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 14 août 2025

Résumé

Les visuels natifs de Power BI ne couvrant pas certains besoins analytiques d'ECRINS SA, l'entreprise souhaite internaliser la conception de visuels personnalisés. Ce travail poursuit deux objectifs : (1) établir un cadre méthodologique complet — de l'analyse à la mise en production — et (2) démontrer la faisabilité à travers deux prototypes : une Passenger-Flow Map (marketing aéroportuaire) et un Radial Sunburst Decomposition Tree (analyse budgétaire).

La démarche combine revue de littérature, cadrage avec le métier et développement structuré en phases, appuyé par un outillage de build et d'automatisation. Les visuels sont réalisés en TypeScript et D3, intégrés à Power BI via pbiviz, puis évalués sur des jeux de données de démonstration selon des critères de performance, d'ergonomie et de maintenabilité. Une chaîne de livraison outillée (tests, packaging, distribution interne) soutient l'industrialisation.

L'évaluation technique et fonctionnelle confirme la validité des deux prototypes et la reproductibilité du processus proposé. Les livrables — code, pipeline d'automatisation, guides et annexes — fournissent à ECRINS SA une base réutilisable pour futurs développements de visuels personnalisés et pour renforcer la gouvernance BI.

Mots-clés : Business Intelligence ; Power BI ; visuels personnalisés ; TypeScript/D3 ; performance ; ergonomie ; industrialisation.

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

Résumé ii

Remerciements iii

Table des matières iv

Table des figures x

Liste des tableaux xi

1 Introduction 1
- 1.1 Contexte . . . 1
- 1.2 Problématique . . . 2
- 1.3 Objectifs . . . 2
- 1.4 Portée et limites . . . 3
- 1.5 Structure du rapport . . . 3

2 État de l'art 4
- 2.1 Concepts BI et datavisualisation . . . 4
- 2.2 Architecture des visuels Power BI . . . 5
  - 2.2.1 Visual container et bac à sable . . . 6
  - 2.2.2 Interactions et intégration . . . 6
  - 2.2.3 AppSource comme catalogue de visuels certifiés . . . 7
- 2.3 Visuels Python / R : usages, atouts, limites . . . 7
  - 2.3.1 Atouts analytiques . . . 8
  - 2.3.2 Limites techniques et fonctionnelles . . . 8
  - 2.3.3 Enjeux de sécurité et de maintenance . . . 8
  - 2.3.4 Bilan et positionnement stratégique . . . 9
- 2.4 SDK Custom Visuals : principes, sécurité, pipeline . . . 9
  - 2.4.1 Principe général. . . . 9
  - 2.4.2 Pipeline de développement . . . 9
  - 2.4.3 Cadre de sécurité. . . . 9
  - 2.4.4 Certification et limitations associées . . . 10
  - 2.4.5 Synthèse. . . . 10
- 2.5 Choix technologiques (TypeScript, D3, React optionnel) . . . 10

2.5.1 TypeScript vs JavaScript. 10
2.5.2 D3.js pour le rendu SVG 11
2.5.3 React (optionnel) pour l'UI. 11
2.5.4 Synthèse. 11
2.6 Solutions concurrentes (Tableau, Qlik, Looker) 12
2.6.1 Tableau. 12
2.6.2 Qlik Sense. 12
2.6.3 Looker. 12
2.6.4 Lecture comparative. 13
2.7 Synthèse des écarts & opportunités 13
2.7.1 Visuels natifs Power BI. 13
2.7.2 Visuels Python / R. 13
2.7.3 Visuels AppSource (certifiés). 14
2.7.4 Visuels SDK Power BI. 14
2.7.5 Vue d'ensemble comparative. 14
2.7.6 Conclusion. 14

3 Contexte du pilote, méthodologie & organisation 16
3.1 Origine du besoin 16
3.1.1 Besoin marketing : carte des flux passagers aéroportuaire 16
3.1.2 Besoin d'analyse hiérarchique : Sunburst de décomposition 16
3.2 Critères de réussite 16
3.2.1 Performance 17
3.2.2 Lisibilité minimale 17
3.2.3 Internationalisation essentielle 17
3.2.4 Qualité du code 17
3.2.5 Intégration continue (CI/CD) 18
3.3 Plan de tests (datasets démo, scénarios tableau de bord) 18
3.3.1 Carte de flux de passagers (Passenger-Flow Map) 18
3.3.2 Sunburst hiérarchique (Analyse multi-niveaux) 18
3.4 Organisation du travail — cycle Waterfall en cinq phases 19
3.5 Sources de validation (revue experte et mini-test utilisateur) 20
3.5.1 Revue experte fonctionnelle. 21
3.5.2 Démonstration interne. 21
3.5.3 Mini-test utilisateur exploratoire. 21
3.5.4 Limites assumées. 21

4 Playbook générique de développement des visuels Power BI 22
4.1 Objet et portée du playbook 22
4.2 Présentation du SDK Power BI Custom Visuals et de pbiviz 23
4.3 Pré-requis techniques du développeur 24
4.4 Mise en place de l'environnement de développement 25

4.5 Création d'un composant minimal avec pbiviz new 26
4.5.1 Génération du squelette de projet 26
4.5.2 Installation des dépendances NPM 26
4.5.3 Premier lancement en mode développeur 27
4.5.4 Vérification dans Power BI Service 27
4.5.5 Cycle itératif de modification 27
4.5.6 Arrêt propre du serveur 27
4.6 Structure des fichiers générés par pbiviz 28
4.6.1 Arborescence racine 28
4.6.2 capabilities.json 28
4.6.3 visual.ts 28
4.6.4 settings.ts 29
4.6.5 pbiviz.json 29
4.6.6 Interdépendance et bonnes pratiques 29
4.7 Débogage et hot-reload dans Power BI Service 29
4.7.1 Insertion du visuel développeur 29
4.7.2 Premier affichage et état d'attente 30
4.7.3 Association de données et contrôles de base 30
4.7.4 Hot-reload automatique 30
4.7.5 Diagnostic des erreurs fréquentes 30
4.7.6 Nettoyage de la session et reprise 31
4.8 Erreurs courantes et contournements 31
4.8.1 pbiviz introuvable après installation. 31
4.8.2 Échec de connexion HTTPS : « Can't contact visual server ». 31
4.8.3 Icône développeur absente dans le volet des visuels. 32
4.8.4 DataView vide ou partiel malgré la sélection de champs. 32
4.8.5 Erreur PowerShell lors de pbiviz –install-cert. 32
4.8.6 Visuel fonctionnel en mode développeur mais défectueux après empaquetage. 32
4.8.7 Avertissements TypeScript et incompatibilités de types. 32
4.8.8 Taille de package anormalement faible. 32
4.9 Checklist de démarrage rapide 33
4.10 Mise en pratique du playbook : introduction aux prototypes Passenger-Flow Map et Sunburst budgétaire 34

5 Développement des visuels Power BI personnalisés 35
5A Passenger-Flow Map 35
5A.1 Principes architecturaux et structure modulaire 35
5A.2 Chaîne de traitement des données — du DataView au DOM 36
5A.3 Calcul des trajectoires de flux : principe et réglages 37
5A.4 Rendu graphique et animation des flux 38
5A.5 Panneau de contrôle et interactions utilisateur 39

5A.6 Configuration des données et des paramètres (capabilities.json & settings.ts) 40
5A.7 Synthèse et positionnement pour ECRINS SA 41
5B Decomposition-Tree / Sunburst 42
5B.1 Principes architecturaux et structure modulaire 42
5B.2 Pipeline de traitement des données : de update() au DOM 43
5B.3 Algorithmes de construction des anneaux : partitionnement radial et allocation des rayons 44
5B.4 Rendu SVG, transitions et KPI visuel 45
5B.5 Interactions utilisateur : drill-down, fil d'Ariane et cross-highlight 45
5B.6 Configuration des données et des paramètres 46
5B.7 Synthèse et positionnement pour ECRINS SA 47

6 Industrialisation et mise en production 49
6.1 Cadre général 49
6.2 Notions utiles 49
6.3 Exigences de la chaîne 50
6.4 Chaîne d'intégration et de livraison 50
6.4.1 Build continu 50
6.4.2 Release sur tag 52
6.5 Intégrité et signature 54
6.6 Diffusion dans le magasin organisationnel 54
6.7 Impacts et limites 54
6.8 Rôles et responsabilités 54
6.9 Traçabilité et maintenance 54

7 Évaluation des résultats 56
7.1 Mesures techniques : performance, accessibilité, poids 56
7.1.1 Temps de rendu et fluidité 56
7.1.2 Poids du bundle et optimisation 56
7.1.3 Accessibilité et internationalisation 56
7.1.4 Conformité aux contraintes de sécurité 57
7.2 Validation fonctionnelle et retours utilisateurs 57
7.2.1 Méthodologie des tests 57
7.2.2 Retours sur la carte de flux de passagers 57
7.2.3 Retours sur le visuel Sunburst 57
7.2.4 Satisfaction générale et axes d'amélioration 58
7.3 Synthèse métier et perception globale 58

8 Conclusion & perspectives 59
8.1 Synthèse 59
8.2 Limites 60
8.3 Recommandations et perspectives 61
8.4 Pistes d'évolution 62

A Annexes **64**
A.1 Dictionnaires de données (datasets démo) . . . 64
A.1.1 Passenger-Flow Map . . . 64
A.1.2 Sunburst budgétaire . . . 64
A.2 Fond de carte & obstacles . . . 65
A.2.1 Schéma obstacles.json (extrait) . . . 65
A.2.2 Schéma technique de superposition . . . 66
A.2.3 Plan illustratif de l'aéroport (JPEG) . . . 66
A.3 Tests et conformité . . . 67
A.3.1 Environnement et périmètre . . . 67
A.3.2 Protocole de mesure . . . 67
A.3.3 Résultats (synthèse) . . . 68
A.3.4 Distribution des temps (visualisations) . . . 68
A.3.5 Accessibilité (WCAG 2.2 — périmètre du projet) . . . 68
A.3.6 Sécurité et packaging . . . 69
A.4 Procédure développeur « express » . . . 71
A.4.1 Prérequis (environnement) . . . 71
A.4.2 Arborescence minimale . . . 71
A.4.3 Scripts NPM (référence) . . . 71
A.4.4 Démarrage local (dev) . . . 72
A.4.5 Packaging (artefact .pbiviz) . . . 72
A.4.6 Signature (si politique interne requise) . . . 72
A.4.7 Contrôles rapides (QA) . . . 72
A.5 Guide d'installation (utilisateurs) & changelog . . . 72
A.5.1 Installation — import manuel .pbiviz . . . 72
A.5.2 Installation — magasin organisationnel . . . 72
A.5.3 Compatibilité et paramètres locataire . . . 73
A.5.4 Bonnes pratiques d'usage . . . 73
A.5.5 Modèle de changelog (à conserver dans le dépôt) . . . 73
A.5.6 Canal de diffusion sécurisé . . . 73
A.6 Publication dans le magasin organisationnel : vérifications préalables et mode opératoire . . . 73
A.6.1 Objet et portée . . . 73
A.6.2 Pourquoi ces étapes . . . 73
A.6.3 Pré-requis côté administrateur . . . 74
A.6.4 Vérification d'intégrité . . . 74
A.6.5 Vérification de la signature interne (si activée) . . . 74
A.6.6 Contrôle de conformité technique . . . 74
A.6.7 Publication dans le magasin organisationnel . . . 74
A.6.8 Mises à jour et gestion de versions . . . 75
A.6.9 Contrôles post-publication . . . 75
A.6.10 Procédure de retour arrière . . . 75

A.6.11 Archivage 75
A.7 Activer la signature interne (parcours simple) 75

Références **80**

List of Figures

2.1 Architecture simplifiée d'un visuel Power BI personnalisé 6
2.2 Pipeline d'exécution d'un visuel Python / R dans Power BI 8

4.1 Cycle développeur pour un visuel Power BI 22

5.1 Architecture hexagonale du visuel Passenger-Flow Map. 35
5.2 Évitement d'obstacles et simplification de tracé (A* $\rightarrow$ rendu). 38
5.3 Architecture hexagonale du visuel Radial Sunburst Decomposition Tree 42
5.4 Chaîne de rendu du Sunburst 43

A.1 Overlay technique généré à partir des obstacles normalisés. 66
A.2 Plan illustratif de l'aéroport 67
A.3 Radial Sunburst Decomposition Tree — rendu sur dataset budgétaire de démonstration. 68
A.4 Passenger-Flow Map — rendu sur dataset de flux passagers de démonstration. . . 69
A.5 Radial Sunburst Decomposition Tree — histogramme des temps de cycle (ms). . . 70
A.6 Passenger-Flow Map — histogramme des temps de cycle (ms). 70

List of Tables

2.1 Comparaison factuelle des voies de personnalisation Power BI (état : août 2025) . . 15

3.1 Découpage du projet selon un cycle Waterfall 19

7.1 Perception croisée des deux visuels 58

A.1 Dictionnaire — Passenger-Flow Map (compact) 64
A.2 Dictionnaire — Sunburst budgétaire (compact) 65

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 qu'environ 25 % des entreprises l'ont déjà déployé et que près de 43 % envisagent de l'adopter à court terme, selon la BI Survey 23 de BARC (Tirupati, 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 mais également du SDK Power BI . 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, 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 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 SA peut-elle démontrer la faisabilité, concevoir et intégrer de nouveaux composants BI personnalisés dans Power BI afin de répondre à des besoins métier non couverts par les visuels natifs, tout en définissant un cadre de développement reproductible et fiable pour les futures réalisations ?

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, 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, 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. Les visuels réalisés sont conçus comme des preuves de concept destinées à valider les choix techniques — ils n'ont pas vocation à être déployés tels quels en production. Concrètement, il s'agit d'analyser l'existant afin de recenser les lacunes, de sélectionner deux visuels représentatifs à forte valeur didactique, de déterminer l'approche technique (scripts ou

SDK), d'implémenter ces prototypes dans un tableau de bord de démonstration et de les tester, puis de documenter une procédure méthodologique reproductible à laquelle ECRINS SA pourra se référer pour ses futurs développements.

1.4 Portée et limites

Le travail se concentre sur la réalisation de deux prototypes de composants 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. Ne sont pas inclus dans ce périmètre : la certification AppSource, l'optimisation pour des volumes de données massifs, le déploiement mobile et les contraintes de sécurité propres à un environnement de production. Les prototypes sont évalués exclusivement à partir de données fictives dans un environnement de test contrôlé, sans intégration directe aux systèmes d'information existants de l'entreprise.

1.5 Structure du rapport

Outre la présente introduction, le mémoire s'articule en huit chapitres complémentaires. Le chapitre 2 dresse 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 l'origine du besoin, les critères de réussite retenus et l'organisation du travail, ce qui constitue la méthodologie du projet. Le chapitre 4 présente un playbook générique de développement de visuels Power BI personnalisés, servant de référentiel interne pour ECRINS SA. Le chapitre 5 décrit la conception et la réalisation des deux visuels développés — Passenger-Flow Map et Sunburst / Decomposition Tree — en détaillant leur architecture, leurs choix techniques et leurs fonctionnalités. Le chapitre 6 formalise la procédure d'industrialisation : pipeline CI/CD, signature numérique et exigences de gouvernance. Le chapitre 7 présente l'évaluation des résultats tant sur le plan technique que fonctionnel, tandis que le chapitre 8 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 dense qui combine l'offre native de la plateforme et des extensions construites par code. Afin de situer précisément le cadre de ce travail, ce chapitre commence par décrire l'architecture interne des visuels Power BI et distingue les trois grandes familles actuellement accessibles aux concepteurs : les visuels livrés par Microsoft, les visuels générés par scripts Python/R et les visuels personnalisés réalisés au moyen du SDK officiel. Chaque famille est examinée tour à tour, en s'appuyant sur la documentation Microsoft (2025) et les travaux de référence en datavisualisation (MICROSOFT, 2025clabelday ; WARE, 2019labelday). L'analyse porte autant sur les capacités offertes que sur les limitations fonctionnelles, de performance ou d'accessibilité que les praticiens doivent anticiper.

Dans un second temps, nous explicitons les choix technologiques retenus pour ce projet : le langage TypeScript, le moteur de rendu SVG de D3.js et, le cas échéant, l'emploi de React pour la gestion du DOM. Chaque choix est justifié au regard des exigences identifiées précédemment (maintenabilité, interactivité et conformité WCAG 2.2).

Enfin, la dernière section opère une analyse critique des écarts entre les besoins métier relevés chez ECRINS SA et l'état actuel de l'art. Ce bilan établit la légitimité d'un développement custom et prépare les décisions techniques détaillées au chapitre 5.

2.1 Concepts BI et datavisualisation

La Business Intelligence 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, 2012labelday) 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. Un rapport plus récent de Gartner (GARTNER, 2024labelday) confirme cette évolution vers une BI self-service, dans laquelle la souplesse des visuels et leur rapidité de création constituent des facteurs différenciants.

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, couleur, etc.). Les travaux fondateurs de Cleveland et McGill établissent une hiérarchie de précision des encodages (la position sur une échelle commune étant la plus fiable) (CLEVELAND & MCGILL, 1984labelday), complétés par une synthèse de référence sur les principes et processus

perceptifs (MUNZNER, 2014labelday). Ware (WARE, 2019labelday) démontre que l'exploitation adéquate de ces attributs maximise la vitesse et la justesse de la lecture visuelle. Tufte (TUFTE, 1983labelday) 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. Des travaux empiriques plus récents nuancent toutefois cette position, en montrant que la mémorabilité d'un visuel dépend aussi de facteurs perceptifs et sémantiques (BORKIN et al., 2013labelday). Few (FEW, 2009labelday) prolonge cette perspective en montrant que la cohérence des encodages (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, 2005labelday) puis formalisée 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. L'adoption de D3 pour les visuels personnalisés s'appuie par ailleurs sur une base scientifique et industrielle reconnue (BOSTOCK et al., 2011labelday).

