à évoluer au cours du projet, un format pivot standardisé (fichier CSV structuré + dictionnaire de schéma) a été défini pour garantir la compatibilité avec le code développé.

En résumé, l'organisation du travail repose sur un équilibre entre structure formelle (phases, jalons, livrables) et agilité opérationnelle (journal de bord, révisions hebdomadaires, tableaux de tâches). Cette méthode garantit la qualité du livrable final tout en respectant les contraintes de temps et les exigences pédagogiques du travail de Bachelor.

## 3.5 Sources de validation (revue experte et mini-test utilisateur)

Dans la mesure où ce travail constitue avant tout une *proof of concept*, l'objectif est de démontrer la faisabilité technique et la pertinence métier des visuels, plutôt que de produire une évaluation statistiquement généralisable. Selon Mohagheghi et Dehlen, un PoC vise essentiellement à réduire l'incertitude d'un projet en vérifiant, sur un périmètre réduit, que la solution envisagée répond aux attentes clés(**MohagheghiPoC2008**). La démarche de validation retenue est donc qualitative et pragmatique : elle combine une revue experte fonctionnelle et un mini-test utilisateur exploratoire.

**Revue experte fonctionnelle.** La professeure référente, également active au sein d'ECRINS SA, évaluera les visuels sur la base d'un jeu de scénarios représentatifs : filtrage horaire et heat-map pour la carte de flux, navigation drill-down pour le Sunburst, cohérence des libellés et des agrégations. Cette revue donnera lieu à des commentaires formalisés, consignés dans le journal de bord, puis, le cas échéant, à des ajustements fonctionnels avant la démonstration interne.

**Démonstration interne.** En fin de développement, une courte présentation sera organisée devant un collaborateur d'ECRINS SA (chef de projet ou responsable métier). L'objectif est de recueillir un retour immédiat sur l'*adhérence métier* : la carte met-elle bien en évidence les axes de transit attendus ? Le Sunburst restitue-t-il correctement la hiérarchie analysée ? Ce retour oral, même informel, constitue une validation qualitative de la pertinence pratique du prototype.

**Mini-test utilisateur exploratoire.** Enfin, un test utilisateur *exploratoire* sera conduit avec un utilisateur unique. Conformément aux recommandations de Lallemand et Gronier, un tel protocole vise à observer les réactions et la compréhension sans prétention de représentativité statistique (**LallemandUX2016**). Le participant sera invité à réaliser deux actions : (1) identifier la zone la plus dense sur la carte de flux et (2) interpréter la part d'une catégorie donnée dans le Sunburst. Les observations et commentaires recueillis permettront de détecter d'éventuels points d'incompréhension ou de friction.

**Limites assumées.** Cette combinaison revue-expert / démonstration / test exploratoire ne fournit pas de preuve quantitative ; elle répond néanmoins aux exigences d'un PoC : confirmer la faisabilité, vérifier l'utilité métier et recueillir des pistes d'amélioration avant toute industrialisation future.

# 4 Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

# I An appendix

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

# Références


AMYROTOS, G., VASSILIADIS, S. & KARACAPILIDIS, N. (2024). Personalization in Business Intelligence and Analytics Systems: A State-of-the-Art Review and Conceptualization. *Decision Support Systems*, *170*, 114122. doi :10.1016/j.dss.2024.114122

BEYER, K., FARIA, L. & ZIMMERMANN, T. (2023). An Empirical Study of TypeScript Adoption and Its Impact on JavaScript Projects. *Empirical Software Engineering*, *28*(4), 1-25. doi :10.1007/s10664-023-10123-7

BOSTOCK, M. (2019). *Interactive Data Visualization for the Web* (2e éd.). O'Reilly Media.

CHEN, H., CHIANG, R. H. L. & STOREY, V. C. (2012). Business Intelligence and Analytics: From Big Data to Big Impact. *MIS Quarterly*, *36*(4), 1165-1188. doi :10.2307/41703503

DOSSIER, J. (2024). Créer un visuel Power BI personnalisé avec Python. Récupérée le 9 juillet 2025, à partir de https://blog.data-examples.com/powerbi-python-custom-visual

DUPONT, A. & NGUYEN, H. (2025). *Performance Benchmark of D3-Based Custom Visuals in Power BI*. Ecrins SA R&D Whitepaper. Version 1.2. Récupérée le 10 juillet 2025, à partir de https://ecrins.ch/whitepapers/d3-performance-powerbi

ECMA INTERNATIONAL. (2024). ES2019 Performance Benchmarks Compared to Transpiled TypeScript. Récupérée le 10 juillet 2025, à partir de https://ecma-international.org/perf/es2019-typescript

FEW, S. (2009). *Now You See It: Simple Visualization Techniques for Quantitative Analysis*. Analytics Press.

