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

Depuis plusieurs années, les outils de Business Intelligence (BI) sont devenus indispensables aux organisations qui souhaitent baser leurs décisions sur des données fiables et facilement interprétables. **Power BI** de Microsoft s'est imposé comme l'une des plateformes de référence, grâce à sa prise en main intuitive, sa connectivité riche et son intégration étroite à l'écosystème Microsoft (Microsoft, 2024). Malgré une bibliothèque déjà fournie de visuels standards, certains besoins métier demeurent inassouvis : diagrammes de Gantt avancés pour la gestion de projet, variances financières multi-niveaux, ou encore respect strict de chartes graphiques internes. **ECRINS SA**, société suisse spécialisée dans le conseil BI, fait régulièrement face à ces demandes « hors catalogue » et souhaite internaliser la création de *visuels personnalisés* afin de gagner en réactivité, en qualité et en gouvernance.

1.2 Problématique

La société n'a encore jamais mis en place de processus complet pour concevoir, tester et déployer des visuels Power BI sur mesure ; chaque tentative ponctuelle a fait ressortir plusieurs points critiques :

- **Incertitude technique** : absence de référentiel sur les performances, la scalabilité et la sécurité des composants personnalisés ;
- **Manque de normes et de traçabilité** : code développé au cas par cas, difficile à maintenir et à auditer ;
- **Planning imprévisible** : effort, coût et délai impossibles à estimer sans méthode éprouvée ;
- **Risque fonctionnel** : sans cadre clair, le livrable peut ne pas répondre pleinement aux attentes métier ou se révéler difficile à faire évoluer.

La question centrale se formule ainsi :

> **Comment concevoir et valider un cadre méthodologique complet permettant à ECRINS SA d'industrialiser, de manière fiable et pérenne, la création et la mise en production de visuels Power BI personnalisés ?**

1.3 Objectifs

O1 **Élaborer un cadre méthodologique détaillé**, couvrant analyse du besoin, développement, tests, gouvernance et mise en production ;

O2 **Réaliser un visuel pilote**, sélectionné avec la Product Owner, pour démontrer la faisabilité du cadre proposé ;

O3 **Évaluer la solution**, à l'aide de critères techniques (maintenabilité, performances, sécurité) et fonctionnels (adéquation métier).

1.4 Portée et limites

Inclus :

- développement d'un visuel pilote représentatif ;
- environnements Power BI Desktop et Service internes ;
- mise en place d'une chaîne CI/CD (GitHub Actions).

Exclus :

- publication sur Microsoft AppSource ;
- support mobile/tablette étendu ;
- constitution d'une bibliothèque exhaustive de visuels.

1.5 Structure du rapport

1. **Chapitre 2 – Cadre théorique et état de l'art** : principes des visuels Power BI, limites des visuels Python/R, justification du SDK ;
2. **Chapitre 3 – Analyse interne** : audit, benchmark, recueil des besoins et sélection du visuel pilote ;
3. **Chapitre 4 – Méthodologie de projet** : organisation Agile (Scrum-but), artefacts et planification ;
4. **Chapitre 5 – Conception et développement** : architecture, implémentation (TypeScript + D3) et tests ;
5. **Chapitre 6 – Industrialisation** : pipeline CI/CD, signature numérique et normes internes ;

6. **Chapitre 7 – Évaluation** : résultats techniques et validation fonctionnelle ;
7. **Chapitre 8 – Conclusion et perspectives**.

Cette structure garantit la cohérence et la traçabilité de l'ensemble du travail, des besoins initiaux jusqu'aux recommandations finales.

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 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 (M. P. B. TEAM, 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 (M. P. B. TEAM, 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 (M. F. COMMUNITY, 2017). Microsoft publie d'ailleurs le code source de nombreux visuels natifs sur GitHub, attestant de sa volonté d'encourager un écosystème ouvert (MICROSOFT, 2024).

2.1.1 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 (LEARN, 2023). Power BI exécute alors le visuel dans un **bac à sable sécurisé** (*sandbox*) : une `iframe` isolée du reste du rapport (FERRARI, 2022). 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 (FERRARI, 2022). Cette mesure garantit qu'aucun code malveillant ne peut lire ou exfiltrer des données sans autorisation (**MediumSecurityPBI2023**).

2.1.2 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* (LEARN, 2024a). 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, assurant ainsi une **intégration uniforme** des visuels natifs et ajoutés (LEARN, 2024b).

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.