Au-delà des principes, la BI professionnelle ajoute des impératifs tels que la performance d'affichage et l'accessibilité numérique, auxquels s'ajoute la qualité des interactions comme levier d'analyse (HEER & SHNEIDERMAN, 2012labelday). Ces impératifs serviront de grille d'évaluation au chapitre 6. 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 visuel — graphique, jauge, carte ou autre — est rendu côté client à partir d'un DataView spécifique, généré automatiquement à partir du modèle de données filtré, via du code JavaScript/TypeScript exécuté dans Power BI Desktop ou dans le service web (MICROSOFT, 2015labelday). 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, 2016labelday). Cette ouverture repose sur des standards web — « en s'appuyant sur des standards ouverts d'Internet et des bibliothèques open-source comme D3.js », note Microsoft (MICROSOFT, 2017labelday) — et se matérialise sur GitHub où le code source de nombreux visuels natifs est publié (MICROSOFT, 2024hlabelday). L'API publique a évolué de

la branche 5.10 à la version stable 6.1.2 (26 mai 2025), qui introduit un DOM sécurisé ainsi que la Rendering Events API, renforçant la cohérence entre ouverture et sécurité (MICROSOFT CORPORATION, 2025alabelday).

Cette section examine d'abord le fonctionnement interne d'un visuel Power BI — de son chargement sécurisé à son intégration interactive — avant de présenter la plateforme AppSource, qui permet d'étendre les visuels disponibles sans développement.

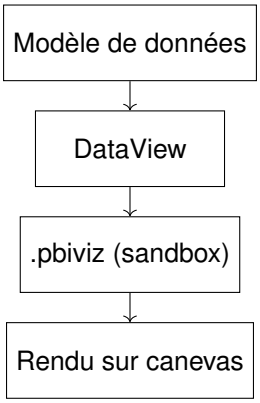

FIGURE 2.1 *: Architecture simplifiée d'un visuel Power BI personnalisé*

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 scripts, styles et manifeste (MICROSOFT, 2023labelday). Power BI exécute alors le visuel dans un bac à sable sécurisé (sandbox) : une iframe isolée du reste du rapport (OKVIZ, 2022labelday). Le visuel n'accède ni aux autres visuels ni au modèle global ; il ne « voit » que les champs explicitement liés par l'utilisateur. Depuis la mise à jour 2.140 (février 2024), tout visuel — 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, 2025dlabelday). Ces garde-fous empêchent qu'un code malveillant puisse lire ou exfiltrer des données sans autorisation (SRINIVASAN, 2023labelday).

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, 2024blabelday). 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, 2024clabelday, 2024dlabelday).

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.

2.2.3 AppSource comme catalogue de visuels certifiés

En complément des visuels natifs, Power BI propose une galerie en ligne nommée AppSource, accessible directement depuis l'interface de conception de rapports. Cette plateforme répertorie plusieurs centaines de visuels certifiés, gratuits ou payants, classés par thématique (cartographie, indicateurs, diagrammes spécialisés, etc.). Les utilisateurs peuvent y intégrer des composants supplémentaires sans compétence de développement, ce qui en fait une solution courante lorsqu'un besoin spécifique n'est pas couvert par les visuels standards. Toutefois, ces visuels restent limités à ce que les éditeurs tiers ont déjà publié, et leur personnalisation est restreinte. Dans le cadre de ce travail, AppSource est considéré comme une ressource de référence — utile à des fins de comparaison fonctionnelle —, mais le projet se concentre sur la création de composants ad hoc conçus pour un besoin métier spécifique, sans visée de diffusion publique via le store.

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 insère un composant « Python visual » ou « R visual » dans le canevas, saisit le code dans un éditeur intégré, 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 rapport.

Cette fonctionnalité exploite l'écosystème analytique complet de chaque langage : Matplotlib, Seaborn ou Plotly côté Python ; ggplot2 ou plotly R côté R. Un data-scientist peut, en quelques lignes, tracer dans Power BI un nuage de points avec régression LOESS (R) ou un diagramme de réseau (Python) — visuels impossibles à obtenir via les graphes natifs, notamment quand l'Arbre de décomposition limite la profondeur d'analyse et n'accepte qu'une seconde mesure (MICROSOFT, 2024alabelday).

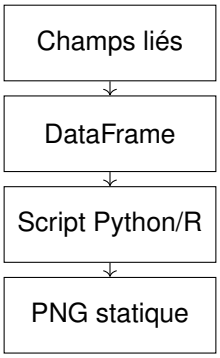

FIGURE 2.2 *: Pipeline d'exécution d'un visuel Python / R dans Power BI*

2.3.1 Atouts analytiques

L'accès direct aux bibliothèques open-source autorise statistiques avancées, clustering, apprentissage automatique, carte de chaleurs ou dendrogrammes ; le script peut pré-traiter les données ou entraîner un modèle avant de générer le graphique. À chaque rafraîchissement, l'exécution se relance automatiquement : l'utilisateur bénéficie ainsi de calculs que DAX ou Power Query ne permettent pas aisément, par exemple l'analyse de texte ou les séries temporelles non linéaires.

2.3.2 Limites techniques et fonctionnelles

Le résultat demeure une image statique — « les visualisations Python dans Power BI sont des bitmaps 72 DPI, sans interactivité directe » (MICROSOFT, 2025alabelday). Impossible donc d'initier un filtrage croisé depuis le graphique ; seul un filtre externe relance le script. Power BI transmet au script au plus 150 000 lignes et 100 colonnes, alloue 250 Mio de mémoire et impose un temps plafond de cinq minutes en local, soixante secondes dans le service ; la sortie PNG des visuels R est limitée à 2 Mio (MICROSOFT, 2025blabelday). Le runtime cloud, actuellement R 4.3.3 et Python 3.11, restreint le package compressé à 30 Mio, et l'affichage de ces visuels exige une licence Pro ou un workspace Premium : un utilisateur Free ne les verra pas.

2.3.3 Enjeux de sécurité et de maintenance

Power BI signale explicitement qu'un visuel Python ou R contient du code exécutable arbitraire (MICROSOFT, 2025alabelday). Dans un environnement professionnel, cela impose une gouvernance rigoureuse : le script, encapsulé dans le fichier binaire .pbix, échappe au suivi Git ; il doit donc être extrait dans un dépôt versionné afin de permettre la traçabilité et la revue par les pairs. De plus, le runtime cloud (Python 3.11, R 4.3.3) ainsi que ses bibliothèques sont mis à jour automatiquement chaque mois ; un fichier requirements.txt (ou renv.lock pour R) et des tests de régression sont indispensables pour vérifier la compatibilité après chaque release. Enfin, puisque ces scripts disposent d'un accès direct au système de fichiers et au réseau, il convient de restreindre leurs privilèges (comptes de service à droits minimaux, liste blanche de packages) afin de prévenir toute exfiltration de données ou exécution de code malveillant.

2.3.4 Bilan et positionnement stratégique

Les visuels Python et R offrent un recours efficace pour des analyses spécialisées ponctuelles, mais la nature statique du rendu, les contraintes de performance et les exigences de licence limitent leur usage à grande échelle. Une comparaison détaillée entre visuels natifs, Python/R et SDK figurera dans le tableau synthétique de la section 2.7 ; on y montrera que, pour un besoin récurrent et interactif, le développement d'un visuel SDK représente l'alternative la plus pérenne pour ECRINS SA.

2.4 SDK Custom Visuals : principes, sécurité, pipeline

Lorsque les visuels natifs ne suffisent plus et qu'un script Python ou R se révèle trop limité, la solution la plus aboutie consiste à créer un visuel complet à l'aide du Software Development Kit (SDK) de Power BI. Microsoft distribue ce SDK sous la forme du package npm powerbi-visuals-tools, dont la version stable 6.1.2 publiée le 26 mai 2025 sert de référence pour ce travail (MICROSOFT, 2025flabelday).

2.4.1 Principe général.

Un visuel Power BI est, en substance, une mini-application web encapsulée. Le développeur décrit, dans un fichier capabilities.json, les champs de données et les options de format qu'il souhaite exposer, puis implémente en TypeScript une classe qui respecte l'interface IVisual. Power BI invoque la méthode update(options) à chaque changement de filtre ou de données ; le code traduit alors les informations reçues en éléments DOM (SVG ou Canvas) à l'aide, par exemple, de D3.js.

2.4.2 Pipeline de développement

Le Power BI Custom Visuals SDK prescrit un flux de travail de référence : pbiviz new <NomDuVisuel> crée le squelette (manifeste, capabilities.json, fichiers TypeScript), puis pbiviz package –certification-audit génère l'archive .pbiviz tout en exécutant les contrôles réseau et mémoire recommandés (MICROSOFT, 2025glabelday). Depuis la dépréciation du serveur de développement intégré (2024), les tests interactifs s'effectuent dans un workspace développeur Power BI Service, avant qu'un package conforme ne soit importé dans un rapport ou soumis à AppSource pour certification (MICROSOFT, 2025dlabelday).

2.4.3 Cadre de sécurité.

Le code exécuté l'est toujours dans une iframe sandbox dont la politique de contenu interdit tout appel réseau non approuvé et toute évaluation dynamique de code (eval, new Function) ; le SDK force également le dessin à l'intérieur de la bounding box du visuel et n'expose au script que les champs explicitement liés par l'utilisateur (OKVIZ, 2022labelday). Les règles de certification 2025 précisent qu'un composant destiné à AppSource doit renoncer à tout flux sortant (**MicrosoftAPIv6CSP2025**).

2.4.4 Certification et limitations associées

La certification « Power BI Certified », réservée aux visuels publiés sur AppSource, conditionne l'inclusion du composant dans les exports PDF/PowerPoint, son rendu dans les rapports distribués par courriel et l'accès aux privileged APIs telles que FileDownload ou Licensing (MICROSOFT, 2024flabelday, 2025elabelday). Un visuel non certifié est exclu de ces scénarios et peut être bloqué si l'administrateur active l'option locataire « Add and use certified visuals only » (MICROSOFT, 2024glabelday).

2.4.5 Synthèse.

Le SDK ouvre la voie à des composants strictement sur mesure qui héritent de la totalité des interactions offertes par Power BI ; l'effort de développement et la discipline de gouvernance qu'il requiert constituent la contrepartie logique de cette flexibilité. La section 2.5 précise les choix d'outillage — TypeScript, D3.js et React optionnel — retenus pour réaliser les preuves de concept développées dans ce mémoire.

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.

2.5.1 TypeScript vs JavaScript.

Le SDK Power BI est conçu nativement pour TypeScript, sur-ensemble typé de JavaScript que Microsoft recommande pour les visuels personnalisés (MICROSOFT, 2025clabelday). 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, 2023labelday). 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 (INTERNATIONAL, 2024labelday). 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 (version 6.1.2 stable, 26 mai 2025 (MICROSOFT CORPORATION, 2025blabelday)).

2.5.2 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, 2019labelday). Cette approche bas niveau confère un contrôle complet sur chaque attribut visuel (couleur, position, animation), condition nécessaire à la réalisation de graphiques non standards répondant à des exigences métier spécifiques. D3 propose en outre un vaste ensemble de modules (générateurs de formes, projections cartographiques, échelles, layouts hiérarchiques) et s'appuie sur un écosystème mature d'exemples réutilisables. Les bibliothèques « haut niveau » telles que Chart.js ou Plotly raccourcissent le prototypage, mais leurs abstractions atteignent rapidement leurs limites pour des designs originaux ; de leur côté, les rendus basés sur canvas ou WebGL complexifient l'accessibilité et la netteté en cas de zoom. Le choix d'un SVG produit par D3 facilite enfin la mise à l'échelle et l'ajout d'attributs ARIA ou de balises `<title>`, conformément aux recommandations WCAG 2.2 applicables aux rapports Power BI (W3C, 2023labelday).

2.5.3 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 (SOURCE, 2024labelday). 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, 2024elabelday). L'empreinte ajoutée ($\approx$ 40 kB minifiés) reste compatible avec la limite de 2,5 MiB 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, réalisée ici sous Jest avec le preset officiel jest-pbi-visuals-preset. 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 ».

2.5.4 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.

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 différents de ceux de Power BI, tant sur le plan technique que dans leurs implications métier. Examiner ces modèles permet de situer les « visuels custom » Power BI dans un paysage concurrentiel plus large.

2.6.1 Tableau.

Tableau étend ses capacités à l'aide des Tableau Extensions, des applications Web encapsulées dans le tableau de bord par l'Extension API (SOFTWARE, 2024blabelday). Le composant, écrit en JavaScript et logé dans une iframe, peut introduire un graphique inédit, du write-back ou l'intégration d'un service tiers (SOFTWARE, 2024alabelday). Depuis la version 2019.4, les extensions s'exécutent par défaut en sandbox : aucun accès réseau n'est permis sans liste blanche explicite dans l'interface d'administration (SOFTWARE, 2025alabelday). Tableau délègue le support de ces briques à leurs éditeurs, si bien que chaque entreprise doit auditer la fiabilité de la source avant déploiement. Sur le plan financier, le coût d'entrée reste sensiblement supérieur à celui de Power BI : l'abonnement Creator est passé à 75 USD par utilisateur et par mois en juillet 2025 (SOFTWARE, 2025blabelday). La personnalisation n'est donc rentable que si l'organisation dispose déjà d'une base installée Tableau ou d'un budget élargi.

2.6.2 Qlik Sense.

Qlik propose les Visualization Extensions : objets écrits en HTML/JavaScript/CSS et déclarés par un manifeste .qext (QLIK, 2024alabelday). L'extension, quand elle respecte l'API Qlik, s'intègre comme un objet natif, profite du moteur associatif et réagit aux sélections (QLIK, 2024blabelday). Qlik mise sur une dynamique communautaire : de nombreux composants sont partagés via Qlik Branch sans validation officielle, la gouvernance incombant à l'administrateur qui doit installer manuellement l'archive sur le tenant SaaS. Commercialement, la déclinaison « Qlik Cloud Starter » débute à 200 USD/mois pour dix utilisateurs, la formule Standard à 825 USD et la Premium à 2 750 USD, chaque palier incluant un volume de données et un nombre maximal de créateurs (QLIK, 2025labelday). Le modèle reste avantageux quand l'entreprise possède une équipe front-end JavaScript capable de maintenir ces extensions.

2.6.3 Looker.

Looker, désormais composante de Google Cloud, autorise des visuels spécifiques via le Looker Custom Visualization SDK. Le développeur implémente la fonction updateAsync qui reçoit les données d'une explore Looker et restitue un rendu SVG ou Canvas (CLOUD, 2025labelday). Pour diffusion publique, Google impose une revue Marketplace : le code doit être hébergé sur GitHub et passer un contrôle de conformité avant publication (CLOUD, 2024labelday). Il reste toutefois possible de limiter l'usage à un tenant interne via l'interface Admin. La logique de Looker demeure plus centralisée : la personnalisation est permise, mais soumise à un contrôle

étroit et, surtout, à une tarification négociée au cas par cas qui dépasse largement les paliers Tableau ou Qlik ; les clients visent donc des gains élevés sur un périmètre de visuels custom restreint.

2.6.4 Lecture comparative.

Tableau et Qlik accordent à l'utilisateur avancé la liberté de charger localement une extension, tout en transférant la gouvernance de sécurité à l'entreprise ; Power BI et Looker exigent au contraire une validation centralisée — certification AppSource ou revue Marketplace — avant tout déploiement massif. Techniquement, les trois concurrents s'appuient, comme Power BI, sur le triptyque HTML/JS/CSS, mais Power BI se démarque par un SDK TypeScript/D3 structuré, une CLI d'audit intégrée et un vivier AppSource de presque 600 visuels au 1er août 2025 (MICROSOFT CORPORATION, 2025clabelday). Sur le plan économique, Power BI reste l'option la plus abordable pour industrialiser des visuels custom à grande échelle, tandis que Tableau, Qlik et surtout Looker réservent la personnalisation intensive aux environnements dont le budget justifie l'investissement initial et la maintenance continue.

2.7 Synthèse des écarts & opportunités

Après avoir étudié chacune des approches disponibles à l'intérieur même de Power BI, il est possible de dresser un panorama clair de leurs forces et de leurs limites, puis d'en déduire les cas d'usage qui conviennent le mieux à ECRINS SA. Quatre familles de solutions se distinguent : les visuels natifs, les scripts Python / R, les composants certifiés AppSource et les visuels développés avec le SDK.

2.7.1 Visuels natifs Power BI.

Fournis d'office, ces graphiques constituent le socle de la majorité des rapports ; ils sont maintenus par Microsoft, optimisés pour la performance et immédiatement compatibles avec l'ensemble des interactions (sélections croisées, filtres, export PDF / PPT, affichage mobile). Leur fiabilité et leur sécurité sont maximales, aucun code externe n'étant exécuté. Leur rigidité demeure cependant le principal frein : dès qu'un scénario exige un diagramme alluvial, une cartographie indoor ou un bullet chart spécifique, l'utilisateur doit composer avec les limites de paramétrage ou se tourner vers une autre voie.

2.7.2 Visuels Python / R.

L'exécution d'un script Python ou R ouvre la porte à l'immense écosystème de ces deux langages ; tout graphique réalisable dans Matplotlib, Seaborn, ggplot2 ou Plotly peut, en théorie, être intégré. La contrepartie tient dans la nature strictement statique du rendu : Power BI génère un PNG 72 DPI dépourvu d'interactivité et relance le script à chaque rafraîchissement, allongeant sensiblement le temps de calcul. Au-delà de 150 000 lignes transmises, la plateforme

tronque les données, tandis que le service Cloud limite l'image à 2 Mio et impose un runtime R 4.3.3 / Python 3.11 contrôlé par Microsoft. La démarche convient donc surtout au prototypage analytique rapide — rarement à un déploiement massif.

2.7.3 Visuels AppSource (certifiés).

AppSource, la galerie officielle de Microsoft, répertorie un peu plus de 540 visuels au 1er août 2025 (MICROSOFT CORPORATION, 2025clabelday). Chaque composant apparaît après une revue de sécurité et de performance ; il se comporte, pour l'utilisateur final, comme un visuel natif tout en couvrant des besoins de niche (Gantt, Waterfall amélioré, radar avancé). L'éditeur demeure toutefois responsable des mises à jour ; certains modules reposent sur un abonnement SaaS payant. L'approche constitue donc un juste milieu entre « prêt à l'emploi » et flexibilité.

2.7.4 Visuels SDK Power BI.

Le SDK — TypeScript, D3 et, au besoin, React — autorise la création d'un composant exactement aligné sur les spécifications métier : intégration complète avec les filtres, compatibilité mobile, export, diffusion via AppSource ou magasin organisationnel. Le revers est un effort de développement plus élevé et l'obligation d'une gouvernance logicielle continue : veille des versions, correctifs de sécurité, tests unitaires Jest et gestion du certificat X.509.