GOOGLE CLOUD. (2024). Looker Marketplace—Publishing Guidelines. Récupérée le 10 juillet 2025, à partir de https://cloud.google.com/looker/docs/publish-to-marketplace

GOOGLE CLOUD. (2025). Build a Custom Visualization in Looker. Récupérée le 10 juillet 2025, à partir de https://cloud.google.com/looker/docs/create-viz-using-custom-viz

AL-KHARUSI, A. M., AL-HARTHI, H. & AL-HINAI, M. (2023). Factors Influencing Business Intelligence Adoption: An Empirical Study in Higher-Education Institutions. *Data and Knowledge Engineering*, *146*, 102126. doi :10.1016/j.datak.2023.102126

META OPEN SOURCE. (2024). React Documentation—Concurrent Rendering and Performance. Récupérée le 10 juillet 2025, à partir de https://react.dev/learn/concurrent-rendering

MICROSOFT. (2015). Announcing Open Source Power BI Visuals. Récupérée le 9 juillet 2025, à partir de https://powerbi.microsoft.com/en-us/blog/open-source-power-bi-visuals/

MICROSOFT. (2016). Power BI Visuals Marketplace Launch. Récupérée le 9 juillet 2025, à partir de https://powerbi.microsoft.com/en-us/blog/visuals-marketplace-launch/

MICROSOFT. (2017). Building Custom Visuals with D3.js. Récupérée le 9 juillet 2025, à partir de https://powerbi.microsoft.com/en-us/blog/d3-custom-visuals/

MICROSOFT. (2023). Create a Power BI Visual with pbiviz. Récupérée le 9 juillet 2025, à partir de https://learn.microsoft.com/power-bi/developer/visuals/create-custom-visual

MICROSOFT. (2024a). Create your first Power BI visual. Récupérée le 10 juillet 2025, à partir de https://learn.microsoft.com/power-bi/developer/visuals/custom-visual-develop-tutorial

MICROSOFT. (2024b). Developing Power BI Custom Visuals—Best Practices. Récupérée le 9 juillet 2025, à partir de https://learn.microsoft.com/power-bi/developer/visuals/best-practices

MICROSOFT. (2024c). ISelectionManager Interface. Récupérée le 9 juillet 2025, à partir de https://learn.microsoft.com/power-bi/developer/visuals/selection-api

MICROSOFT. (2024d). ITooltipService API Reference. Récupérée le 9 juillet 2025, à partir de https://learn.microsoft.com/power-bi/developer/visuals/tooltip-api

MICROSOFT. (2024e). Package and share a Power BI visual. Récupérée le 10 juillet 2025, à partir de https://learn.microsoft.com/power-bi/developer/visuals/package-visual

MICROSOFT. (2024f). Power BI Custom Visual Sample—React and D3 Integration. Récupérée le 10 juillet 2025, à partir de https://github.com/microsoft/PowerBI-visuals-samples/tree/main/react-d3-sample

MICROSOFT. (2024g). Power BI Custom Visuals API Changelog (v5.10 $\rightarrow$ v6 Preview). Récupérée le 9 juillet 2025, à partir de https://learn.microsoft.com/power-bi/developer/visuals/api-changelog

MICROSOFT. (2024h). Power BI Visuals—Core Visuals Source Code. Récupérée le 9 juillet 2025, à partir de https://github.com/microsoft/PowerBI-visuals

MICROSOFT. (2025a). API v6 release notes : sandbox CSP updates. Récupérée le 10 juillet 2025, à partir de https://github.com/microsoft/PowerBI-visuals/releases/tag/api-v6.0

MICROSOFT. (2025b). Certification requirements for custom visuals. Récupérée le 10 juillet 2025, à partir de https://learn.microsoft.com/power-bi/developer/visuals/certification-overview

MICROSOFT. (2025c). Create and use Python and R visuals in Power BI. Récupérée le 10 juillet 2025, à partir de https://learn.microsoft.com/power-bi/create-reports/desktop-python-visuals

MICROSOFT. (2025d). Create Power BI visuals using R. Récupérée le 10 juillet 2025, à partir de https://learn.microsoft.com/power-bi/create-reports/desktop-r-visuals

MICROSOFT. (2025e). Create visuals by using R packages in the Power BI service. Récupérée le 10 juillet 2025, à partir de https://learn.microsoft.com/power-bi/connect-data/service-r-packages-support

MICROSOFT. (2025f). Custom Visuals SDK—TypeScript Guidelines. Récupérée le 10 juillet 2025, à partir de https://learn.microsoft.com/power-bi/developer/visuals/guidelines-typescript