2.2 Visuels natifs : capacités et limites

Power BI inclut en standard une collection d'une trentaine de visuels natifs couvrant les besoins courants de visualisation : histogrammes, courbes, secteurs, tables, matrices, cartes géographiques, jauges, etc., ainsi que des visuels analytiques plus avancés comme l'arborescence de décomposition (*Decomposition Tree*) ou l'analyse d'influence (*Key Influencers*). Ces visuels ont l'avantage d'être prêts à l'emploi, optimisés par Microsoft et intégrés de façon transparente à l'interface. Ils supportent nativement des fonctionnalités utiles telles que le *tooltip* (info-bulle) survolé, le *drill-down* (exploration hiérarchique), ou encore le paramétrage riche via le volet de format (couleurs, étiquettes, axes, etc.). De plus, les visuels natifs bénéficient de mises à jour régulières de Microsoft pour améliorer leurs fonctionnalités et performances. Par exemple, la fonctionnalité *small multiples* (mini-graphiques multiples) a été ajoutée à plusieurs visuels natifs suite aux retours des utilisateurs, comblant ainsi certaines lacunes de représentation dans les versions antérieures de Power BI.

Cependant, les visuels intégrés montrent des limitations de flexibilité et de personnalisation. Dans la mesure où ils sont génériques, ils ne permettent pas de répondre à tous les cas d'utilisation ou chartes graphiques spécifiques. Une critique fréquemment formulée concerne la faible latitude de personnalisation avancée comparé à des outils concurrents comme Tableau(ACADEMY, 2024). Par exemple, la mise en forme conditionnelle complexe, l'ajustement fin de la disposition des éléments, ou la combinaison de plusieurs types de visualisations en un seul objet sont souvent impossibles avec les visuels standards de Power BI. Il est rapporté que « Power BI est moins flexible en termes de design et d'interactivité. . . Créer des rapports hautement sur-mesure peut se révéler contraignant »(ACADEMY, 2024). Concrètement, si un besoin de visualisation s'écarte des modèles prévus (par exemple un diagramme spécifique à un domaine métier, ou une variante de graphique non proposée), l'utilisateur devra recourir soit à un visuel custom tiers, soit à des astuces (DAX pour produire une image, etc.), illustrant la portée limitée des visuels par défaut(ACADEMY, 2024).

Un autre point de comparaison défavorable est l'uniformité visuelle des rapports Power BI utilisant exclusivement les visuels natifs. Même s'il est possible de personnaliser les couleurs, polices et certains styles, les éléments graphiques conservent globalement la même apparence et disposition standardisées. Certaines entreprises, pour des raisons de communication et d'ergonomie, souhaitent des graphiques sortant du cadre habituel (par ex. un design graphique particulier, des infographies, des animations spécifiques). Power BI natif n'offre pas cette liberté créative, là où des frameworks de visualisation personnalisée (ou des outils comme Tableau avec son API) le permettent plus aisément(ACADEMY, 2024).

Performance et contraintes techniques. Par ailleurs, les visuels natifs sont soumis à des contraintes de performance lorsqu'ils manipulent de très larges volumes de données ou de nombreuses catégories. Microsoft recommande de limiter à ~10–20 le nombre de champs différents assignés à un visuel pour des raisons de lisibilité et de performance, avec un maximum technique de 100 champs par visuel(LEARN, 2024c). De plus, l'exportation de données depuis un visuel est limitée (ex. 150 000 lignes maximum exportables pour un visuel standard en mode Pro, au-delà il faut passer en mode Premium)(P. B. COMMUNITY, 2023).

Ces restrictions poussent parfois les utilisateurs avancés à chercher des alternatives. Une analyse critique note ainsi que « Power BI, contrairement à Tableau, est assez rigide sur la personnalisation détaillée des visuels », ce qui peut freiner des besoins spécifiques ou l'exploration interactive avancée des données(ACADEMY, 2024).

En somme, les visuels natifs offrent une base solide et fiable pour la plupart des tableaux de bord, mais ils montrent leurs limites dès qu'il s'agit de s'écarter des formats classiques ou de dépasser certaines capacités intégrées. Ces limites ont motivé l'apparition de solutions complémentaires, notamment les visuels *custom* disponibles sur le marketplace AppSource, ou l'utilisation de scripts *Python*/*R* intégrés, que nous abordons dans les sections suivantes.