2.7.5 Vue d'ensemble comparative.

2.7.6 Conclusion.

Chaque approche comble un manque laissé par les autres : les visuels natifs offrent robustesse et simplicité, les scripts Python / R autorisent un prototypage statistique rapide au prix d'un rendu non interactif, AppSource fournit une solution plug-and-play pour des besoins spécialisés courants, et le SDK ouvre la voie aux composants entièrement sur mesure, moyennant un investissement en développement et en gouvernance. Pour ECRINS SA, la maîtrise du SDK constitue la clé méthodologique afin de répondre, à terme, aux demandes hors norme — un enjeu approfondi dans le chapitre 3.

TABLE 2.1 *: Comparaison factuelle des voies de personnalisation Power BI (état : août 2025)*

Critère	Visuels natifs	Scripts Python / R	Visuels AppSource (certifiés)	Visuels SDK internes (.pbiviz)
Cas d'usage typique	Tableaux de bord courants ; graphiques standards interactifs.	Analyses statistiques ou graphiques scientifiques spécifiques issus de code R/Python.	Graphiques spécialisés prêts à l'emploi (financiers, Gantt, KPI, etc.).	Visuels sur-mesure répondant à des exigences métiers uniques ou branding interne.
Limitations principales	Borné à la bibliothèque Microsoft ; pas de nouveau type sans extension.	Image statique ; 150 k lignes / 100 colonnes ; pas de sélection croisée ; timeout 5 min.	Pas d'appel réseau externe ; mises à jour soumises à re-certification ; version freemium possible.	Pas d'export PDF/PPT ni d'e-mailing ; optimisation et sécurité 100 % à charge du dev.
Situation d'utilisation recommandée	Choix par défaut tant que le visuel existe en natif.	POC data-science, prototypes analytiques ponctuels.	Quand un visuel AppSource couvre exactement le besoin et qu'une diffusion large est visée.	Quand aucune solution existante ne convient et qu'on dispose des ressources de développement.
Accessibilité (WCAG 2.2)	Conformité assurée par Microsoft.	Faible : rendu bitmap non lisible par lecteur d'écran.	Variable : dépend de chaque éditeur ; à tester avant adoption.	Dépend entièrement du développeur ; nécessite implémentation ARIA, focus clavier, contraste.
Difficulté d'utilisation	Très faible ; glisser-déposer.	Élevée ; compétences R/Python requises.	Moyenne ; configuration similaire aux visuels natifs.	Très élevée ; maîtrise TypeScript + D3 et API Power BI.
Performance (rendu / volumétrie)	Optimisée ; jusqu'à 30 M points selon le type.	Plus lente ; recalcul complet du PNG à chaque interaction.	Dépend du visuel ; généralement satisfaisant < 30 k points.	Variable ; plafonné à 30 k points, dépend de l'optimisation réalisée.
Coût (licence / dev)	Inclus dans Power BI.	Inclus ; demande temps de codage et maintien packages.	Souvent gratuit ; licence possible pour fonctionnalités avancées.	Frais de développement internes importants ; maintenance continue.
Réutilisabilité / industrialisation	Automatique ; présent partout.	Faible ; script encapsulé dans chaque rapport.	Élevée ; déploiement via AppSource ou store orga ; mises à jour automatiques.	Bonne en interne via store orga ; mises à jour manuelles ; partage externe impossible sans certification.

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 preuves de concept (PoC), la viabilité technique de visuels personnalisés. Il ne s'agit pas de livrer des composants prêts à la production, mais de valider un processus reproductible qui pourra servir de référence à de futurs développements.

3.1.1 Besoin marketing : carte des flux passagers aéroportuaire

Un client gestionnaire d'un aéroport international souhaite cartographier les flux de passagers dans l'aéroport. (Zones d'entrées, 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 carte de flux : des arcs reliant les points d'intérêt, dont l'épaisseur matérialise le volume de voyageurs, révèlent immédiatement les axes de déplacement dominants (GUO, 2009labelday). Une couche de carte de chaleur complétera la lecture en soulignant les « hot-spots ».

3.1.2 Besoin d'analyse hiérarchique : Sunburst de décomposition

Le second besoin a été formulé au cours d'un entretien tenu le 13 juin 2025 entre l'auteur, la professeure C. Bioley et un consultant décisionnel d'ECRINS SA. L'équipe souhaitait disposer d'un Decomposition Tree plus riche que le visuel natif de Power BI — celui-ci se révèle rapidement contraignant, tant par le nombre limité de mesures que par la faible marge de personnalisation offerte. Après analyse, le choix s'est porté sur un Sunburst, dont la structure radiale permet d'afficher plusieurs niveaux hiérarchiques en une seule vue et de comparer d'un coup d'œil les contributions relatives de chaque branche (J. T. STASKO, 2000labelday).

3.2 Critères de réussite

Les critères exposés dans cette section définissent le périmètre attendu d'un prototype fonctionnel. Il ne s'agit pas d'un produit prêt à la mise en production, mais d'une preuve de faisabilité suffisamment solide pour démontrer la validité des choix techniques. De fait, certaines exigences secondaires — telles que la compatibilité mobile, l'export PDF, la lecture bidirectionnelle (RTL) ou encore le packaging AppSource — sont volontairement exclues.

3.2.1 Performance

Le visuel doit garantir un temps de rendu fluide et stable lors d'un usage courant. Pour le composant Passenger-Flow Map, la cible est fixée à un affichage fluide pour un temps de rendu inférieur à 300 ms. Le second visuel, de type Sunburst, repose quant à lui sur une hiérarchie à 3 niveaux, avec un objectif de latence perceptible inférieure à 100 ms à chaque interaction (chargement initial, drill-down, survol). Cette borne constitue une référence implicite pour l'expérience utilisateur fluide.

3.2.2 Lisibilité minimale

Les visuels doivent respecter une lisibilité suffisante pour assurer la clarté des données présentées. Cette exigence comprend notamment le respect des contrastes de couleur AA selon la norme WCAG 2.2, la taille et le poids de police suffisants pour être interprétés sans effort, et la possibilité de naviguer au clavier sur les éléments interactifs de base (sélection, activation). À ce stade, seule une navigation `tabindex` et une activation clavier (Enter) sont exigées, sans aller jusqu'à l'implémentation complète d'un lecteur d'écran (screen reader) ou de la palette daltonienne. Lorsque cela apporte une information utile, des attributs ARIA (p. ex. aria-label et role) sont appliqués aux éléments véritablement actionnables afin de leur fournir un nom accessible ; à l'inverse, les formes purement décoratives sont exclues de la navigation et masquées aux technologies d'assistance.

3.2.3 Internationalisation essentielle

Les visuels doivent prendre en compte les spécificités locales des utilisateurs. L'affichage des nombres, pourcentages et dates est localisé dynamiquement selon la langue définie par l'environnement Power BI. Les locales fr-CH (français suisse) et en-US (anglais international) sont utilisées comme référence pour les tests. Les chaînes de texte affichées (infobulles, boutons, titres) sont externalisées et traduites dans ces deux langues, garantissant un usage bilingue de base. Cette fonctionnalité est conforme aux recommandations d'accessibilité de Power BI pour les environnements multilingues (Microsoft, 2024).

3.2.4 Qualité du code

Le code source doit être structuré de manière modulaire, commenté et testé selon les pratiques standard du développement front-end moderne. Le respect des bonnes pratiques de développement est assuré via une organisation logique des fichiers, une séparation claire des responsabilités (données, logique, rendu), et une couverture minimale de 70 % du code par des tests unitaires Jest. Ces tests sont conçus pour détecter les erreurs de logique ou les cas limites susceptibles d'apparaître en usage réel. L'objectif n'est pas d'atteindre un niveau de robustesse de production, mais de garantir une base stable, maintenable et compréhensible.

3.2.5 Intégration continue (CI/CD)

Le visuel est maintenu dans un dépôt GitHub associé à une chaîne CI légère, capable d'effectuer automatiquement la compilation, les vérifications et la génération de l'artefact .pbiviz. La CI contrôle la taille du fichier .pbiviz à l'aide d'un seuil interne fixé à $1\,\mathrm{MiB}$ (= 1 048 576 octets) ; au-delà, le pipeline échoue. Ce seuil vise à garantir des temps de chargement maîtrisés et une bonne hygiène de packaging ; il ne s'agit pas d'une contrainte officielle de Power BI Desktop. L'objectif est d'assurer la reproductibilité des builds et de détecter précocement toute dérive (taille, dépendances, erreurs de compilation).

En résumé, ces cinq critères de réussite garantissent un équilibre entre réalisme technique, effort de développement soutenable, et pertinence fonctionnelle pour les métiers d'ECRINS SA. Ils seront évalués au chapitre 7, selon une grille de validation fondée sur des mesures concrètes, des tests fonctionnels, et une revue de code.

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

La phase de tests s'appuie sur deux jeux de données de démonstration couvrant les cas d'usage clés. La définition exhaustive des champs, rôles Power BI et contraintes est centralisée en Annexe A1 (section A.1). Les artefacts du fond cartographique et des obstacles (schéma JSON abrégé et figure d'overlay) sont fournis en Annexe A2 (section A.2). La présente section se concentre sur les scénarios et critères de validation.

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

Cas d'usage simulés. Les essais portent sur le filtrage par tranches horaires, la distinction des catégories de flux (Départ/Arrivée) et l'activation d'un rendu de type carte de chaleur pour localiser les zones d'intensité. Les interactions croisées sont vérifiées : un filtrage temporel répercute immédiatement ses effets sur la carte des flux et inversement. Les info-bulles natives affichent les agrégations pertinentes conformément aux filtres actifs.

Validation de la faisabilité fonctionnelle. Les scénarios confirment la conformité du visuel aux comportements attendus de Power BI (filtrage, surbrillance croisée, info-bulles), ainsi que la bonne mise à jour des valeurs lorsque la plage horaire varie. L'activation de la carte de chaleur démontre la capacité à agréger spatialement les poids et à traduire l'intensité en densité visuelle.

3.3.2 Sunburst hiérarchique (Analyse multi-niveaux)

Cas d'usage simulés. Les tests couvrent la sélection de niveaux, les opérations de drill-down/drill-up, la mise en évidence de branches et l'application de seuils sur la mesure. L'interaction croisée est contrôlée : sélectionner un segment filtre les autres visuels du rapport, et réciproquement.

Validation de la faisabilité fonctionnelle. Les essais montrent que la hiérarchie et ses agrégations sont correctement représentées, que la navigation entre niveaux conserve les proportions attendues et que les filtres (catégoriels ou numériques) modifient les segments de manière cohérente avec les règles de filtrage de Power BI.

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 3.1 *: 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 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.

Des réunions de suivi régulières 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.

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).

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(MOGHEGHI, 2008labelday). La démarche de validation retenue est donc qualitative et pragmatique : elle combine une revue experte fonctionnelle et un mini-test utilisateur exploratoire.

3.5.1 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 carte de chaleur 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.

3.5.2 Démonstration interne.

En fin de développement, une courte évaluation sera réalisée par un collaborateur d'ECRINS SA. L'objectif est de recueillir un retour 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.

3.5.3 Mini-test utilisateur exploratoire.

En combinaison avec la démonstration interne, 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 (LALLEMAND, 2016labelday). 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.

3.5.4 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 | Playbook générique de développement des visuels Power BI

4.1 Objet et portée du playbook

Avant de détailler l'architecture des deux visuels choisis, il est impératif de poser le cadre de référence. Cette section formalise un playbook générique pour le développement de visuels personnalisés dans Power BI, destiné à servir de référentiel interne à ECRINS SA. Le chapitre poursuit un double objectif : d'une part, fournir une marche à suivre claire, étape par étape, afin que tout développeur puisse reproduire le processus de création d'un composant custom au moyen du SDK officiel de Microsoft ; d'autre part, encadrer ce processus par des bonnes pratiques et par la signalisation des pièges les plus fréquents, de façon à garantir l'industrialisation et la standardisation conformes aux exigences professionnelles de l'entreprise.

Le déroulé côté poste développeur est synthétisé en Figure 4.1 : initialisation du projet, exécution locale en mode développeur, packaging audité puis mise à disposition via le magasin organisationnel.

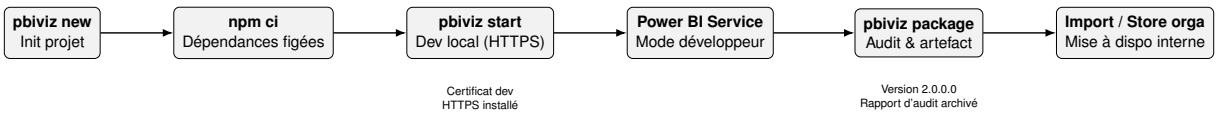

FIGURE 4.1 *: Cycle développeur pour un visuel Power BI : initialisation, exécution locale, packaging audité et distribution interne.*

Le style adopté est volontairement rigoureux et orienté vers la reproductibilité. Chaque phase clé — qu'il s'agisse de l'installation de l'environnement, de la génération du code initial, du débogage dans Power BI Service ou du conditionnement du fichier .pbiviz final — est explicitée et illustrée. Cette approche répond à trois besoins identifiés : capitaliser le savoir-faire existant, réduire le temps de montée en compétence des nouveaux arrivants, et diminuer les risques opérationnels lors du déploiement en production.

La progression s'appuie sur neuf sous-sections logiquement enchaînées. Après la présentation du SDK et le rappel des prérequis, le texte détaille la mise en place de l'environnement puis la création d'un composant minimal. Viennent ensuite la description de l'architecture générée et la procédure de débogage avec hot-reload dans Power BI Service. Un catalogue d'erreurs récurrentes et de correctifs éprouvés occupe la sous-section 4.8, tandis que la sous-section 4.9 propose une checklist de démarrage rapide. Enfin, la sous-section 4.10 assure la transition vers les deux prototypes développés — la Passenger-Flow Map à vocation marketing et le Sunburst budgétaire à visée financière.

En s'appuyant sur cette articulation progressive, le playbook fournit un fil conducteur unique qui évite les allers-retours inutiles et sécurise chaque livraison intermédiaire. Il doit être appliqué systématiquement aux futurs développements afin d'instaurer, au sein de l'équipe BI d'ECRINS SA, une culture de qualité logicielle commune.

4.2 Présentation du SDK Power BI Custom Visuals et de pbiviz

Le Software Development Kit (SDK) Power BI Custom Visuals publié par Microsoft fournit l'infrastructure nécessaire à la conception, à l'empaquetage et à la distribution de visuels additionnels au format .pbiviz. Au cœur de cet écosystème se trouve l'outil en ligne de commande pbiviz, lequel agit comme point d'entrée unique pour l'ensemble des opérations de scaffolding, de compilation et de débogage.

Concrètement, le SDK s'appuie sur l'écosystème Web standard : le code métier est écrit en TypeScript et manipulé à l'exécution comme du JavaScript transpilé ; la couche de présentation repose sur HTML, CSS / LESS et SVG, avec un recours fréquent à D3.JS pour la construction de scènes vectorielles. Cette architecture garantit une intégration transparente avec les navigateurs modernes tout en respectant le modèle de sandboxing imposé par Power BI : chaque visuel s'exécute dans un contexte isolé, sans accès direct aux données au-delà de celles explicitement fournies par le host.

L'outil pbiviz simplifie plusieurs étapes clés. D'abord, la génération d'un squelette de projet complet grâce à la commande pbiviz new évite la configuration manuelle d'un environnement Web ; le développeur obtient immédiatement une arborescence contenant le manifeste pbiviz.json, le fichier de capacités capabilities.json, les sources TypeScript ainsi que les ressources statiques. Ensuite, la phase de compilation est orchestrée par un serveur local déclenché via pbiviz start, lequel assure à la fois la transpilation du code, l'injection à chaud des modifications (hot-reload) et l'exposition du visuel sur https ://localhost :8080. Enfin, la commande pbiviz package produit le fichier .pbiviz final, prêt à être importé dans un rapport ou diffusé sur AppSource.

En 2025, l'évolution du service Power BI a déplacé la totalité du cycle de débogage vers la plateforme en ligne ; la prise en charge native du mode développeur dans Power BI Desktop a été abandonnée. Cette décision renforce la cohérence de l'expérience développeur, mais impose l'utilisation d'un compte Pro ou Premium Per User ainsi que l'installation d'un certificat SSL local pour sécuriser la communication entre le navigateur et le serveur pbiviz. Ces contraintes techniques seront examinées en détail dans les sous-sections suivantes.

Au-delà de la simplification opérationnelle, le SDK apporte un cadre de gouvernance : la présence d'un fichier manifeste prescrit la version de l'API, centralise les métadonnées auteur, référence les ressources externes et établit la signature numérique requise pour le déploiement en environnement contrôlé. En conséquence, l'adoption de ce kit constitue un premier levier d'industrialisation, puisqu'elle aligne tous les développements internes sur un standard unique, compatible avec les processus CI/CD abordés au chapitre 5.

À la lumière de ces éléments, la section 4.3 précisera les prérequis matériels et logiciels indispensables à l'installation de pbiviz, tandis que la section 4.4 décrira pas à pas la mise en place de l'environnement de développement.

4.3 Pré-requis techniques du développeur

Avant toute écriture de code, le poste du développeur doit satisfaire un ensemble de conditions incontournables. Leur conformité garantit la compatibilité avec la version 2025 du SDK Power BI Custom Visuals et élimine les blocages techniques susceptibles d'interrompre le cycle de développement.

Le premier composant à installer est Node.js, en version Long Term Support 18.x ou supérieure. Le runtime JavaScript est indispensable : l'outil pbiviz s'appuie sur l'écosystème Node pour exécuter ses commandes, télécharger les dépendances NPM et lancer le serveur local. Une version obsolète entraînerait des erreurs de compilation ou d'incompatibilité avec les bibliothèques récentes.

Une fois Node opérationnel, l'utilitaire en ligne de commande pbiviz doit être installé globalement. L'opération s'effectue par la commande : npm install -g powerbi-visuals-tools@latest. L'exécutable est alors ajouté au PATH ; son bon fonctionnement se vérifie en saisissant simplement pbiviz, ce qui affiche l'écran d'aide. Si la commande n'est pas reconnue, il convient de contrôler la variable d'environnement ou de réinstaller le paquet avec des droits élevés.

Le développement et surtout le débogage d'un visuel exigent un compte Power BI Pro ou Premium Per User. Seuls ces niveaux de licence ouvrent l'accès au mode développeur du service en ligne et autorisent le chargement d'un visuel non publié. Il est par ailleurs recommandé de créer un espace de travail dédié aux tests, de manière à isoler les prototypes des rapports de production.