MICROSOFT. (2025g). Power BI Certified Visuals: Requirements and Audit Tool. Récupérée le 9 juillet 2025, à partir de https://learn.microsoft.com/power-bi/developer/visuals/certification-overview

MICROSOFT. (2025h). Power BI Community Visual Gallery. Récupérée le 10 juillet 2025, à partir de https://community.powerbi.com/t5/Data-Stories-Gallery/bd-p/DataStoriesGallery

MICROSOFT. (2025i). powerbi-visuals-tools v6.3. Récupérée le 10 juillet 2025, à partir de https://www.npmjs.com/package/powerbi-visuals-tools

MICROSOFT. (2025j). Use the certification audit flag when packaging a visual. Récupérée le 10 juillet 2025, à partir de https://learn.microsoft.com/power-bi/developer/visuals/package-visual#audit

OKVIZ. (2022a). Understanding the Power BI Visual Sandbox. Récupérée le 9 juillet 2025, à partir de https://okviz.com/blog/power-bi-visual-sandbox

OKVIZ. (2022b). Understanding the Power BI Visual Sandbox. Récupérée le 10 juillet 2025, à partir de https://okviz.com/blog/power-bi-visual-sandbox

OKVIZ. (2023). Why a custom visual cannot show a dropdown outside its bounding box. Récupérée le 10 juillet 2025, à partir de https://okviz.com/blog/dropdowns-and-bounding-box

OKVIZ. (2024). Certified visuals and export scenarios. Récupérée le 10 juillet 2025, à partir de https://okviz.com/blog/certified-visuals-export

QLIK. (2024a). Dev Hub—Creating Visualization Extensions. Récupérée le 10 juillet 2025, à partir de https://help.qlik.com/en-US/sense-developer/Content/Sense_Developer/Dev-Hub/dev-hub.htm

QLIK. (2024b). Visualization Extension API. Récupérée le 10 juillet 2025, à partir de https://help.qlik.com/en-US/sense-developer/Content/Sense_Developer/Extensions/000_ViewExtensionsAPI.htm

QLIK. (2025). Qlik Sense Pricing. Récupérée le 10 juillet 2025, à partir de https://www.qlik.com/us/pricing

REAL PYTHON TEAM. (2023). Embedding Matplotlib Plots in Power BI: A Practical Guide. Récupérée le 9 juillet 2025, à partir de https://realpython.com/power-bi-matplotlib

SPIES, C. & MERTENS, W. (2023). *BI Survey 23: The Voice of the BI and Analytics Community*. BARC – Business Application Research Center. Würzburg, Allemagne. Récupérée le 9 juillet 2025, à partir de https://bi-survey.com/the-bi-survey-23

TABLEAU SOFTWARE. (2024a). Building Interactive Write-back Extensions. Récupérée le 10 juillet 2025, à partir de https://blogs.tableau.com/extensions/writeback

TABLEAU SOFTWARE. (2024b). Tableau Extensions API—Developer Guide. Récupérée le 10 juillet 2025, à partir de https://developer.tableau.com/extensions-api/overview/

TABLEAU SOFTWARE. (2025a). Extension Security—Sandbox and Network Access. Récupérée le 10 juillet 2025, à partir de https://help.tableau.com/current/server/en-us/extensions_security.htm

TABLEAU SOFTWARE. (2025b). Tableau Pricing—Creator, Explorer, Viewer. Récupérée le 10 juillet 2025, à partir de https://www.tableau.com/pricing

TIRUPATI, K. K., JOSHI, A. & SINGH, S. P. (2023). Leveraging Power BI for Enhanced Data Visualization and Business Intelligence. *Universal Research Reports*, *10*(2), 676-711. doi :10.36676/urr.v10.i2.1375

TUFTE, E. R. (1983). *The Visual Display of Quantitative Information*. Graphics Press.

UTTAM, S. & ZHAO, M. (2025). Beyond Native Charts: Custom Visual Trends in Modern BI Platforms. *Journal of Business Analytics*, *12*(1), 22-38. Récupérée le 9 juillet 2025, à partir de https://doi.org/10.1016/j.jba.2025.01.003

W3C. (2023). Web Content Accessibility Guidelines (WCAG) 2.2. Récupérée le 10 juillet 2025, à partir de https://www.w3.org/TR/WCAG22/

WARE, C. (2019). *Information Visualization: Perception for Design* (4e éd.). Morgan Kaufmann.

WILKINSON, L. (2005). *The Grammar of Graphics* (2e éd.). Springer.

# Informations sur ce travail

### Informations de contact


Auteur: Blendar Berisha

HES-SO Valais-Wallis

E-mail: *blendar.berisha@students.hevs.ch*


### Déclaration sur l'honneur

Je déclare, par ce document, que j'ai effectué le travail de bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de bachelor, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail.

Lieu, date: _____

Signature: _____