2.3 Visuels Python/R : usages, atouts, limites

En plus des visuels préfabriqués, Power BI permet d'incorporer des scripts Python ou R pour générer des visuels sur mesure. Concrètement, l'utilisateur peut ajouter un élément de type "Python Visual" ou "R Visual" dans un rapport, puis fournir un script dans l'éditeur associé. Le moteur Power BI va alors exécuter ce script en coulisse, en lui transmettant les données du modèle (colonnes et mesures sélectionnées) sous forme de *dataframe*, et récupérer en résultat un graphique statique produit par le code Python ou R.

Cette fonctionnalité vise à tirer parti de l'écosystème analytique de ces langages : il devient possible d'utiliser des bibliothèques populaires comme *Matplotlib*, *Seaborn* ou *Plotly* en Python, ou *ggplot2*, *plotly R* en R, afin de créer des visualisations avancées non disponibles nativement. Par exemple, un data scientist peut réaliser dans Power BI un nuage de points avec une régression LOESS en R, ou un diagramme de réseau en Python, en quelques lignes de code utilisant des packages spécialisés — des visuels difficiles à obtenir autrement.

Les atouts de cette approche sont liés à la puissance et à la flexibilité des langages R et Python en matière de visualisation et de calculs statistiques. On bénéficie de l'énorme bibliothèque de packages open-source : analyses statistiques poussées, machine learning, visualisations scientifiques (heatmaps, dendrogrammes, etc.), tout peut théoriquement être intégré dans un rapport. De plus, ces visuels scriptés permettent d'automatiser des traitements de données directement avant l'affichage. Par exemple, on peut programmer un script Python qui agrège des données, calcule des indicateurs personnalisés ou entraîne un modèle prédictif, puis affiche un graphique du résultat — le tout s'exécutant dynamiquement lors du rafraîchissement du visuel. Cela étend les capacités de Power BI au-delà du seul langage DAX, en offrant la richesse de Python/R pour des besoins spécifiques (analyses de texte, séries temporelles avancées, etc.).

Cependant, les visuels Python/R présentent des limitations importantes dues à leur nature même de scripts externes. D'abord, le rendu produit est une image statique (format PNG) intégrée dans le rapport. Ainsi, « les visualisations Python dans Power BI ne sont que des images statiques (résolution 72 DPI), sans aucune interactivité »(PYTHON, 2023). Concrètement, l'utilisateur ne peut pas cliquer sur un élément du graphique Python/R pour filtrer d'autres visuels — toute la surface du visuel est une image plane. Il y a bien une interaction partielle : si l'on applique un filtre ou sélectionne un élément sur un autre visuel du rapport, Power BI ré-exécute le script Python/R avec les données filtrées, ce qui met à jour l'image correspondante(PYTHON, 2023). Mais cela reste plus lent et moins fluide que les visuels natifs, car il faut relancer le calcul du script à chaque changement.

Microsoft documente que ces visuels scriptés « se rafraîchissent lors des mises à jour ou filtrages des données, mais l'image en elle-même n'est pas interactive »(DOCUMENTATION, 2024). Ils répondent aux *highlightings* provenant d'autres visuels, « mais on ne peut pas sélectionner des éléments du visuel Python/R pour croiser le filtre »(DOCUMENTATION, 2024). Cette absence d'interactivité locale est un frein dans un tableau de bord où l'on attend généralement de pouvoir explorer les données de façon interactive.

Une autre contrainte forte est la taille des données transmises aux scripts. Pour éviter des temps d'exécution excessifs, Power BI limite à 150 000 lignes le volume de données qu'un visuel Python ou R peut traiter. « Si plus de 150 000 lignes sont sélectionnées, seules les 150 000 premières sont utilisées », et un message d'avertissement s'affiche sur l'image(DOCUMENTATION, 2024). De plus, l'ensemble des données en entrée du script ne doit pas excéder 250 MB en mémoire(DOCUMENTATION, 2024). Ces seuils signifient que les visuels Python/R sont adaptés à des échantillons de données ou des agrégats, mais pas au traitement de données massives brutes. Au-delà, il faut compter sur le modèle tabulaire de Power BI pour pré-agréger ou filtrer avant d'envoyer au script.

En termes de performance, il existe également un *timeout* de 5 minutes : si le script met plus de 5 minutes à s'exécuter, il sera interrompu et le visuel affichera une erreur(DOCUMENTATION, 2024). Cela empêche l'utilisation de calculs trop longs. Par exemple, entraîner un modèle complexe de machine learning sur un jeu de données volumineux dépasserait ce temps imparti.