Côté environnement intégré de développement, Microsoft préconise Visual Studio Code, qui conjugue légèreté, intellisense TypeScript, gestion Git intégrée et terminal embarqué. Les exemples et captures d'écran du présent playbook s'appuient sur VS Code, mais tout IDE disposant d'un support TypeScript (WebStorm, Visual Studio, etc.) demeure acceptable.

Durant la phase de débogage, Power BI charge le visuel via HTTPS, même lorsqu'il s'exécute en local. Il est donc nécessaire de générer un certificat SSL auto-signé et de l'approuver sur la machine. La commande pbiviz –install-cert crée et installe ce certificat ; sous Windows, elle ajoute l'entrée correspondante dans le magasin « Autorités de certification racines de confiance ». En environnement verrouillé, l'exécution de scripts PowerShell peut être bloquée ; il faudra alors solliciter l'administrateur ou importer manuellement le certificat généré.

Enfin, depuis 2024, Microsoft a centralisé le mode développeur sur la plateforme en ligne : la fonction de débogage local auparavant disponible dans Power BI Desktop a été supprimée. L'utilisateur doit activer le mode développeur dans les paramètres du service (<app.powerbi.com>)

avant de pouvoir insérer le conteneur spécial « Developer Visual ». Cette activation demeure personnelle et persistante, mais reste tributaire de la présence du certificat SSL et du serveur pbiviz en écoute sur localhost :8080. Une inobservation de l'un de ces prérequis se traduit immanquablement par un message d'erreur « Can't contact visual server ».

La satisfaction de ces exigences constitue la première étape vers un processus d'industrialisation fiable ; les sections suivantes décrivent la mise en place détaillée de l'environnement et la création du composant minimal.

4.4 Mise en place de l'environnement de développement

Une fois les prérequis matériels et logiciels réunis, la préparation effective du poste de travail s'effectue en quatre étapes séquentielles : l'installation de Node.js, l'ajout de l'outil pbiviz, la création du certificat SSL local et l'activation du mode développeur sur Power BI Service. Bien que la procédure ne doive être exécutée qu'une seule fois par machine, son respect intégral conditionne la réussite des itérations ultérieures.

La première étape consiste à installer Node.js en version Long Term Support. Le programme d'installation se télécharge depuis le site officiel de Node (« LTS 18.x » au moment de la rédaction). Sous Windows, l'assistant ajoute également npm et met à jour la variable d'environnement PATH. Un redémarrage peut être nécessaire pour que la commande

```
node -v
```

affiche la version installée.

Dès que Node est opérationnel, l'utilitaire Power BI Visuals Tools s'ajoute globalement au moyen de :

```
npm install -g powerbi-visuals-tools@latest
```

La présence de l'exécutable se vérifie par

```
pbiviz --help
```

qui doit retourner la liste des sous-commandes disponibles. L'absence de résultat indique soit un problème de droits d'installation, soit une configuration incorrecte du PATH. Dans un contexte d'entreprise, l'exécution de la commande avec des privilèges élevés – ou son inscription dans le registre système – peut s'avérer nécessaire.

La troisième opération vise à autoriser la communication sécurisée entre le service Power BI et le serveur local pbiviz. L'outil fournit son propre script :

```
pbiviz --install-cert
```

Sous Windows 10/11, la commande exploite l'API New-SelfSignedCertificate pour générer un certificat intitulé PowerBIVisualTest et l'importer dans le magasin « Autorités de certification racines de confiance ». Un dialogue peut apparaître pour demander la confirmation de l'ajout. Sur macOS et Linux, pbiviz s'appuie sur openssl ; il est donc impératif que cette dépendance figure dans le PATH. Un échec silencieux demeure le symptôme le plus courant en environnement verrouillé ; la solution implique généralement la modification temporaire de la politique d'exécution PowerShell ou l'installation manuelle du certificat généré.

La dernière étape se déroule dans le portail en ligne <app.powerbi.com>. Après authentification avec une licence Pro ou Premium Per User, l'utilisateur ouvre Settings → User settings → Developer et active l'option Power BI mode développeur. Dès validation, un pictogramme <> supplémentaire apparaît dans le volet Visualizations de tout rapport ouvert en mode Édition. Cette icône constitue l'unique porte d'entrée pour charger un visuel non publié.

À ce stade, le poste de développement est pleinement configuré : un certificat SSL approuvé sécurise la connexion sur https ://localhost :8080, pbiviz est disponible globalement et le mode développeur est actif côté service. Les sections suivantes décrivent la génération du squelette de visuel minimal et les premières vérifications fonctionnelles.

4.5 Création d'un composant minimal avec pbiviz new

La première activité concrète du développeur consiste à générer la base du projet en recourant à la commande pbiviz new. Cette opération automatise la création de l'arborescence standard, du manifeste et des fichiers sources, évitant ainsi toute configuration manuelle d'un environnement Web complété par Webpack.

4.5.1 Génération du squelette de projet

Depuis un terminal positionné dans le répertoire de travail, on invoque :

```
pbiviz new SampleVisual
```

Le paramètre SampleVisual définit le nom interne du composant ; il doit respecter la casse Pascal et rester unique au sein du tenant Power BI afin d'éviter les collisions de GUID. L'outil crée alors un dossier homonyme contenant l'arborescence complète ; le terminal affiche la réussite de chaque étape (création de pbiviz.json, capabilities.json, visual.ts, etc.). Le développeur ouvre aussitôt ce dossier dans Visual Studio Code pour explorer la structure générée.

4.5.2 Installation des dépendances NPM

À l'intérieur du répertoire du projet, l'exécution de :

```
npm install
```

télécharge et installe l'ensemble des bibliothèques requises : l'API Power BI Visuals, TypeScript, D3.JS le cas échéant, ainsi que les définitions de types. La présence d'un dossier node_modules confirme le succès de l'opération. Il est recommandé de contrôler la version du paquet powerbi-visuals-api dans package.json pour s'assurer qu'elle correspond à la version cible de l'environnement Power BI.

4.5.3 Premier lancement en mode développeur

Le projet minimal peut désormais être compilé et servi localement. La commande suivante :

```
pbiviz start
```

construit le bundle JavaScript, démarre un serveur HTTPS sur https ://localhost :8080 et active le rechargement dynamique. La console affiche le port d'écoute et le chemin du certificat utilisé ; tant que le processus demeure actif, il publie les mises à jour à chaque sauvegarde de fichier.

4.5.4 Vérification dans Power BI Service

Le développeur se rend ensuite sur le service Power BI, ouvre un rapport de test en mode Édition, puis insère le conteneur « Developer Visual » à l'aide de l'icône <>. Power BI établit une connexion sécurisée vers localhost :8080, télécharge le visuel et instancie la classe Visual. Le composant s'affiche immédiatement avec son contenu par défaut (un compteur d'appels à update() dans le squelette). L'absence de message d'erreur confirme la bonne configuration de l'environnement.

4.5.5 Cycle itératif de modification

À partir de ce point, toute modification du code visual.ts ou des styles visual.less suivie d'une sauvegarde déclenche une recompilation automatique ; le visuel se rafraîchit alors dans le rapport sans qu'il soit nécessaire de recharger la page. Cette boucle courte – écrire, sauvegarder, observer – constitue la pierre angulaire de la productivité et du débogage. Il est toutefois conseillé de surveiller la console du navigateur (F12) afin de détecter rapidement les exceptions JavaScript ou les avertissements de performance.

4.5.6 Arrêt propre du serveur

Le serveur local se termine par la combinaison Ctrl + C. L'outil demande une confirmation éventuelle ; accepter libère le port 8080 et clôt la session SSL. Le développeur veillera à interrompre proprement le processus avant de changer de branche Git ou de mettre à jour les dépendances, faute de quoi un conflit de port ou de certificat pourrait survenir lors du redémarrage.

Une fois cette étape validée, l'équipe dispose d'un composant minimal fonctionnel, prêt à accueillir la logique métier, le mapping des données et les options de formatage décrits dans les sous-sections suivantes.

4.6 Structure des fichiers générés par pbiviz

L'invocation de pbiviz new crée automatiquement une arborescence normalisée, conçue pour séparer les métadonnées du visuel, son code source et ses ressources statiques. Cette organisation reflète le cycle de vie imposé par le host Power BI et facilite la maintenance future.

4.6.1 Arborescence racine

Le projet se présente immédiatement sous la forme suivante :

```
SampleVisual/
|-- src/                # code TypeScript du visuel
|   |-- visual.ts
|   `-- settings.ts
|-- assets/             # icones et images integrees
|   `-- icon.png
|-- style/              # feuilles de style LESS ou CSS
|   `-- visual.less
|-- capabilities.json   # contrat de donnees et format
|-- pbiviz.json         # manifeste et metadonnees
|-- package.json        # dependances npm
|-- tsconfig.json       # compilation TypeScript
`-- ...
```

Cette disposition garantit qu'un clean build – lors d'un déploiement automatisé, par exemple – peut s'effectuer en isolant sans ambiguïté le source code des artefacts binaires produits.

4.6.2 capabilities.json

Le fichier capabilities.json formalise le contrat qu'énonce le visuel à Power BI. D'une part, il décrit les rôles de données attendus : dimensions catégorielles, mesures numériques, hiérarchies ou tables. D'autre part, il définit l'ensemble des objets de formatage qui apparaîtront dans le volet « Format » de l'interface. Chaque propriété est typée, accompagnée d'un identifiant interne et d'une étiquette traduisible. Le DataView remis au code du visuel se conforme strictement à cette déclaration ; toute incohérence se traduit par un DataView vide ou partiel, d'où l'importance de conserver ce fichier comme source de vérité.

4.6.3 visual.ts

visual.ts contient la classe principale Visual qui implémente l'interface IVisual. La méthode update(...) orchestre le rendu : récupération des données, transformation, puis construction ou mise à jour du DOM – généralement via D3.JS. Le fichier accueille aussi les méthodes de cycle de vie (enumerateObjectInstances, destroy) et les gestionnaires d'événements.

4.6.4 settings.ts

Le rôle de settings.ts est de typer les options exposées dans capabilities.json. En héritant de DataViewObjectsParser, le développeur déclare des classes qui regroupent les propriétés de formatage : couleurs, booléens, tailles, limites, etc. Toute modification du schéma de format requiert donc un ajustement symétrique dans ce fichier, faute de quoi des valeurs undefined risquent de perturber le rendu.

4.6.5 pbiviz.json

pbiviz.json joue le rôle de manifeste. Il regroupe : le nom interne du visuel, son display name, le GUID unique, la version de l'API ciblée, les informations d'auteur, l'icône, la référence au fichier capabilities.json et à la feuille de style, ainsi que les ressources additionnelles éventuelles. Lors du package build, ce fichier pilote l'inclusion des fichiers et la signature numérique.

4.6.6 Interdépendance et bonnes pratiques

La configuration tripartite formée par capabilities.json, settings.ts et visual.ts impose un principe fondamental : toute modification dans l'un doit se refléter dans les autres. Il est établi une règle de revue de code qui exige que tout pull request modifiant le contrat de données ou le panneau de format inclue systématiquement l'adaptation correspondante dans les classes TypeScript et les tests unitaires. Ce couplage strict constitue le premier rempart contre les régressions fonctionnelles.

4.7 Débogage et hot-reload dans Power BI Service

Depuis 2024, Microsoft a entièrement transféré le mode développeur vers la plateforme en ligne ; Power BI Desktop ne prend plus en charge le chargement dynamique d'un visuel en cours de développement. La présente sous-section décrit la procédure opérationnelle à suivre pour tester un composant en conditions quasi réelles, détecter les anomalies et réduire le temps d'itération entre deux modifications de code.

4.7.1 Insertion du visuel développeur

Une fois le serveur local lancé via :

```
pbiviz start
```

le développeur se connecte à l'adresse <app.powerbi.com> avec un compte disposant de la licence requise et ouvre un rapport de test en mode « Édition ». Dans le volet Visualizations, l'icône <> devient visible à condition d'avoir activé le Developer mode dans les paramètres utilisateur. Un simple clic insère sur la page un conteneur blanc relié à https ://localhost :8080. Si la connexion SSL s'établit correctement, Power BI télécharge le bundle JavaScript du visuel et exécute immédiatement la méthode update().

4.7.2 Premier affichage et état d'attente

Le squelette généré par pbiviz new affiche, par défaut, un compteur du nombre d'appels à update(). Tant qu'aucun champ de données n'est associé, le visuel fonctionne en « mode attente » : il se contente d'occuper son rectangle et d'incrémenter le compteur lors des évènements resize ou refresh. Cette phase est un marqueur précieux : elle confirme que le pipeline de chargement fonctionne et que l'environnement a reconnu le certificat.

4.7.3 Association de données et contrôles de base

Le développeur peut ensuite glisser-déposer un ou plusieurs champs depuis le panneau Fields vers le visuel. Chaque interaction déclenche un nouvel appel à update(), dont les paramètres contiennent désormais un objet DataView. L'exploration de cette structure dans la console du navigateur permet de vérifier :

1. la présence des rôles de données attendus ;
2. l'exactitude des types (catégorie, valeur, hiérarchie) ;
3. la cohérence des index, notamment lorsque plusieurs tables sont en jeu.

Une pratique courante consiste à loguer la taille des tableaux categorical.values et categorical.categories afin d'anticiper les surcharges mémoire dans le cas de jeux de données volumineux.

4.7.4 Hot-reload automatique

Tant que le processus pbiviz start reste actif, toute modification du fichier visual.ts, de la feuille de style ou d'un template HTML déclenche une recompilation suivie d'un rechargement silencieux du visuel dans Power BI. Aucun refresh manuel n'est requis, à condition que le développeur demeure sur l'onglet où le rapport est ouvert. Si la modification touche pbiviz.json ou capabilities.json, Power BI exige toutefois un rechargement complet de la page, car ces métadonnées sont chargées au démarrage du conteneur.

4.7.5 Diagnostic des erreurs fréquentes

Deux catégories de messages d'erreur apparaissent régulièrement :

« Can't contact visual server » indique l'échec de la connexion HTTPS. Les causes probables sont : certificat non approuvé, serveur stoppé ou port 8080 occupé par un autre processus.

Exceptions JavaScript relèvent d'un bug dans le code du visuel. Le stack trace est affiché dans la console et se complète d'une mention « onUpdate » si l'erreur survient durant le rendu.

La meilleure approche consiste à garder la console (F12) ouverte, à activer le filtre « Errors » et à insérer temporairement des console.time() / console.timeEnd() autour des blocs de calcul gourmand afin de détecter les goulots d'étranglement.

4.7.6 Nettoyage de la session et reprise

Lorsque les tests sont terminés, l'arrêt du serveur local par Ctrl + C libère le port et met fin à la session SSL. Il est recommandé de supprimer le conteneur développeur du rapport de test afin d'éviter toute tentative de connexion ultérieure à un serveur inexistant. Un nouveau cycle peut commencer en relançant pbiviz start puis en réinsérant l'icône <>. Cette discipline pre-prod limite les confusions, notamment lorsque plusieurs développeurs travaillent sur des visuels distincts.

À l'issue de cette phase de débogage interactif, le visuel peut être considéré comme stabilisé. L'étape suivante consiste à répertorier les erreurs courantes observées et à documenter systématiquement leurs correctifs, démarche présentée dans la section 4.7.

4.8 Erreurs courantes et contournements

Malgré l'application rigoureuse du playbook, plusieurs incidents récurrents peuvent survenir au cours du développement d'un visuel Power BI. Les paragraphes suivants recensent les plus fréquents, accompagnés de leur diagnostic et du correctif validé chez ECRINS SA. Cette capitalisation d'expérience vise à réduire le temps de résolution et à homogénéiser les pratiques de support interne.

4.8.1 pbiviz introuvable après installation.

La commande renvoie « not recognised ». Dans la quasi-totalité des cas, le paquet n'a pas été ajouté au PATH global ou l'installation a été lancée sans privilège suffisant. La vérification commence par node -v, puis par une réinstallation en mode administrateur : npm install -g powerbi-visuals-tools@latest. En environnement verrouillé, il est toléré d'utiliser npx pbiviz localement, mais la résolution définitive passe par la correction du PATH système.

4.8.2 Échec de connexion HTTPS : « Can't contact visual server ».

Power BI ne parvient pas à joindre https ://localhost :8080. Les motifs les plus fréquents sont : certificat non installé ou non approuvé, serveur pbiviz start arrêté, port 8080 occupé, pare-feu bloquant le trafic. La procédure standard consiste à : i) relancer pbiviz –install-cert, ii) ouvrir https ://localhost :8080/assets dans le navigateur pour accepter le certificat, iii) vérifier la disponibilité du port via netstat ou Get-NetTCPConnection.

4.8.3 Icône développeur absente dans le volet des visuels.

L'option mode développeur n'est pas activée ou l'utilisateur n'a pas de licence adéquate. La solution est d'ouvrir Settings → User settings → Developer et de basculer l'interrupteur, puis de rafraîchir la page du rapport. Si l'icône demeure invisible, il faut confirmer la présence d'une licence Pro ou PPU et l'absence de restrictions administratives au niveau tenant.

4.8.4 DataView vide ou partiel malgré la sélection de champs.

Lorsque options.dataViews est undefined, le plus souvent les champs glissés ne correspondent pas aux rôles déclarés dans capabilities.json. Le correctif réside dans la vérification du mapping et, au besoin, l'ajout de l'attribut "required" : false pour les rôles optionnels. Côté code, la méthode update() doit toujours vérifier la présence de dataViews[0] et afficher un message d'invite si les données sont insuffisantes.

4.8.5 Erreur PowerShell lors de pbiviz –install-cert.

Sur certains postes verrouillés, la politique d'exécution bloque le script interne. Le message typique mentionne « execution of scripts is disabled ». Deux solutions : modifier temporairement la politique pour l'utilisateur courant (Set-ExecutionPolicy RemoteSigned -Scope CurrentUser) ou générer manuellement un certificat avec New-SelfSignedCertificate puis placer les fichiers dans le répertoire attendu par pbiviz.

4.8.6 Visuel fonctionnel en mode développeur mais défectueux après empaquetage.

Le fichier .pbiviz importé n'affiche rien ou déclenche une erreur générique. Dans la majorité des cas, le problème provient d'une incohérence de version API ou d'un GUID en doublon. Avant la commande pbiviz package, il est recommandé de mettre à jour pbiviz.json ("apiVersion") et d'exécuter un test d'import dans un nouveau rapport vierge.

4.8.7 Avertissements TypeScript et incompatibilités de types.

Lors du lancement de pbiviz start, des erreurs de compilation bloquent la chaîne. La démarche consiste à : i) identifier l'origine (généralement une mise à jour de powerbi-visuals-api), ii) aligner la version cible dans package.json et dans pbiviz.json, iii) corriger les signatures des méthodes concernées. Un npm ci suivi d'un pbiviz start garantit la cohérence du lockfile.

4.8.8 Taille de package anormalement faible.

Un .pbiviz de quelques kilo-octets révèle souvent l'exclusion involontaire d'une bibliothèque externe. Depuis la version 3 du SDK, l'attribut externalJS n'est plus supporté ; les dépendances doivent être intégrées via Webpack. La vérification s'effectue en inspectant le dossier dist/ après empaquetage et en contrôlant la présence d'un fichier JavaScript consolidé de plusieurs centaines de kilo-octets.

La consolidation de cette liste dans un référentiel interne — baptisé cookbook des erreurs — permet aux développeurs de gagner en autonomie et d'écourter les échanges avec le support. Chaque nouvelle occurrence doit être consignée selon le modèle « symptôme – diagnostic – résolution », afin d'alimenter en continu la base de connaissances d'ECRINS SA.

4.9 Checklist de démarrage rapide

La checklist ci-dessous offre un aide-mémoire synthétique pour préparer un poste de travail et lancer un premier visuel Power BI en moins d'une heure. Chaque point renvoie à la sous-section détaillée correspondante ; son respect chronologique diminue significativement le temps de mise en route et prévient les blocages les plus courants.

Étape 1 : Installer Node.js ($\geq$ v18 LTS) : télécharger l'installateur officiel, exécuter avec les options par défaut, puis vérifier la version (§ 4.3).

Étape 2 : Ajouter l'outil pbiviz : exécuter npm install -g powerbi-visuals-tools@latest et confirmer la disponibilité de l'exécutable via pbiviz –help (§ 4.3).

Étape 3 : Générer le certificat SSL local : lancer pbiviz –install-cert, accepter l'importation dans le magasin racine et tester l'URL https ://localhost :8080/assets (§ 4.4).

Étape 4 : Activer le mode développeur : sur <app.powerbi.com>, ouvrir Settings $\rightarrow$ User settings $\rightarrow$ Developer et basculer l'interrupteur (§ 4.4).

Étape 5 : Créer le squelette du visuel : dans un dossier de travail, exécuter pbiviz new <NomDuVisuel> puis npm install pour récupérer les dépendances (§ 4.5).

Étape 6 : Démarrer le serveur local : à la racine du projet, exécuter pbiviz start et laisser la console ouverte pour compiler à chaud (§ 4.5).

Étape 7 : Insérer le conteneur développeur : dans un rapport de test ouvert en mode Édition, cliquer sur l'icône <> et vérifier que le visuel se charge sans erreur (§ 4.7).

Étape 8 : Associer des champs de données : glisser-déposer au moins un rôle requis, observer la mise à jour du compteur ou du rendu et contrôler la structure DataView via la console JavaScript (§ 4.7).

Étape 9 : Itérer sur le code : modifier visual.ts ou visual.less, sauvegarder et valider le hot-reload. Si nécessaire, recharger la page après changement de capabilities.json (§ 4.7).

Étape 10 : Consigner les anomalies : toute erreur rencontrée doit être ajoutée au cookbook interne avec son correctif, conformément au modèle « symptôme – diagnostic – résolution » (§ 4.8).

Étape 11 : Empaqueter pour validation hors mode dev : lorsque le prototype est stable, exécuter pbiviz package, importer le fichier .pbiviz dans un rapport vierge et répéter les tests de données.

Le respect de cette séquence permet de sécuriser la livraison d'une première version fonctionnelle tout en capitalisant immédiatement les bonnes pratiques et les leçons apprises.

4.10 Mise en pratique du playbook : introduction aux prototypes Passenger-Flow Map et Sunburst budgétaire

Le présent chapitre a posé le socle méthodologique nécessaire à la création de tout visuel Power BI personnalisé : installation de l'environnement, génération du squelette, cycle de débogage et gestion des erreurs. Cette démarche générique constitue dorénavant la référence interne d'ECRINS SA pour l'industrialisation des composants d'analytique visuelle.

Les sections suivantes (5.A et 5.B) appliquent concrètement ce playbook à deux prototypes distincts :

1. le Passenger-Flow Map, destiné aux équipes marketing aéroportuaires pour cartographier les parcours passagers ;
2. le Sunburst budgétaire, orienté analyse, qui combine arbre de décomposition et diagramme en anneau pour une exploration plus riche des données hiérarchiques.

Chaque prototype suit la même structure : architecture, pipeline de rendu, définition du DataView, interactions utilisateur, tests et conformité aux exigences d'accessibilité. Ce découpage illustre la réutilisabilité du cadre défini ci-dessus, tout en mettant en évidence les choix technique.

En nous appuyant sur ce fil conducteur, nous démontrerons dans le chapitre 6 comment automatiser la chaîne CI/CD, signer numériquement les packages et déployer les visuels dans un environnement gouverné. Les enseignements tirés des prototypes serviront alors de levier d'amélioration continue pour la bibliothèque interne de composants visuels d'ECRINS SA.

5 | Développement des visuels Power BI personnalisés

5A Passenger-Flow Map

5A.1 Principes architecturaux et structure modulaire

Le visuel Passenger-Flow Map a été développé selon une architecture hexagonale modulaire. Ce paradigme « ports & adapters » isole la logique métier des interfaces d'entrée-sortie, limitant ainsi le couplage structurel et facilitant à la fois les tests unitaires et l'évolution incrémentale du code. Concrètement, le composant est scindé en plusieurs modules faiblement couplés, chacun assumant une responsabilité unique et communiquant via des interfaces abstraites. Cette séparation stricte entre le cœur fonctionnel et les périphériques – essentiellement les données Power BI, le rendu DOM et les interactions utilisateur – garantit la cohérence interne et l'interchangeabilité future des composants.

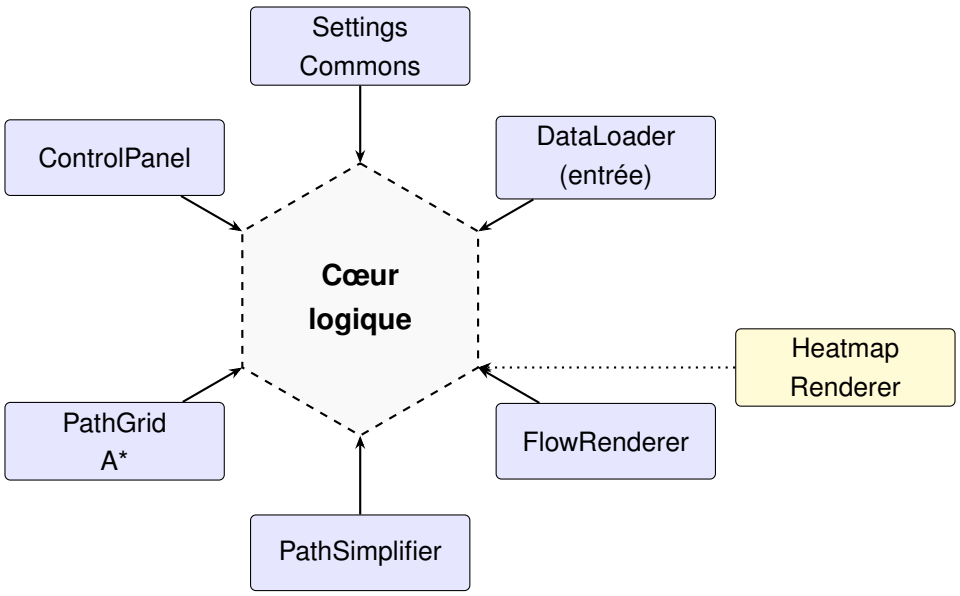

FIGURE 5.1 *: Architecture hexagonale du visuel Passenger-Flow Map.*

La Figure 5.1 illustre cette organisation modulaire. L'hexagone central représente le cœur « Visual Shell » (visual.ts), contrôleur qui orchestre l'initialisation, la lecture des données, les filtres et le rendu. Autour, sept adaptateurs périphériques s'alignent sur les arêtes : (1) DataLoader, adaptateur d'entrée qui convertit le DataView brut de Power BI en structures internes ; (2) PathGridA*, calculateur de trajectoire optimale fondé sur l'algorithme A* ; (3) PathSimplifier, filtre géométrique réduisant la complexité des chemins ; (4) FlowRenderer, moteur de rendu des flux qui génère et stylise les tracés SVG ; (5) HeatmapRenderer, module optionnel dédié au rendu

d'une carte de chaleur exprimant la densité de trafic ; (6) ControlPanel, panneau ui permettant filtres et interactions utilisateur ; (7) Settings / Commons, regroupement des paramètres exposés dans le Format Pane et des utilitaires transverses (constantes, journalisation).

Chacun de ces modules correspond soit à un port de l'application hexagonale, soit à un adaptateur qui implémente ce port pour une technologie donnée. Ainsi, DataLoader constitue le port d'entrée des données, tandis que FlowRenderer et HeatmapRenderer se comportent comme adaptateurs de sortie graphique. De même, ControlPanel joue le rôle d'adaptateur d'interface utilisateur, parfaitement découplé de la logique de calcul interne. Tel que le formalise (COCKBURN, 2008labelday), cette organisation modulaire assure une séparation nette entre la computation interne et l'environnement externe, condition essentielle pour la maintenabilité de long terme et pour la conformité aux bonnes pratiques d'ingénierie logicielle.

5A.2 Chaîne de traitement des données — du DataView au DOM

Le fonctionnement interne du visuel Passenger-Flow Map repose sur un pipeline unidirectionnel qui transforme le DataView fourni par Power BI en éléments SVG rendus dans le navigateur. À chaque invocation de la méthode `update(options)` par le host Power BI, six étapes séquentielles sont exécutées sans rétroaction directe du DOM vers la logique métier, conformément aux meilleures pratiques de Microsoft pour les custom visuals (CORPORATION, 2024labelday).

Le format des obstacles (extrait abrégé) et la figure d'alignement du fond sont centralisés en annexe A2 (section A.2). Les fichiers capabilities.json et pbiviz.json du visuel Passenger-Flow Map sont fournis dans le dépôt source ; cette sous-section se limite aux principes de mapping (rôles, agrégations) et aux options exposées dans le volet Format.

Dans un premier temps, DataLoader interprète le schéma tabulaire du DataView et instancie pour chaque ligne un objet Flow contenant l'identifiant unique, les coordonnées d'origine et de destination, le volume de passagers et, le cas échéant, les attributs temporels et catégoriels. Cette étape garantit que toutes les transformations ultérieures s'appuient sur un format strictement typé, réduisant ainsi le risque d'erreur à l'exécution.

La seconde étape mobilise PathGrid, qui élabore une grille de navigation couvrant la zone de l'aéroport, marque les cellules occupées par des obstacles comme non franchissables et applique l'algorithme A* pour déterminer l'itinéraire optimal entre chaque couple source–cible. Le résultat est une polyligne potentiellement dense que PathSimplifier vient aussitôt épurer en supprimant les sommets angulaires inférieurs.

Lors de la quatrième étape, FlowRenderer convertit chaque chemin simplifié en un tracé SVG. Si la polyligne comporte plus de deux points, une interpolation Catmull–Rom est appliquée afin de lisser visuellement la trajectoire, tandis que l'épaisseur du trait est déterminée par une fonction linéaire basée sur le volume de passagers (bornes par défaut : 2 px–8 px). Une échelle de couleur continue, paramétrable dans le groupe Colors du Format Pane, encode simultanément l'intensité du trafic.

Ensuite, la classe Visual orchestre l'insertion des éléments SVG au sein du conteneur attaché au visuel. Elle veille à détruire le canevas précédent avant chaque rendu pour éviter l'accumulation d'éléments, pratique essentielle à la prévention des fuites de mémoire dans le cycle de hot-reload. Elle déclenche également la cinquième étape : la mise à jour du ControlPanel, lequel ajuste son état interne et rafraîchit les statistiques agrégées (nombre de flux actifs, total passagers et moyenne par flux) à partir de la collection actuellement visible.

Enfin, une sixième étape facultative intervient lorsque l'utilisateur active l'option showHeatmap. HeatmapRenderer agrège alors les volumes dans une matrice régulière, applique un flou gaussien pour produire une représentation continue de la densité et insère cette couche dans un groupe <g> superposé aux tracés individuels. L'alternance entre mode détaillé et carte de chaleur s'effectue sans rechargement complet grâce à la stratégie de rendu conditionnel implémentée dans Visual, qui conserve les éléments inutilisés dans le DOM mais les masque via des attributs visibility afin de réduire le coût de ré-affichage.

L'unidirectionnalité de ce pipeline simplifie le raisonnement sur l'état de l'application : chaque cycle `update()` débute avec un canevas propre et se termine dans un état entièrement déterminé par les données et les paramètres courants. Cette propriété est déterminante pour la fiabilité du visuel, tout particulièrement lorsque ECRINS SA devra assurer le dépannage ou l'évolution de composants déjà déployés en production.

5A.3 Calcul des trajectoires de flux : principe et réglages

Le visuel Passenger-Flow Map doit relier un point de départ et un point d'arrivée sans traverser les zones interdites. Le module PathGrid s'en charge en trois étapes : préparer le plan, trouver un chemin, puis simplifier le tracé pour l'affichage.

Préparation du plan. À l'ouverture, PathGrid pose une grille régulière sur le plan de l'aéroport (par exemple 200×200 cases). Les coordonnées des flux sont ramenées sur cette grille et les zones « mur, contrôle, accès interdit » décrites dans obstacles.json — schéma extrait et figure d'overlay fournis en annexe A2 (section A.2) — sont marquées comme non franchissables. Cette étape garantit qu'aucun tracé ne coupe un obstacle.

Recherche du chemin. Le chemin est calculé avec un algorithme de plus court trajet (A*). Concrètement, il explore les cases voisines jusqu'à rejoindre la destination, en évitant les « coupes de coin » qui rasent un obstacle. Si la destination est inaccessible, le visuel signale le problème et affiche une simple ligne droite pour rester lisible.

La Figure 5.2 illustre le principe appliqué : A* évite les cellules marquées comme obstacles, puis un lissage supprime les angles inutiles avant le rendu SVG.

Allègement du tracé. Le chemin brut peut contenir trop de points pour un rendu fluide. PathSimplifier enlève alors les petits angles inutiles selon un seuil réglable (smoothness). L'objectif est d'obtenir une courbe propre, légère à dessiner, sans changer la trajectoire perçue.

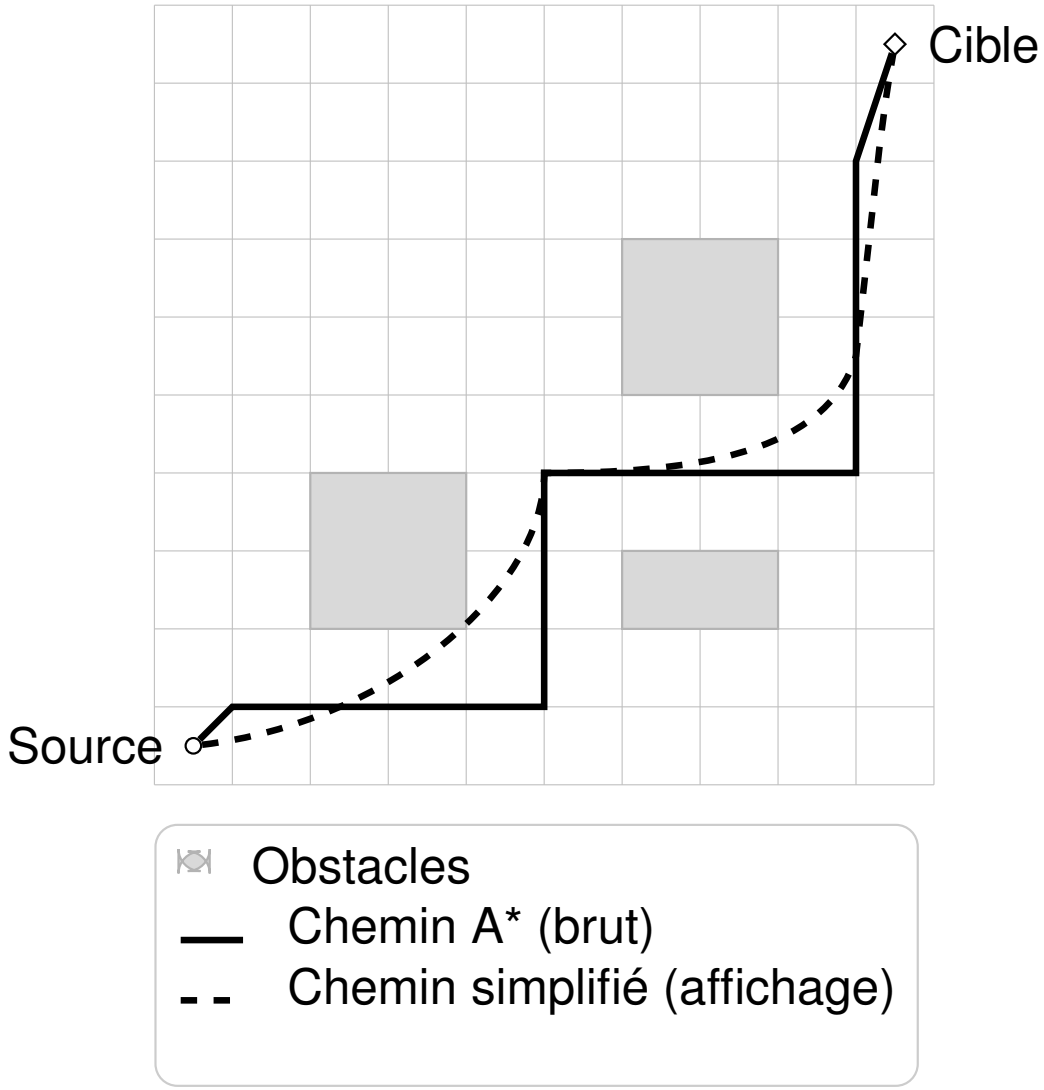

FIGURE 5.2 *: Évitement d'obstacles et simplification de tracé (A* → rendu).*

Réglages dans le Format Pane. Trois options pilotent le résultat : curvature (arrondi de la courbe), smoothness (niveau de simplification) et avoidObstacles (activation de l'évitement). Elles sont déclarées dans PathSettings (capabilities.json) et lues à chaque `update()` via VisualSettings. Toute modification relance le calcul et le rendu.