Il convient aussi de noter des limitations techniques supplémentaires : les chaînes de caractères de plus de 32 766 caractères sont tronquées lors de la transmission au *dataframe*(DOCUMENTATION, 2024) ; certains appareils ou environnements cloud ne supportent pas ces visuels (par exemple, les rapports publiés via *publish to web* ne peuvent pas afficher de visuels Python/R pour des raisons de sécurité)(DOCUMENTATION, 2024). De plus, pour utiliser un visuel Python ou R en Power BI Desktop, il faut que l'utilisateur ait installé localement l'interpréteur Python ou R, ainsi que les packages nécessaires. Sur le service Power BI en ligne, l'exécution des scripts est prise en charge côté serveur, mais seulement pour les tenants disposant de capacités Premium ou pour les utilisateurs Pro — ce qui signifie qu'un utilisateur Free ne pourra pas voir un visuel Python/R dans un rapport à moins que celui-ci soit hébergé sur un *workspace Premium*(DOCUMENTATION, 2024).

Enfin, du point de vue sécurité, Power BI traite les visuels Python/R comme du code potentiellement risqué. Lorsqu'on ajoute pour la première fois un tel visuel, un avertissement de sécurité apparaît, invitant l'utilisateur à n'exécuter que des scripts de source fiable(DOCUMENTATION, 2024). En entreprise, l'utilisation de ces visuels peut soulever des enjeux de validation du code et de maintenance (le script étant incorporé dans le rapport, il doit être maintenu manuellement en cas de mise à jour de librairie, etc.).

Bilan. Les visuels Python et R offrent donc une solution d'appoint puissante pour réaliser des visualisations avancées ou des analyses spécifiques au sein de Power BI, sans avoir à développer un visuel *custom* complet. Ils comblent certaines lacunes des visuels natifs en ouvrant la porte à la riche panoplie des bibliothèques *data science*. Cependant, ils doivent être utilisés en connaissance de leurs limites : performances réduites, absence d'interactivité directe, et restrictions d'usage dans l'environnement de service Power BI. Pour un besoin de visualisation récurrent, à destination d'un large public, ou nécessitant une expérience utilisateur interactive, il peut être préférable de développer un visuel personnalisé via le SDK (ou d'utiliser un visuel custom existant sur AppSource) plutôt que de s'appuyer sur un script Python/R intégré. C'est ce que nous examinons dans la section suivante.

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) qui permet aux développeurs de coder, tester et empaqueter de nouveaux visuels sous forme d'un fichier .pbiviz(MICROSOFT LEARN, 2024c ; MICROSOFT POWER BI, 2024b).

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 **??**).

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 LEARN, 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` produit le fichier `.pbiviz` final, qui encapsule tout (code minifié, ressources, manifest).

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(P. B. TEAM, 2023).

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 LEARN, 2024b), et propose même un concours et une galerie communautaire pour encourager ces contributions(MICROSOFT POWER BI, 2024a).

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 DOCS, 2023b).

(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 DOCS, 2023a).

Une illustration concrète est la suivante : un slicer natif de Power BI peut afficher une liste déroulante qui dépasse la taille du visuel pour lister de nombreux éléments, alors qu'un visuel custom ne pourrait afficher un tel menu déroulant qu'en étendant son propre conteneur (ce qui n'est pas dynamique).

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 éventuellement via des approbations explicites. Les règles de certification stipulent qu'un visuel certifié ne doit émettre aucune communication externe (pas d'appels web), sous peine de rejet(BI, 2022). Même pour les visuels non certifiés, le sandbox implémente une politique de sécurité (Content Security Policy) très stricte qui bloque par défaut les requêtes sortantes non autorisées(BI, 2023b ; OVERFLOW, 2022).

Par exemple, un développeur qui tenterait dans son code d'appeler une API web externe se verra généralement opposer un refus par le navigateur intégré de Power BI (CSP violée). Ces restrictions peuvent être contournées dans certains cas (certains visuels cartographiques chargent des tuiles externes, ce qui est précisément une raison pour laquelle ils ne peuvent pas être certifiés selon Microsoft(BI, 2023c)).

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(BI, 2023a).

Les visuels certifiés sont signalés par une icône spéciale et offrent des garanties accrues aux utilisateurs. Certaines fonctionnalités de Power BI n'acceptent que des visuels certifiés : par exemple, l'export PDF/PPT d'un rapport ou l'abonnement email ne rendront visuellement que les visuels certifiés(OKVIZ DOCS, 2023c).