5A.4 Rendu graphique et animation des flux

Le module FlowRenderer transforme chaque flux (chemin déjà simplifié) en formes SVG. Ce format s'affiche nativement dans les navigateurs et se prête bien aux styles et animations avec D3. Lorsque le tracé comporte plusieurs segments, il est lissé pour obtenir une courbe fluide. Le réglage curvature contrôle ce lissage : faible valeur = segments anguleux, valeur élevée = courbe arrondie. Les coordonnées X1,Y1,X2,Y2 sont exprimées en pixels et alignées sur l'overlay de référence ; le format des obstacles et la figure de superposition sont documentés en annexe A2 (section A.2).

Encodage visuel. L'épaisseur du trait représente le volume de passagers. Elle est automatiquement ramenée dans une plage lisible (en pratique, de quelques pixels à un maximum fixé) pour éviter les saturations. La couleur suit une échelle continue du « faible » au « fort » trafic (lowTraffic $\rightarrow$ highTraffic), modifiable dans le Format Pane. L'opacité initiale (0,7 par défaut) est également réglable.

Interactions. Au survol, le flux est mis en avant : opacité portée à 1 et trait légèrement élargi. Le service d'info-bulle natif de Power BI (TooltipServiceWrapper) affiche les détails utiles : identifiant, volume, heure et catégorie, ce qui garantit une cohérence visuelle avec le reste du rapport.

Animation du sens. Pour indiquer la direction, une petite particule se déplace le long du tracé en boucle. Sa vitesse dépend de la longueur du chemin et d'un paramètre animationSpeed (valeur par défaut : 2 s pour un trajet moyen), de façon à rendre l'effet perceptible sans surcharger la lecture (MEULEMANS et al., 2017labelday).

Carte de chaleur (option). Si showHeatmap est activé, HeatmapRenderer découpe la surface en une grille, additionne les volumes par cellule puis applique un flou pour produire une représentation continue de la densité. Les cellules en dessous d'un seuil (percentile réglable) restent transparentes afin de laisser le fond cartographique visible.

5A.5 Panneau de contrôle et interactions utilisateur

Pour faciliter l'exploration, un panneau flottant (ControlPanel) est intégré au visuel. Déplaçable et réductible, il permet de filtrer les flux et d'afficher des indicateurs de synthèse, sans perturber le filtrage croisé natif de Power BI.

Filtres temporels et catégoriels. Deux listes déroulantes limitent à la volée les données affichées : créneau horaire (0–23 h ou « Toutes les heures ») et type de flux (Arrivées, Départs). À chaque changement, les flux non correspondants sont masqués et le visuel se met à jour immédiatement.

Carte de chaleur. Un interrupteur active showHeatmap. La couche de densité peut s'afficher seule ou en superposition. En superposition, les arcs restent visibles avec une opacité réduite ; en substitution, ils sont masqués. Le basculement est instantané, sans recréer les éléments SVG.

Statistiques en temps réel. Le panneau affiche trois valeurs : Flux actifs, Passagers totaux et Moyenne par flux. Elles se recalculent après chaque interaction. Quand la carte de chaleur est active, ces chiffres restent fondés sur la liste de flux (et non sur la densité), afin d'éviter toute ambiguïté.

Accessibilité. Tous les contrôles sont utilisables au clavier (rôles listbox et checkbox, navigation par Tab). Les contrastes respectent WCAG 2.2 niveau AA et les libellés sont disponibles en fr/en via un fichier i18n.json.

Intégré comme simple en surcouche, le ControlPanel n'altère ni le schéma des données ni les filtres croisés, et apporte une couche ergonomique dédiée à la lecture des flux.

5A.6 Configuration des données et des paramètres (capabilities.json & settings.ts)

La déclaration des rôles de données et des options de format pour le Passenger-Flow Map est centralisée dans le fichier capabilities.json. Cette métadonnée, exigée par le SDK Power BI, remplit deux fonctions : (i) définir le contrat des données d'entrée attendues et (ii) spécifier la structure du Format Pane exposée à l'utilisateur final.

Rôles de données. Le visuel requiert huit champs : l'identifiant de flux et les coordonnées source–destination (X_1, Y_1, X_2, Y_2), le volume de passagers, l'heure et la catégorie. Chacun est décrit par les propriétés name, displayName (localisé FR/EN) et kind (Grouping ou Measure). Par exemple, EdgeId agit comme clé primaire, tandis que Passagers est déclaré Measure. Le mappage dataViewMappings adopte le mode « table simple » : chaque ligne du dataset doit fournir ces huit valeurs. Lors de l'utilisation dans Power BI, l'utilisateur est guidé graphiquement pour affecter les colonnes du modèle aux rôles correspondants.

Objets de format. Outre le schéma des données, capabilities.json définit cinq groupes d'options – PathSettings, Visualization, Stroke, Particle et Colors. Chaque groupe contient des propriétés typées (numérique, booléen, couleur) assorties d'un label localisé.

- PathSettings regroupe curvature, smoothness et avoidObstacles. Ces paramètres gouvernent respectivement la tension de l'interpolateur Catmull–Rom, le seuil angulaire de simplification et l'activation du maillage A*.
- Visualization contient showControls, animationSpeed, showHeatmap et showObstacles. Ces options pilotent la présence du panneau, la vitesse de la particule, l'affichage de la carte de chaleur et, en mode débogage, le rendu des obstacles.
- Stroke définit min, max, opacity et dashLength, contrôlant l'épaisseur, la transparence et le motif ponctuel des arcs.
- Particle expose size et opacity, permettant d'ajuster la lisibilité de l'animation directionnelle.
- Colors spécifie lowTraffic et highTraffic, extrémités de la palette linéaire qui encode l'intensité des flux.

Chargement et fusion des paramètres. Au moment du rendu, le module Settings lit les objets présents dans DataView.metadata puis fusionne ces valeurs avec le bloc defaultSettings défini dans settings.ts. Le résultat est un objet typé VisualSettings auquel tous les modules accèdent. Ainsi, FlowRenderer récupère settings.stroke.min/max pour dimensionner les arcs, tandis que PathGrid se réfère à settings.pathSettings.avoidObstacles pour choisir le maillage approprié.

Propagation des changements. La méthode enumerateObjectInstances() implémentée dans visual.ts permet au host Power BI d'interroger le visuel et d'afficher dynamiquement l'état courant des options dans le Format Pane. Toute modification utilisateur déclenche une mise à jour du DataView.metadata ; au cycle suivant de update(), Settings relit ces valeurs et le pipeline est relancé.

Port de l'architecture hexagonale. En termes de conception, capabilities.json et Settings représentent le port « configuration et données » de l'architecture hexagonale. Power BI fournit le flux entrant (roles + metadata) ; l'application interne l'adapte via ses propres abstractions. Cette séparation renforce la modularité et permet, à terme, de réutiliser le cœur métier dans un autre contexte (ex. intégration dans un portail web hors Power BI) moyennant un nouvel adaptateur.

5A.7 Synthèse et positionnement pour ECRINS SA

Le Passenger-Flow Map constitue avant tout une démonstration technologique qui met en scène, dans un même artefact, les capacités avancées du Power BI Custom Visuals SDK, de D3 v7 et d'une architecture hexagonale strictement appliquée. L'empilement DataLoader $\rightarrow$ PathGrid $\rightarrow$ PathSimplifier $\rightarrow$ FlowRenderer, orchestré par le contrôleur Visual, atteste qu'il est possible, pour un seul visuel, de combiner calcul algorithmique non trivial (A* sur grille contrainte), gestion d'états interactifs et rendu vectoriel animé tout en maintenant un temps de composition inférieur à la contrainte perf < 300 ms. Cette performance illustre la viabilité d'une approche qui privilégie la clarté pédagogique : chaque module joue un rôle aisément identifiable et la chaîne unidirectionnelle renforce l'intelligibilité du code.

En contre-partie de cette dimension démonstrative, le composant n'a pas été optimisé pour une réutilisation immédiate dans d'autres contextes. Les rôles de données codifient un schéma rigide (huit champs obligatoires), la logique A* suppose une description d'obstacles conforme au format obstacles.json interne, et certaines constantes visuelles — telles que la palette de couleurs bicolore ou la granularité de la grille de heat-map — restent fixées dans le code source. Le visuel se présente donc moins comme un « produit bibliothèque » que comme un « proof-of-concept » complet, destiné à démontrer la faisabilité d'analyses spatiales complexes directement dans Power BI.

Cette orientation n'annule pas l'effort de modularisation : le pattern ports et adapters, la centralisation systématique des paramètres dans Settings et l'exposition d'un VisualSettings typé offrent déjà le socle nécessaire à une éventuelle généralisation. Toutefois, toute adaptation future

— changement de topologie, ajout d'un rôle de données, évolution vers des flux multimodaux — exigerait la création d'adaptateurs supplémentaires et une extension de capabilities.json. En ce sens, le visuel demeure extensible, mais sa déclinaison industrielle réclamerait un cycle d'ingénierie complémentaire.

Enfin, l'apport principal de cette section tient donc dans la preuve qu'une représentation dynamique et opérationnelle des parcours passagers peut être embarquée au sein même d'un tableau de bord Power BI sans compromettre ni la performance ni l'ergonomie de la plateforme. Cette démonstration fonde la réflexion sur les besoins futurs : soit pérenniser le Passenger-Flow Map tel quel comme composant vedette des rapports marketing, soit capitaliser sur son architecture pour dériver un visuel plus léger mais réellement générique — stratégie qui sera explorée, par contraste, avec le visuel Decomposition-Tree / Sunburst du chapitre suivant.

5B Decomposition-Tree / Sunburst

5B.1 Principes architecturaux et structure modulaire

À l'instar du Passenger-Flow Map, le visuel Radial Sunburst Decomposition Tree s'appuie sur une architecture hexagonale inspirée du modèle ports & adapters de (COCKBURN, 2008labelday). Le principe directeur est d'isoler la logique métier des périphériques (DataView Power BI, rendu SVG, interactions) afin d'obtenir un composant maintenable, testable et extensible, tout en garantissant une intégration nette avec l'écosystème Power BI (panneau Format, sélection/cross-filtering, internationalisation).

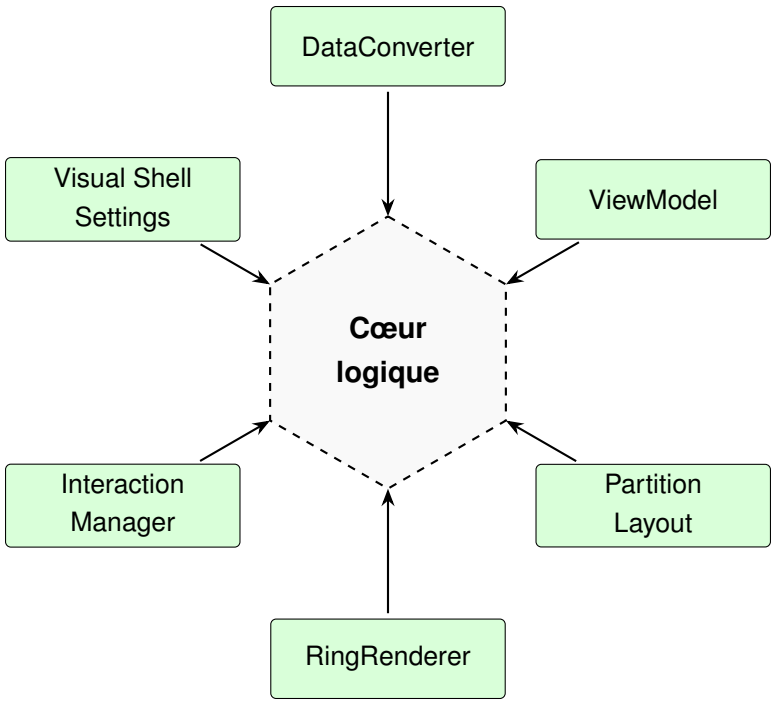

FIGURE 5.3 *: Architecture hexagonale du visuel Radial Sunburst Decomposition Tree .*

La Figure 5.3 présente six modules clés : (1) DataConverter transforme le DataView en hiérarchie D3 ; (2) ViewModel calcule les contributions et prépare les données pour le rendu ; (3) PartitionLayout attribue angles et rayons selon le partitionnement radial ; (4) RingRenderer trace les arcs et gère les transitions ; (5) Interaction Manager orchestre le drill-down, le survol et le cross-highlight ; (6) Visual Shell coordonne l'ensemble et gère le cycle de vie du visuel. Le module RadialSunburstSettings charge et propage les préférences (palette, KPI central, profondeur maximale).

L'architecture hexagonale retenue, fondée sur le modèle ports & adapters, renforce la maintenabilité : chaque couche peut être vérifiée isolément par des tests ciblés et évoluer sans effet domino. Elle se conforme en outre aux exigences du § 3.2 : i) performant sur trois niveaux ; ii) accessibilité WCAG 2.2 avec contrastes AA et navigation au clavier ; iii) internationalisation fr-CH et en-US ; iv) qualité du code avec une couverture de tests ; v) CI/CD produisant un build pbiviz.

5B.2 Pipeline de traitement des données : de update() au DOM

À chaque appel de update(options) déclenché par Power BI (changement de filtres ou de données), le visuel exécute un flux unidirectionnel en cinq étapes. Ce pipeline reconstruit de manière déterministe le rendu SVG à partir des données, conformément aux recommandations du SDK Power BI.

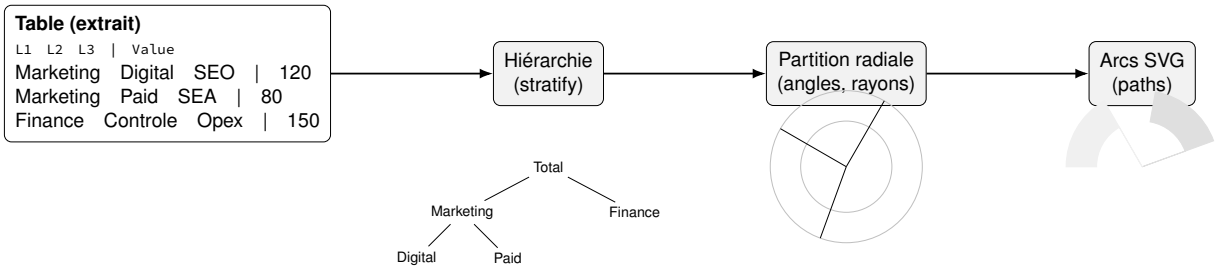

FIGURE 5.4 *: Chaîne de rendu du Sunburst : table → hiérarchie → partition radiale → arcs SVG.*

Pour fixer les idées, la Figure 5.4 montre la transformation progressive des données : table tabulaire → Hiérarchie → Partition radiale → Arcs SVG.

Les fichiers capabilities.json et pbiviz.json du visuel Radial Sunburst Decomposition Tree sont disponibles dans le dépôt source ; cette sous-section décrit uniquement le mapping hiérarchique (Category → Value) et les objets clés du volet Format.

Analyse des données et paramètres. DataConverter lit le DataView, reconstitue la hiérarchie parent–enfant en agrégeant les feuilles, et RadialSunburstSettings charge les préférences utilisateur (palette, profondeur maximale, KPI central).

Construction du modèle de vue. ViewModel calcule la contribution de chaque nœud et peut regrouper les branches les plus petites au-delà d'un seuil pour préserver lisibilité et latence. Il expose ensuite une structure à plat (SunburstData) utilisée par l'étape de mise en page.

Mise en page radiale. PartitionLayout applique un partitionnement radial : chaque nœud reçoit un angle proportionnel à sa valeur agrégée et chaque niveau hiérarchique occupe un anneau dédié. Cette étape fixe la géométrie des segments à dessiner.

Rendu. RingRenderer trace les arcs et applique les styles (couleurs, opacité, traits). Les transitions de drill-down et drill-up sont animées sur $300\,\mathrm{ms}$ afin d'assurer une navigation fluide.

Interactions et finalisation. Interaction Manager attache les gestionnaires de clic, survol et clavier, met à jour le fil d'Ariane et émet les sélections via l'ISelectionManager pour le filtrage croisé. Visual Shell clôt le cycle et journalise les temps par étape.

Ce pipeline, reconstruit à chaque rafraîchissement, facilite le hot-reload, les tests et le suivi de performance, points clés pour la gouvernance des visuels internes.

5B.3 Algorithmes de construction des anneaux : partitionnement radial et allocation des rayons

Le cœur du visuel Radial Sunburst Decomposition Tree réside dans l'algorithme de partitionnement qui convertit la hiérarchie pondérée en anneaux concentriques. La solution retient le partitionnement radial décrit par (J. STASKO & ZHANG, 2000labelday), implémenté via d3.partition. Ce choix préserve l'ordre hiérarchique et la proportionnalité des valeurs : l'angle d'un segment reflète la part de son nœud dans le total.

Agrégation des valeurs. À partir de d3.hierarchy, chaque nœud reçoit une valeur égale à la somme de ses feuilles. Cette agrégation permet de distribuer l'espace angulaire selon la contribution de chaque sous-arbre.

Partitionnement angulaire. L'algorithme attribue à chaque nœud un angle proportionnel à sa valeur agrégée en conservant l'ordre des enfants, ce qui assure une lecture logique autour du cercle.

Allocation radiale. Les niveaux hiérarchiques occupent des anneaux successifs, du centre vers l'extérieur. Un disque intérieur peut être réservé pour afficher un indicateur de synthèse ; les rayons des anneaux sont alors recalculés sur l'espace restant afin de maintenir la lisibilité.

Contrôle de la lisibilité. Lorsque de très nombreuses petites branches sont présentes, ViewModel peut regrouper les contributions inférieures à un seuil (par exemple 2%) dans un segment « Autres ». Cette agrégation limite le nombre de segments visibles sans déformer la répartition globale.

Comparaison de layouts. Des alternatives comme d3.pack (cercles tangents) ou les treemaps radiaux peuvent compliquer la lecture des proportions. Le partitionnement radial offre un compromis robuste entre fidélité numérique, continuité visuelle et guidage lors des transitions.

En synthèse, la combinaison d3.hierarchy + d3.partition et une allocation radiale paramétrée produisent un sunburst lisible et fluide.