Un visuel non certifié apparaîtra vide dans une exportation de rapport, à moins d'être explicitement autorisé par l'admin tenant. Ceci encourage fortement les développeurs à faire certifier leurs visuels s'ils sont destinés à un usage large. La certification impose notamment de supprimer toute dépendance externe ou communication réseau, comme mentionné. Chaque mise à jour d'un visuel certifié doit repasser le processus, garantissant qu'aucune modification introduite ne viole les règles de sécurité.

Pour les organisations, Microsoft recommande de restreindre l'usage des visuels aux seuls visuels certifiés par configuration administrative, afin d'éviter l'insertion de visuels non vérifiés pouvant comporter des risques. Si un besoin métier nécessite un visuel non certifié (par ex. un visuel maison interne), il est possible de l'ajouter au store organisationnel après un examen interne du code par l'équipe sécurité, pour s'assurer qu'il ne comporte pas de comportements indésirables.

Synthèse

En résumé, le développement de visuels custom via le SDK offre une flexibilité maximale pour répondre à des besoins de visualisation spécifiques, au prix d'un effort de développement non négligeable et du respect de contraintes de sécurité. Il faut considérer le coût de maintenance (mettre à jour le visuel si l'API Power BI évolue, corriger les bugs, etc.) et l'importance de se conformer aux directives de Microsoft pour garantir la pérennité du composant. Dans la section suivante, nous discutons des choix technologiques concrets (langages, frameworks) pour implémenter un visuel custom, en justifiant l'usage de TypeScript, D3.js et éventuellement React dans ce contexte.

2.5 Choix technologiques : TypeScript, D3, (option React)

Créer un visuel Power BI custom revient essentiellement à développer une application web monopage embarquée. À ce titre, le choix des technologies de développement est déterminant pour réussir à la fois l'implémentation technique et la maintenabilité du code.

À l'heure actuelle (2023–2025), un consensus s'est formé autour du trio technologique suivant pour les visuels Power BI : **TypeScript** comme langage de développement, **D3.js** comme bibliothèque de visualisation de bas niveau, et éventuellement **React** (ou un autre framework UI) pour structurer l'interface si nécessaire. Ces choix ne sont pas exclusifs – en principe, le

SDK permet l'usage de n'importe quel framework JavaScript du moment qu'il peut s'intégrer dans le bundle en module ES6[1] – mais ils reflètent les recommandations et pratiques courantes dans la communauté.

2.5.1 TypeScript comme langage par défaut

Le SDK Power BI est conçu pour TypeScript, un sur-ensemble typé de JavaScript. Lorsqu'on initialise un projet de visuel, les fichiers générés (ex. `visual.ts`, `settings.ts`) sont en TypeScript[2].

L'adoption de TypeScript présente plusieurs avantages :

- (a) Typage statique permettant d'attraper des erreurs à la compilation et de mieux documenter les intentions du code, ce qui est précieux pour des visuels complexes ;
- (b) Meilleure productivité grâce à l'autocomplétion et la documentation intégrée dans les IDE (Microsoft fournit les définitions d'interfaces du Power BI Visuals API en TypeScript) ;
- (c) Interopérabilité avec JavaScript – TypeScript compile en JavaScript standard, ce qui permet d'utiliser toute librairie JS existante.

En somme, TypeScript est devenu le langage de facto pour ce genre de développement front-end structuré. D'après les experts, « pour bien faire les choses, il faut utiliser le SDK TypeScript, qui donne accès à toutes les APIs fournies par Microsoft »[3].

Un développeur purement JavaScript pourrait en théorie coder un visuel, mais il perdrait les bénéfices du typage et devrait écrire des bindings à la main pour l'API – d'où l'intérêt de se conformer à l'outil TypeScript officiel.

2.5.2 D3.js pour le rendu visuel

D3.js (Data-Driven Documents) est une bibliothèque JavaScript largement utilisée pour manipuler le DOM en fonction des données, en particulier via SVG, pour créer des visualisations personnalisées. Microsoft elle-même a misé sur D3.js dès l'origine des custom visuals : « en s'appuyant sur des librairies open-source comme D3.js, nous avons rendu la création de nouveaux visuels incroyablement simple »[4].

La majorité des visuels personnalisés (et même plusieurs visuels natifs sous le capot) utilisent D3 pour générer les éléments graphiques (formes, échelles, axes...).