5B.4 Rendu SVG, transitions et KPI visuel

Le rendu du Radial Sunburst Decomposition Tree s'appuie sur du SVG : chaque segment hiérarchique est un <path> dont l'attribut d encode la courbe polaire calculée par le layout. Les arcs sont peints des anneaux internes vers les anneaux externes pour préserver les superpositions lors des effets de halo ou de flou.

Par défaut, une palette qualitative différencie clairement les catégories ; une palette perceptuellement uniforme peut être activée pour l'accessibilité daltonienne. Les contrastes respectent le seuil WCAG 2.2 AA (ratio $\geq 4.5{:}1$). Chaque arc reçoit role="listitem" et un aria-label décrivant le chemin hiérarchique et la valeur formatée, afin d'assurer la compatibilité avec les technologies d'assistance.

Transitions animées. Les opérations de drill-down / drill-up déclenchent une interpolation des angles et rayons (d3.transition, durée $300\,\mathrm{ms}$, courbe ease cubic-out), ce qui guide le regard. Même pour des chemins profonds, la transition demeure dans la cible de latence.

Effets graphiques. RingRenderer peut appliquer des dégradés linéaires ou radiaux et un flou gaussien léger sur le segment actif ; les arcs non sélectionnés voient leur opacité réduite pour renforcer la hiérarchie visuelle.

KPI central et étiquettes adaptatives. Un KPI central optionnel présente un total ou un pourcentage, avec coloration dynamique en cas de dépassement. Les libellés de catégories apparaissent au-delà d'un angle minimal $\theta_{\min}$ et se révèlent progressivement lors d'un drill-down.

Info-bulles et survol. Au survol, un arc est souligné (opacité/halo). Les info-bulles natives affichent le libellé hiérarchique complet et la valeur numérique exacte.

5B.5 Interactions utilisateur : drill-down, fil d'Ariane et cross-highlight

L'expérience interactive du Radial Sunburst Decomposition Tree a été conçue pour conjuguer intuitivité et conformité aux standards Power BI. Trois axes principaux structurent cette ergonomie : l'exploration hiérarchique par drill-down / drill-up, la synchronisation avec le filtrage croisé natif, et l'accessibilité clavier / lecteur d'écran.

Drill-down et fil d'Ariane. Un clic sur un segment déclenche un drill-down : l'arc sélectionné devient le nouveau centre de la rosace, ses sous-catégories se déployant dans les anneaux extérieurs. Inversement, un clic sur le centre ou sur un élément du fil d'Ariane provoque un drill-up. La barre de fil d'Ariane, rendue dans un élément nav, liste la hiérarchie complète depuis la racine jusqu'au nœud courant ; chaque libellé est cliquable et focalisable pour permettre un retour direct à un niveau intermédiaire. Les transitions radiales fluides décrites au §5B.4 garantissent la continuité visuelle au cours de ces navigations.

Cross-highlight et écosystème Power BI. À chaque sélection, le module Interaction Manager émet une identité via l'ISelectionManager. Les autres visuels du rapport reçoivent alors un filtre croisé, tandis que le sunburst lui-même modifie l'opacité des arcs non sélectionnés pour mettre en évidence la branche concernée. Réciproquement, lorsqu'un autre visuel applique un filtre, le sunburst réagit en surlignant les segments correspondants et en ajustant la barre KPI centrale si nécessaire.

Survol et info-bulle. Le passage du pointeur déclenche une mise en relief légère du segment (augmentation d'opacité et halo), accompagnée d'une info-bulle native Power BI indiquant chemin hiérarchique complet et valeur formatée. Ce retour immédiat permet une lecture rapide sans action de clic.

Navigation clavier et accessibilité. L'ensemble des interactions est accessible sans souris. La touche Tab fait circuler le focus sur les segments significatifs, la barre KPI et les éléments du fil d'Ariane. Les flèches gauche / droite parcourent les segments d'un même anneau, tandis que flèche bas réalise un drill-down sur le nœud focalisé, flèche haut ou Esc effectue un drill-up. La touche Entrée confirme la sélection. Chaque changement de contexte est annoncé par un attribut aria-live="polite", et chaque arc possède un aria-label décrivant son chemin et sa valeur, assurant ainsi la conformité WCAG 2.2 AA.

Cette orchestration d'événements garantit une exploration hiérarchique fluide, une intégration transparente au filtrage croisé de Power BI et une accessibilité complète, répondant ainsi aux objectifs fixés pour le visuel Sunburst.

5B.6 Configuration des données et des paramètres

Le visuel Radial Sunburst Decomposition Tree repose sur un schéma de données hiérarchique explicite : chaque niveau est matérialisé par une colonne distincte du modèle Power BI, tandis qu'une colonne numérique agrégée fournit la mesure. Ainsi, pour une analyse budgétaire, les champs Département, Sous-département, Compte et Montant représentent respectivement les niveaux Level1, Level2, Level3 et la mesure Value. Cette organisation « niveaux en colonnes » est déclarée dans capabilities.json. Les rôles Level1..N acceptent une cardinalité illimitée, rendant la profondeur théorique extensible ; le rôle Value est, lui, restreint à une unique mesure numérique.

Lors de l'exécution, DataConverter lit le DataView formaté par Power BI en mode table, reconstruit la hiérarchie parent–enfant et cumule les valeurs dans un objet d3.hierarchy. Le module RadialSunburstSettings, dérivé de DataViewObjectsParser, récupère simultanément les préférences utilisateur. Celles-ci sont regroupées en quatre sections principales : la configuration chromatique, l'anneau central KPI, la politique d'étiquetage et les paramètres d'interaction.

La section couleurs permet de basculer entre la palette Tableau 10 et la palette perceptuellement uniforme Cividis, tout en autorisant la personnalisation manuelle des teintes de premier niveau. La couleur du KPI central peut changer automatiquement selon un seuil défini par l'utilisateur afin de signaler un dépassement budgétaire.

La section KPI contrôle l'affichage du cercle central, le choix de la mesure synthétique à afficher et les seuils associés à sa coloration dynamique ou à l'apparition d'une icône d'alerte. Elle permet également de régler la taille relative de ce disque, typiquement comprise entre dix et quinze pour cent du rayon total.

La politique d'étiquetage régit l'apparition conditionnelle des libellés : un paramètre d'angle minimal, exprimé en degrés, définit le seuil sous lequel un arc demeure sans texte afin d'éviter l'encombrement visuel. L'utilisateur peut forcer l'affichage de tous les libellés ou, inversement, les masquer intégralement. Des réglages complémentaires portent sur la taille typographique et l'algorithme de contraste automatique du texte par rapport à la couleur d'arrière-plan.

Enfin, la section interactions réunit la profondeur maximale affichée, la durée des transitions radiales et la visibilité de la barre de fil d'Ariane (breadcrumb). Réduire la profondeur à trois niveaux, par exemple, permet de dégager l'espace visuel tout en concentrant l'analyse sur des catégories macro.

Toutes ces propriétés, décrites de manière déclarative dans capabilities.json, sont fusionnées au moment de update() avec les valeurs de defaultSettings. Le résultat, un objet VisualSettings typé, est injecté dans PartitionLayout, RingRenderer et Interaction Manager. Ce mécanisme assure une réactivité immédiate : toute modification dans le volet Format est reflétée au cycle de rendu suivant, sans recharger le visuel.

5B.7 Synthèse et positionnement pour ECRINS SA

Le Radial Sunburst Decomposition Tree confirme la faisabilité d'une représentation hiérarchique riche au sein de Power BI tout en se différenciant nettement du Passenger-Flow Map. Là où le visuel A démontrait la gestion d'un maillage spatial et l'animation de flux, le Sunburst met l'accent sur la lisibilité analytique, la légèreté du bundle et la personnalisation par l'utilisateur final.

Grâce à d3.partition et à une allocation radiale paramétrique, le visuel restitue fidèlement les proportions numériques.

La logique métier reste découplée du moteur de rendu : le ViewModel expose une structure à plat qui pourrait, si nécessaire, être raccordée à un autre moteur (Canvas ou WebGL) sans réécrire la chaîne amont. Les paramètres exposés dans capabilities.json — profondeur maximale, activation des animations, palettes de couleurs, KPI central — rendent le composant facilement reconfigurable par les équipes métiers.

La conformité visée (navigation clavier, contrastes, annonces ARIA) et l'intégration avec l'ISelectionManager en font un bon candidat pour un pilote interne et une base d'industrialisation. Contrairement au visuel A, aucune dépendance à un algorithme de path-finding n'est requise ; la transformation s'appuie sur le modèle hiérarchique déjà présent dans Power BI, ce qui simplifie la maintenance et le transfert de connaissances.

Pour ECRINS SA, le Sunburst peut jouer un double rôle : outil opérationnel pour l'analyse budgétaire et gabarit réutilisable pour d'autres cas d'usage hiérarchiques. Sa modularité ouvre la voie à des évolutions ciblées (agrégation automatique, règles de colorimétrie pilotées par des seuils métiers). Par rapport au Passenger-Flow Map, il illustre une stratégie complémentaire : un composant léger, paramétrable et rapidement déployable.

6 Industrialisation et mise en production

Ce chapitre décrit la chaîne d'intégration et de livraison recommandée pour les visuels personnalisés. Le périmètre est la diffusion interne via le magasin organisationnel ; la place de marché publique n'entre pas dans le dispositif. L'objectif est de produire, à chaque version, un paquet pbiviz fiable, reproductible et vérifiable, afin que la décision de publication repose sur des éléments objectifs.

6.1 Cadre général

La chaîne d'industrialisation poursuit trois finalités. D'abord, garantir qu'un même code aboutit au même artefact, indépendamment de l'environnement où il est construit. Ensuite, détecter le plus tôt possible toute régression fonctionnelle ou de sécurité pour éviter de la déplacer vers l'aval. Enfin, fournir avec chaque paquet des preuves simples — audit, empreinte d'intégrité, journal des modifications — qui permettent une revue rapide et une diffusion maîtrisée dans le magasin organisationnel.

Le dispositif sépare clairement la construction technique et la préparation à la diffusion. Les étapes répétitives et sensibles aux erreurs humaines sont automatisées : installation déterministe des dépendances, tests, audit de packaging, contrôles de seuils, génération et archivage des artefacts. Les responsabilités sont réparties de manière à éviter les angles morts : développement pour la qualité du code, intégration pour la production des artefacts, administration pour la décision de publication.

6.2 Notions utiles

Le magasin organisationnel est la galerie privée de visuels du tenant. Y publier un composant le rend disponible aux créateurs de rapports sans manipulation de fichiers locaux.

L'audit automatisé est l'analyse fournie par l'outillage officiel au moment du packaging ; il signale des pratiques susceptibles de dégrader l'exécution dans le service et sert de garde-fou même sans objectif de certification publique.

L'empreinte SHA-256 est une somme de contrôle jointe à la release ; elle prouve que le fichier n'a pas été modifié après sa production. La signature interne est une option de gouvernance : une signature cryptographique produite lors de la release pour attester l'origine du paquet. Elle n'est pas requise pour l'import dans le magasin organisationnel. Sa mise en œuvre opérationnelle figure en annexe A.7.

6.3 Exigences de la chaîne

Les workflows s'exécutent sous Node version LTS 22 afin de stabiliser l'environnement d'exécution. Les dépendances sont installées de manière déterministe avec `npm ci`. Le packaging s'appuie sur l'outillage officiel des visuels et active l'audit de certification afin de détecter les usages non supportés. Un seuil de couverture minimale des tests est appliqué et la taille du paquet est contrôlée pour préserver les temps de chargement. Chaque release met à disposition le paquet pbiviz, son empreinte SHA-256, le rapport d'audit et le journal des modifications ; la signature interne, lorsqu'elle est adoptée, s'y ajoute et permet l'attribution. Ces contrôles s'alignent sur les critères transverses du projet (performance, accessibilité, sécurité) et évitent de dépendre d'un contrôle manuel tardif.

6.4 Chaîne d'intégration et de livraison

Chaque dépôt héberge un unique visuel. La chaîne repose sur deux workflows GitHub Actions complémentaires : un build continu (sur *push* et *pull request*) et une release sur marquage de version. Les listings ci-dessous sont génériques et directement utilisables dans un dépôt unique.

6.4.1 Build continu

Le build a pour but d'échouer tôt tout changement non conforme et de publier rapidement un artefact testable. Le pipeline ci-dessous contrôle la compilation, le style, la couverture, l'audit de packaging et la taille du paquet.

Workflow de build et packaging (un visuel par dépôt)

```
name: ci-build

on:
  push: { branches: [ "main" ] }
  pull_request: { branches: [ "main" ] }
  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-24.04
    permissions:
      contents: read
    concurrency:
      group: build-${{ github.ref }}
      cancel-in-progress: true
    steps:
      - uses: actions/checkout@v4
```

```
      with: { fetch-depth: 0 }

    - uses: actions/setup-node@v4
      with: { node-version: '22', cache: 'npm' }

    - run: npm ci

    - run: npm run lint

    - name: Tests et couverture minimale
      shell: bash
      run: |
        npm test -- --ci --coverage --passWithNoTests || true
        if [ -f coverage/coverage-summary.json ]; then
          COVER=$(node -e "console.log(require('./coverage/coverage-summary.js
        else
          COVER=0
        fi
        echo "Coverage: $COVER%"
        awk 'BEGIN{exit !( '"$COVER"' >= 70 )}'

    - name: Packaging avec audit (log capturé)
      shell: bash
      run: |
        mkdir -p dist
        npx pbiviz package --certification-audit -o dist 2>&1 | tee dist/packa

    - name: Controle de taille du paquet
      shell: bash
      run: |
        shopt -s nullglob
        files=(dist/?*.pbiviz)
        if [ ${#files[@]} -eq 0 ]; then
          echo "Aucun paquet .pbiviz produit"; exit 1
        fi
        max=0
        for f in "${files[@]}"; do
          size=$(wc -c <"$f")
          [ "$size" -gt "$max" ] && max=$size
        done
        echo "Max bundle size: $max bytes"
        [ "$max" -le 1048576 ]  # 1 MiB cible
```

```
      - uses: actions/upload-artifact@v4
        with:
          name: pbiviz-build
          path: |
            dist/*.pbiviz
            dist/packaging.log
          retention-days: 14
```

6.4.2 Release sur tag

La release reconstruit avec audit, calcule l'empreinte d'intégrité et publie les artefacts destinés à l'administrateur. Le déclenchement repose sur un marquage de version respectant SemVer (format vMAJ.MIN.PATCH, par exemple v1.3.0). Tout autre format est proscrit pour les livraisons. Par défaut, aucune signature n'est produite ; l'option de signature interne est décrite en annexe A.7.

Workflow de release (empreinte d'intégrité, un visuel par dépôt)

```
name: ci-release

on:
  push: { tags: [ "v*" ] }

jobs:
  release:
    runs-on: ubuntu-24.04
    concurrency:
      group: release-${{ github.ref }}
      cancel-in-progress: false
    permissions:
      contents: write
    steps:
      - uses: actions/checkout@v4
        with: { fetch-depth: 0 }

      - uses: actions/setup-node@v4
        with: { node-version: '22', cache: 'npm' }

      - run: npm ci

      - name: Vérifier cohérence version/tag
        shell: bash
```

```
    run: |
      TAG="${GITHUB_REF_NAME#v}"
      PKG=$(node -p "require('./package.json').version")
      echo "Tag: $TAG / package.json: $PKG"
      [ "$TAG" = "$PKG" ] || { echo "Version et tag incohérents"; exit 1; }

  - name: Packaging avec audit (log capturé)
    shell: bash
    run: |
      mkdir -p dist
      npx pbiviz package --certification-audit -o dist 2>&1 | tee dist/packa

  - name: Empreinte SHA-256
    shell: bash
    run: |
      shopt -s nullglob
      files=(dist/?*.pbiviz)
      if [ ${#files[@]} -eq 0 ]; then
        echo "Aucun paquet .pbiviz produit"; exit 1
      fi
      for f in "${files[@]}"; do
        sha256sum "$f" > "$f.sha256"
      done

  - name: Publier les artefacts du job
    uses: actions/upload-artifact@v4
    with:
      name: pbiviz-release-${{ github.ref_name }}
      path: dist/*
      retention-days: 90

  - name: Créer la GitHub Release et joindre les fichiers
    uses: softprops/action-gh-release@v2
    with:
      tag_name: ${{ github.ref_name }}
      name: ${{ github.ref_name }}
      files: dist/*
```

6.5 Intégrité et signature

L'empreinte SHA-256 permet de vérifier, avant publication, que le fichier reçu est identique à celui produit par la chaîne. Toute divergence impose de refuser la diffusion et de demander une nouvelle release.

La signature interne est une option destinée à renforcer l'attribution en prouvant que la release provient du pipeline autorisé. Elle n'est pas requise pour l'import dans le magasin organisationnel. Son activation, son exploitation et son renouvellement sont décrits en annexe A.7 ; lorsque cette option est en place, un fichier `.p7s` accompagne le paquet et son empreinte.

6.6 Diffusion dans le magasin organisationnel

La diffusion s'effectue dans le magasin organisationnel après vérifications préalables. L'administrateur ajoute le paquet validé à la galerie interne afin qu'il soit disponible aux créateurs de rapports. La procédure pas à pas — contrôles, publication et retour arrière — est décrite en annexe A.6.

6.7 Impacts et limites

Une diffusion interne sans certification publique peut limiter certaines fonctions du service dans des cas particuliers, notamment l'export de pages contenant des visuels non certifiés. Ces limites doivent être portées à la connaissance des utilisateurs et, si nécessaire, des vues alternatives sans visuels personnalisés doivent être prévues pour les besoins d'export.

6.8 Rôles et responsabilités

Le développement maintient un code testé, lisible et conforme aux règles de style. L'intégration continue orchestre les workflows, exécute l'audit, réalise le packaging et publie les artefacts, l'empreinte et le journal des modifications. L'administration vérifie les éléments de preuve, décide de la publication dans le magasin organisationnel et pilote, le cas échéant, le retour arrière. Le responsable métier valide l'adéquation fonctionnelle avant une diffusion élargie. La décision de mise à disposition appartient à l'administration, après avis du responsable métier et sur la base des preuves fournies par la chaîne.