[1] https://www.reddit.com/r/PowerBI/comments/12ggh93/is_it_possible_to_use_react_with_powerbi_custom/
[2] https://learn.microsoft.com/en-us/power-bi/developer/visuals/custom-visual-develop-tutorial
[3] https://www.reddit.com/r/PowerBI/comments/12ggh93/is_it_possible_to_use_react_with_powerbi_custom/
[4] https://powerbi.microsoft.com/fr-fr/blog/announcing-the-power-bi-custom-visuals-contest/

D3 offre un contrôle très fin sur le rendu : on peut construire quasiment n'importe quel type de visualisation en partant de primitives de base (lignes, cercles, chemins SVG) liées aux données par des bindings data→DOM. Par exemple, pour un graphique en barres custom, D3 facilite la création des `<rect>` dimensionnés selon les valeurs des données et applique des échelles et axes avec quelques fonctions.

Un avantage clé est que D3 est agnostique : on peut l'utiliser dans le sandbox sans souci, et il n'a pas de dépendances externes. En fait, Power BI intègre une prise en charge particulière de D3 : il a été noté que « Power BI attache D3 au contexte global de la sandbox, ce qui le rend immédiatement disponible »[5].

Cela signifie que D3 s'exécute aisément dans le cadre isolé, alors que d'autres librairies nécessitent parfois des ajustements (voir paragraphe sur React). D3 est donc hautement recommandé pour quiconque crée un visuel custom du fait de sa puissance et de sa souplesse.

Bien sûr, son utilisation requiert une certaine expertise (ce n'est pas une librairie de chart "clé en main" mais un outil bas niveau). Alternativement, certains développeurs optent pour des librairies plus haut niveau (par ex. Chart.js, Plotly.js, ou Vega-Lite).

Celles-ci peuvent être utilisées via le SDK, mais elles offrent moins de contrôle et parfois posent des problèmes de taille de bundle ou de compatibilité. D3 reste le standard de facto, au point que même des solutions no-code comme Deneb (un visuel custom permettant de configurer des graphiques via Vega-Lite) utilisent D3 en interne pour le rendu.

2.5.3 React (optionnel) pour la structure UI

L'utilisation de React – ou d'un autre framework comme Angular, Vue – dans un visuel Power BI est possible, bien que non indispensable dans tous les cas. Un visuel Power BI est par nature assez ciblé (il rend une visualisation spécifique dans un espace contraint).

Beaucoup de visuels custom n'emploient donc pas de framework UI sophistiqué : le code TypeScript avec D3 suffit à créer le SVG ou canvas voulu.

Néanmoins, React peut s'avérer utile dans plusieurs scénarios :

- (a) Visuels complexes avec état interne – par exemple un visuel qui propose une petite interface utilisateur (boutons, menus) en plus du graphique. React aide à organiser le code en composants et à gérer les interactions utilisateur de manière déclarative.
- (b) Réutilisation de composants existants – un développeur peut intégrer des librairies React de composants (par ex. un sélecteur de couleur, une timeline interactive) au lieu de tout coder from scratch.

[5] https://community.fabric.microsoft.com/t5/Developer/React-js-not-working-properly-in-custom-visual/td-p/3111857

- (c) Faciliter le développement – pour des développeurs déjà familiers avec React, il peut être plus confortable de raisonner en composant React et virtual DOM qu'en manipulations directes du DOM.

Microsoft a d'ailleurs publié un tutoriel officiel montrant pas à pas la création d'un visuel custom en React (un "circle card" affichant un KPI formaté)[6].

Cela dit, intégrer React dans un visuel custom introduit de la complexité. D'une part, cela alourdit le package car il faut inclure React (et possiblement ReactDOM) dans le bundle du visuel – sauf si l'on s'appuie sur la version déjà incluse par Power BI (non garantie dans le sandbox).

D'autre part, du fait du sandboxing, le `window` global utilisé par React n'est pas le même contexte que celui du host Power BI. Des développeurs ont constaté qu'il fallait ruser pour faire fonctionner React, par exemple en sauvegardant une référence du `window` isolé pour l'utiliser lors du rendu, car « lorsque React est chargé, il est attaché à un objet `window` qui est ensuite modifié par le sandbox, le rendant indisponible lors de l'appel `update` »[7].

Un ingénieur Microsoft a expliqué à ce sujet que « Power BI crée une copie de l'objet `window` pour isoler les librairies du visuel de celles de l'application ; D3 est relié correctement au contexte global, mais React ne l'était pas, nécessitant ce contournement »[8].

Il a suggéré qu'à terme l'isolation pourrait être assouplie. Il s'agit de détails techniques, mais qui illustrent que toute librairie externe doit être testée pour fonctionner dans l'environnement particulier de Power BI.

2.5.4 Autres choix et écosystème

À noter que TypeScript/D3/React ne sont pas les seules technologies possibles, mais représentent un socle éprouvé. Certains visuels spécifiques intègrent par exemple Mapbox (pour la cartographie), Three.js (pour de la 3D), ou d'autres libs selon le besoin – ce qui est faisable tant que cela respecte les contraintes du sandbox et du bundle.

Par ailleurs, pour les utilisateurs non développeurs qui veulent tout de même des visuels sur mesure, des outils comme Charticulator (de Microsoft Research) ou Deneb (basé sur Vega-Lite) permettent de créer des visuels custom sans écrire du code, en restant dans les limites du sandbox et avec un résultat intégrable comme un visuel custom.

Ces alternatives confirment qu'au cœur, le moteur de rendu reste du JS/TS manipulant le DOM SVG/HTML.

[6] https://learn.microsoft.com/en-us/power-bi/developer/visuals/custom-visual-develop-react-circle-card

[7] https://community.fabric.microsoft.com/t5/Developer/React-js-not-working-properly-in-custom-visual/td-p/3111857

[8] https://community.fabric.microsoft.com/t5/Developer/React-js-not-working-properly-in-custom-visual/td-p/3111857

2.5.5 Choix technologique pour ECRINS SA

Dans le contexte du projet de bachelor « Création de composants BI custom dans Power BI » pour ECRINS SA, le choix de TypeScript et D3.js apparaît donc incontournable pour développer de nouveaux visuels performants et maintenables.

TypeScript assurera la robustesse du code et la compatibilité avec l'API Power BI, tandis que D3.js fournira la boîte à outils nécessaire pour représenter visuellement les données de la manière la plus flexible possible.

L'adoption de React dépendra de la complexité des visuels à créer : si les cas d'usage exigent des interactions UI poussées ou si l'équipe de développement maîtrise déjà React, il pourrait être judicieux de l'utiliser pour structurer l'application visuelle. Sinon, on pourra s'en passer afin de minimiser la complexité.

Quoiqu'il en soit, le respect des bonnes pratiques de sécurité (pas de dépendances externes non contrôlées, respect du sandbox) et d'optimisation (volume de données raisonnable, rendu efficace) guidera le développement. En s'appuyant sur ce trio technologique moderne, l'entreprise disposera d'une base solide pour étendre les capacités de Power BI avec des visuels sur mesure adaptés à ses besoins, tout en s'alignant sur l'état de l'art actuel en matière de visualisation de données interactive.

3 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

ACADEMY, F. (2024). Power BI vs Tableau – Which Should You Choose? Consulté le 4 juin 2025. Récupérée le 4 juin 2025, à partir de https://fynd.academy/blog/power-bi-vs-tableau

BI, M. P. (2022). Why certified visuals cannot send external requests. Consulté en juin 2025. Récupérée le à partir de https://medium.com/microsoft-power-bi/why-certified-visuals-cannot-use-external-calls

BI, M. P. (2023a). Certified visuals for Power BI. Consulté en juin 2025. Récupérée le à partir de https://medium.com/microsoft-power-bi/certification-process-for-custom-visuals

BI, M. P. (2023b). Custom visual security guidelines. Consulté en juin 2025. Récupérée le à partir de https://medium.com/microsoft-power-bi/custom-visuals-security

BI, M. P. (2023c). Why some visuals (like maps) can't be certified. Consulté en juin 2025. Récupérée le à partir de https://medium.com/microsoft-power-bi/why-map-visuals-are-not-certifiable

COMMUNITY, M. F. (2017). D3.js Supported Versions for Power BI Custom Visuals. Consulté le 4 juin 2025. Récupérée le 4 juin 2025, à partir de https://community.fabric.microsoft.com/t5/Custom-Visuals-Development/D3-js-Supported-versions/m-p/246488

COMMUNITY, P. B. (2023). Export Limitations for Data from Visuals. Consulté le 4 juin 2025. Récupérée le 4 juin 2025, à partir de https://community.powerbi.com/t5/Service/Exporting-Data-from-Visuals-Limitations/m-p/2898302

DOCUMENTATION, M. (2024). Create Power BI visuals using Python and R scripts. Récupérée le à partir de https://learn.microsoft.com/en-us/power-bi/visuals/service-python-r-visuals

FERRARI, M. (2022). Power BI Custom Visuals: Development, Security, and Updates. Consulté le 4 juin 2025. Récupérée le 4 juin 2025, à partir de https://okviz.com/blog/power-bi-custom-visuals-development-security-and-updates/

LEARN, M. (2023). Develop Custom Visuals in Power BI. Consulté le 4 juin 2025. Récupérée le 4 juin 2025, à partir de https://learn.microsoft.com/en-us/power-bi/developer/visuals/develop-power-bi-visuals

LEARN, M. (2024a). Guidelines for Publishing Power BI Custom Visuals. Consulté le 4 juin 2025. Récupérée le 4 juin 2025, à partir de https://learn.microsoft.com/en-us/power-bi/developer/visuals/guidelines-powerbi-visuals

LEARN, M. (2024b). Power BI Visual Data Point Selections – Selection API. Consulté le 4 juin 2025. Récupérée le 4 juin 2025, à partir de https://learn.microsoft.com/en-us/power-bi/developer/visuals/selection-api

LEARN, M. (2024c). Power BI visuals - Data point limits and performance tips. Consulté le 4 juin 2025. Récupérée le 4 juin 2025, à partir de https://learn.microsoft.com/en-us/power-bi/visuals/power-bi-visualization-best-practices#data-point-limits

MICROSOFT. (2024). Power BI Visuals – Sample Bar Chart (GitHub Repository). Consulté le 4 juin 2025. Récupérée le 4 juin 2025, à partir de https://github.com/microsoft/PowerBI-visuals-sampleBarChart

MICROSOFT LEARN. (2024a). Create a Power BI visual - Step-by-step. Consulté en juin 2025. Récupérée le à partir de https://learn.microsoft.com/en-us/power-bi/developer/visuals/create-custom-visual

MICROSOFT LEARN. (2024b). Power BI visual packages (.pbiviz). Consulté en juin 2025. Récupérée le à partir de https://learn.microsoft.com/en-us/power-bi/developer/visuals/pbiviz-packaging

MICROSOFT LEARN. (2024c). Power BI visuals overview. Consulté en juin 2025. Récupérée le à partir de https://learn.microsoft.com/en-us/power-bi/developer/visuals/power-bi-custom-visuals-overview

MICROSOFT POWER BI. (2024a). Custom visuals community and gallery. Consulté en juin 2025. Récupérée le à partir de https://powerbi.microsoft.com/en-us/custom-visuals/

MICROSOFT POWER BI. (2024b). GitHub – powerbi-visuals-tools. Consulté en juin 2025. Récupérée le à partir de https://github.com/microsoft/PowerBI-visuals-tools

OKVIZ DOCS. (2023a). Bounding box and layout constraints. Consulté en juin 2025. Récupérée le à partir de https://docs.okviz.com/limitations/ui-constraints/

OKVIZ DOCS. (2023b). Custom visuals limitations - data access. Consulté en juin 2025. Récupérée le à partir de https://docs.okviz.com/limitations/custom-visuals/

OKVIZ DOCS. (2023c). Export limitations with non-certified visuals. Consulté en juin 2025. Récupérée le à partir de https://docs.okviz.com/limitations/export-certification/

OVERFLOW, S. (2022). Power BI custom visuals blocked by CSP. Consulté en juin 2025. Récupérée le à partir de https://stackoverflow.com/questions/67860132/power-bi-custom-visual-blocked-by-csp

PYTHON, R. (2023). How to Use Python in Power BI. Récupérée le à partir de https://realpython.com/python-power-bi/

TEAM, M. P. B. (2015). Visualize Your Data, Your Way Using Custom Visuals in Power BI. Consulté le 4 juin 2025. Récupérée le 4 juin 2025, à partir de https://powerbi.microsoft.com/en-us/blog/visualize-your-data-your-way-using-custom-visuals-in-power-bi/

TEAM, M. P. B. (2016). Deep Dive in the Organizational Custom Visuals. Consulté le 4 juin 2025. Récupérée le 4 juin 2025, à partir de https://powerbi.microsoft.com/nl-be/blog/deep-dive-in-the-organizational-custom-visuals/

TEAM, P. B. (2023). How to share custom visuals in your organization. Consulté en juin 2025. Récupérée le à partir de https://medium.com/microsoft-power-bi/how-to-share-custom-visuals-in-your-org

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: _____