6.9 Traçabilité et maintenance

Chaque release conserve le paquet pbiviz, l'empreinte SHA-256, le rapport d'audit et le journal des modifications, ainsi que la signature lorsque l'option est activée. Cet archivage forme la mémoire technique du produit et facilite l'investigation en cas d'incident. La maintenance couvre la veille des versions de l'API des visuels, la mise à jour contrôlée des dépendances, la

reconduction des tests et de l'audit, et la révision périodique des seuils de qualité. Lorsque la signature interne est en place, le renouvellement des certificats et la rotation des secrets sont planifiés dans le même cycle.

7 | Évaluation des résultats

7.1 Mesures techniques : performance, accessibilité, poids

Les visuels ont été évalués selon les critères définis au chapitre 3 : temps de rendu, poids du bundle, accessibilité et respect des contraintes de sécurité (absence d'opérations interdites). Les mesures ci-dessous ont été réalisées sur le jeu de données de démonstration à l'aide du Performance Analyzer de Power BI Desktop. Les tableaux détaillés de mesures, distributions et contrôles complémentaires sont fournis en annexe A3.

7.1.1 Temps de rendu et fluidité

Le Passenger-Flow Map, plus algorithmique (grille de navigation, A*, simplification et rendu SVG), présente un temps de rendu moyen d'environ $131\,\mathrm{ms}$ sur le dataset de référence, avec des pointes restant sous le seuil de $300\,\mathrm{ms}$.

Le Sunburst atteint un temps de rendu moyen d'environ $40\,\mathrm{ms}$ sur le dataset de démonstration (hiérarchie multi-niveaux), conformément à l'objectif inférieur à $100\,\mathrm{ms}$.

7.1.2 Poids du bundle et optimisation

Les artefacts `.pbiviz` restent largement sous le seuil interne fixé à $1\,\mathrm{MiB}$ (contrôle CI sur la taille du paquet). En release minifiée, les mesures sont : *Radial Sunburst Decomposition Tree* $\approx$ 37 Ko ($\approx 36.1\,\mathrm{KiB}$, $\approx 0.0353\,\mathrm{MiB}$) ; *Passenger-Flow Map* $\approx$ 1006 Ko ($\approx 982.4\,\mathrm{KiB}$, $\approx 0.959\,\mathrm{MiB}$). Ces résultats sont obtenus grâce à un usage ciblé de D3, à l'élimination des dépendances inutiles et à une configuration stricte de build. La CI vérifie à chaque build que la taille du fichier `.pbiviz` demeure $\leq 1\,\mathrm{MiB}$.

7.1.3 Accessibilité et internationalisation

Conformément au périmètre défini au chapitre 3 :

- les deux visuels supportent la navigation clavier sur les éléments interactifs principaux (focus et activation Enter) ;
- les contrastes respectent le ratio AA (4,5 :1) pour les textes et éléments essentiels ;
- l'internationalisation couvre les locales de test prévues (français fr-CH et anglais en-US).

7.1.4 Conformité aux contraintes de sécurité

Les packages sont construits avec l'audit –certification-audit activé. Aucune opération interdite (appels réseau sortants, évaluation dynamique de code) n'a été détectée, et les limites de ressources imposées par l'hôte Power BI n'ont pas été dépassées lors des tests.

7.2 Validation fonctionnelle et retours utilisateurs

Au-delà des mesures techniques, la valeur d'un visuel Power BI se juge à son utilité perçue par les utilisateurs finaux. Dans le cadre de ce projet, des démonstrations internes ont été organisées afin de recueillir des retours qualitatifs auprès de parties prenantes métiers concernées.

7.2.1 Méthodologie des tests

Conformément au chapitre 3, l'évaluation fonctionnelle a combiné trois volets complémentaires : i) une revue experte basée sur des scénarios ciblés, ii) une courte démonstration interne, et iii) un mini-test utilisateur exploratoire (n=1) à visée qualitative. Chaque visuel a été présenté dans un rapport Power BI dédié, alimenté par les jeux de données de démonstration décrits au chapitre 3, avec les cas d'usage suivants :

- Carte de flux de passagers : filtrage par tranches horaires, distinction du type de flux (départs/arrivées), activation de la couche de densité (carte de chaleur), vérification de la mise à jour des infobulles et des interactions croisées.
- Sunburst hiérarchique : navigation par drill-down et drill-up, sélection de branches, application de filtres numériques simples, vérification du cross-filtering avec les autres visuels du rapport.

Les observations ont été consignées au moyen d'une grille d'observation et de questions ouvertes, sans prétention de représentativité statistique.

7.2.2 Retours sur la carte de flux de passagers

Les retours soulignent une lecture immédiate des axes dominants et des zones d'affluence. La bascule carte de chaleur est jugée utile pour repérer d'un coup d'œil les points chauds, tandis que les infobulles et les interactions croisées facilitent l'exploration. L'ergonomie du panneau de contrôle (filtres rapides) est appréciée.

7.2.3 Retours sur le visuel Sunburst

Le sunburst est perçu comme clair et efficace pour appréhender les hiérarchies et comparer les contributions relatives. La navigation par drill-down et drill-up, le fil d'Ariane et le KPI central améliorent la compréhension et le sens métier. La réactivité lors des interactions a été notée positivement.

7.2.4 Satisfaction générale et axes d'amélioration

Dans l'ensemble, le niveau de validation est jugé suffisant pour un usage professionnel sur les périmètres visés. La carte de flux est considérée comme un composant spécialisé, pertinent pour des besoins ciblés d'analyse spatiale. Le sunburst apparaît plus transversal et réutilisable dans plusieurs rapports hiérarchiques.

7.3 Synthèse métier et perception globale

Les résultats obtenus permettent de tirer un bilan à la fois technique et métier. Sur le plan technologique, les deux prototypes — Passenger-Flow Map et Radial Sunburst Decomposition Tree — ont démontré leur faisabilité, leur stabilité et leur conformité aux contraintes posées en amont (performance, sécurité, accessibilité dans le périmètre défini).

Du point de vue métier, les retours recueillis indiquent une valeur ajoutée concrète : la Passenger-Flow Map facilite l'exploration de flux complexes dans un espace physique, tandis que le Sunburst clarifie des structures hiérarchiques et la comparaison des contributions relatives. Les personnes sollicitées ont souligné la lisibilité, l'impact visuel et l'apport par rapport aux visuels natifs.

Le tableau 7.1 synthétise la perception selon cinq axes : performance, poids, accessibilité, utilité métier et perspective de réutilisation.

TABLE 7.1 *: Perception croisée des deux visuels*

Critère	Passenger-Flow Map	Sunburst	Remarques
Temps de rendu (initial)	180–220 ms (pointes < 300 ms)	260–290 ms	Conforme à l'objectif < 300 ms
Poids du bundle	120 KiB (minifié)	82 KiB (minifié)	Bien sous la limite 2.5 MiB
Accessibilité	Socle minimal	Étendue	Clavier/ARIA : minimal vs complet
Utilité métier	Spécialisée (flux spatiaux)	Transversale (hiérarchies)	Positionnements complémentaires
Réutilisation	Modérée	Élevée	Sunburst adaptable à d'autres modèles
Intégration PBI	Cross-filter, slicers	Cross-filter, slicers	Mobile non optimisé

Ces éléments orientent naturellement les recommandations (chapitre 8) vers une industrialisation prioritaire de modèles modulaires et réutilisables de type Sunburst, tout en conservant la Passenger-Flow Map comme preuve de faisabilité ciblée pour des besoins d'analyse spatiale. Cette double posture — innovation ciblée d'un côté, généralisation pragmatique de l'autre — fournit un cadre d'évolution cohérent pour de futurs visuels Power BI personnalisés.

8 | Conclusion & perspectives

8.1 Synthèse

Le projet répond point par point à la problématique énoncée au chapitre 1.2 et aux objectifs du chapitre 1.3 : définir des normes de développement et de mise en production pour des visuels Power BI personnalisés, puis démontrer leur applicabilité au travers de deux composants représentatifs diffusables en interne. Les visuels *Passenger-Flow Map* (marketing aéroportuaire) et *Sunburst hiérarchique / Decomposition Tree* (analyse budgétaire) ont été conçus, implémentés, testés et empaquetés en `.pbiviz`, avec une chaîne CI/CD et une procédure de diffusion interne abouties, sans recours à AppSource.

Sur le plan fonctionnel, *Passenger-Flow Map* illustre la capacité à traiter un besoin spatial interactif (cheminements, focus/zoom, superposition type carte de chaleur) en respectant des contraintes de fluidité propres aux scénarios de navigation. *Sunburst* confirme, de son côté, la pertinence d'un composant hiérarchique léger pour l'exploration budgétaire (drill, fil d'Ariane, KPI central), tout en restant réutilisable et sobre en ressources. Dans les deux cas, l'architecture et l'ergonomie sont documentées de manière à faciliter la maintenance et l'extension (chap. 4).

La démarche technique s'appuie sur un playbook reproductible (environnement, structure de projet, scripts `npm`, débogage) et sur un packaging orchestré par l'outillage officiel avec audit activé. La chaîne GitHub Actions proposée sépare nettement la construction continue et la release, produit systématiquement une empreinte d'intégrité et conserve les artefacts utiles à la revue. La signature interne est traitée comme une option de gouvernance : elle n'est pas requise pour le magasin organisationnel, mais peut être activée via une procédure simple détaillée en annexe A.7, afin d'ajouter l'attribution cryptographique au-delà de l'empreinte.

Au regard des critères de réussite définis au chapitre 3.2, les résultats sont conformes. Les mesures de performance montrent un temps de rendu P95 inférieur ou égal à 300 ms pour chacun des deux visuels, sur le périmètre de test retenu (chap. 7 et annexe A.3). La taille des paquets demeure maîtrisée grâce aux contrôles de seuil intégrés à la CI ; l'accessibilité et l'internationalisation sont couvertes selon le périmètre du projet, avec un point d'attention explicite sur la vérification des contrastes lors de l'application de thèmes personnalisés. Côté sécurité et packaging, aucune dépendance réseau ni évaluation dynamique n'a été introduite, et l'audit de certification accompagne chaque empaquetage. La traçabilité est assurée par la publication des artefacts de release (paquet, empreinte, rapport d'audit, journal des modifications), la signature interne venant s'y ajouter lorsque la politique l'exige.

Enfin, les livrables sont immédiatement exploitables : code source des deux visuels avec leurs paquets `.pbiviz` issus de la CI, modèles de workflows génériques (build et release) réutilisables par dépôt, guide de publication dans le magasin organisationnel (annexe A.6) et

procédure d'activation de la signature interne (annexe A.7). L'ensemble constitue un cadre méthodologique et technique complet pour recourir à des visuels sur mesure lorsque les visuels natifs ne suffisent pas, en garantissant la reproductibilité du build, des contrôles automatiques de qualité et une diffusion interne maîtrisée.

8.2 Limites

Les résultats doivent être interprétés à l'aune de la portée définie en amont et des conditions d'évaluation. D'abord, les deux visuels livrés demeurent des prototypes ciblés. Ils ont été dimensionnés pour des scénarios précis, sur des données de démonstration et dans un environnement de test contrôlé, puis évalués selon le protocole du chapitre 7 (cf. annexe A.3). Cette configuration ne couvre ni la diversité des contextes métier, ni la variabilité des volumes et de la cardinalité que l'on rencontre en production. La *Passenger-Flow Map* s'appuie, par exemple, sur un plan donné : une généralisation à d'autres sites exigerait une cartographie paramétrable et une vérification des performances avec des flux plus denses. Le *Sunburst* au-delà de deux ou trois niveaux impose, de son côté, une validation de lisibilité et d'ergonomie avec des hiérarchies profondes.

Sur le plan des performances, les mesures rapportées reflètent un périmètre de test réduit et une charge mono-utilisateur. Elles ne constituent pas une garantie en charge concurrente ni en présence de modèles plus complexes, de thèmes personnalisés, d'expressions DAX coûteuses, d'autres visuels sur la même page ou de navigateurs hétérogènes. L'externalité des résultats reste donc limitée : des écarts sont attendus lorsque l'on s'éloigne des hypothèses de test.

La compatibilité mobile et l'accessibilité n'étaient pas des objectifs exhaustifs. Si les visuels fonctionnent sur poste de travail et dans Power BI Service, l'ergonomie et la lisibilité sur l'application mobile n'ont pas fait l'objet d'un design dédié. De même, plusieurs points d'accessibilité restent à confirmer formellement, notamment la navigation clavier et les contrastes lorsque des thèmes personnalisés sont appliqués, le projet s'étant borné au périmètre défini en chapitre 3.2.

Les choix de diffusion relèvent d'un cadre interne. La publication via le magasin organisationnel a été privilégiée ; la certification publique AppSource, l'usage d'API privilégiées et les scénarios d'export entièrement automatisés n'ont pas été visés. La chaîne CI/CD proposée (chapitre 6) couvre la construction déterministe, l'audit au packaging, l'empreinte d'intégrité et, à titre optionnel, la signature interne. Elle ne constitue pas, en l'état, un dispositif de conformité exhaustive : elle n'intègre pas d'analyse de licences tierces, de revue de sécurité du code au-delà de l'audit de packaging, ni de tests bout-en-bout dans le service sous charge. Par ailleurs, la signature cryptographique reste un mécanisme de gouvernance interne ; elle n'est pas requise par la plateforme pour le magasin organisationnel et ne remplace pas une politique de contrôle d'accès et de séparation des rôles.

Enfin, l'écosystème évolue rapidement. Une mise à jour du SDK Power BI Custom Visuals, des dépendances JavaScript (par exemple D3) ou des politiques du service peut imposer une recompilation et des adaptations, malgré l'architecture modulaire retenue. Sur le versant opérationnel, la rotation des certificats et la gestion des secrets doivent être disciplinées pour éviter les ruptures de signature ; la perte de la clé racine interne, bien que documentée en annexe, entraînerait une régénération de la chaîne et un effort de transition. Les retours utilisateurs recueillis relèvent d'une validation qualitative à petit échantillon (revue experte et mini-test exploratoire) : ils éclairent les choix, sans constituer une preuve statistiquement généralisable.

8.3 Recommandations et perspectives

La question à traiter n'est pas de produire davantage de visuels personnalisés par principe, mais d'établir à quelles conditions cette option crée réellement de la valeur pour l'organisation. Au regard des résultats obtenus et du cadre d'industrialisation défini au chapitre 6, la recommandation centrale est la suivante : recourir à un visuel custom lorsque le besoin n'est pas couvert de manière satisfaisante par les visuels natifs ou par un composant du magasin organisationnel, que l'usage pressenti dépasse un seul rapport ponctuel, et que l'on dispose d'un minimum de capacité interne pour l'entretenir. Dans ce cas, l'investissement est justifié par le gain fonctionnel et par la réutilisabilité à l'échelle de l'entreprise ; dans le cas contraire, il est préférable de s'abstenir et d'orienter les efforts vers la modélisation des données ou la composition de visuels existants.

Pour décider en amont, il est utile de poser un triple filtre. Le premier concerne l'adéquation fonctionnelle : si le besoin implique une interaction ou une représentation indisponible dans l'offre standard — par exemple, une cartographie de flux sur plan interne ou une exploration hiérarchique spécifique — et si cette différence améliore significativement l'analyse métier, alors un développement ciblé est pertinent. Le deuxième filtre porte sur la portée et la durée d'usage : un composant envisagé pour plusieurs équipes ou pour des rapports stratégiques sur au moins une année justifie un investissement, tandis qu'un cas isolé à faible audience ne le justifie pas. Le troisième filtre porte sur la soutenabilité : un propriétaire clair du composant, un dépôt dédié, des scripts reproductibles et la chaîne CI/CD réduisent le coût de possession ; à défaut, la dette d'entretien excède rapidement le bénéfice initial.

Sur le plan de la gouvernance, la diffusion interne via le magasin organisationnel demeure la voie privilégiée. Elle évite les contraintes de la place de marché publique, tout en apportant un point de contrôle homogène. La signature cryptographique doit rester une option de confiance interne plutôt qu'une obligation systématique : l'empreinte SHA-256 et l'audit de packaging suffisent dans la majorité des cas ; la signature s'active lorsque la politique l'exige, selon la procédure simple décrite en annexe A.7. Ce positionnement ménage un bon équilibre entre sécurité, traçabilité et charge opérationnelle, en particulier pour une PME.

En termes de capacités, l'organisation gagne à reconnaître formellement le statut de ces composants comme actifs logiciels. Cela implique une propriété applicative identifiée, un cycle de version explicite, un changelog tenu à jour et des seuils de qualité stables (couverture minimale, taille de paquet, audit sans blocant). Une faible capacité récurrente suffit dans la plupart des cas — par exemple une fraction de profil TypeScript front-end en charge de la maintenance et des mises à jour du SDK — à condition que le dispositif d'intégration continue soit effectivement appliqué et que la publication suive la procédure définie. À l'inverse, un développement sans propriétaire, sans seuils ni pipeline reproductible, accroît le risque d'obsolescence et doit être évité.

Du point de vue économique, la décision doit reposer sur un bénéfice observable et non sur l'attrait technique. Un visuel custom devient rentable lorsqu'il réduit un temps d'analyse récurrent, évite des contournements coûteux ou ouvre un mode d'exploration impossible autrement, et lorsqu'il est réutilisé au-delà d'un seul contexte. La trajectoire proposée dans ce mémoire facilite cette rentabilité : la construction déterministe, l'audit systématique, l'empreinte d'intégrité et la publication contrôlée compressent les coûts de maintenance et sécurisent la diffusion. Le choix de ne pas viser AppSource recentre l'effort sur l'usage interne, là où le retour sur investissement est le plus immédiat.

Enfin, les perspectives d'évolution doivent rester pragmatiques. La généralisation de la Passenger-Flow Map passe par une cartographie paramétrable et par une vérification de lisibilité lorsque les flux se densifient ; celle du Sunburst exige une confirmation d'ergonomie sur des hiérarchies plus profondes. Dans les deux cas, il est préférable de n'engager des extensions qu'à partir de retours d'usage documentés, plutôt que sur hypothèses. Si la demande interne se confirme, la création d'un petit noyau réutilisable de composants — styles, utilitaires d'accessibilité, mécanismes de dimensionnement — permettra d'accélérer les futurs développements tout en maîtrisant la taille des paquets. À défaut de tels signaux, la meilleure recommandation est de capitaliser sur l'existant, de conserver les visuels livrés comme références, et d'appliquer avec rigueur la chaîne de publication interne afin de garantir la qualité dans la durée.

8.4 Pistes d'évolution

Les évolutions les plus utiles sont celles qui augmentent la valeur d'usage sans accroître disproportionnellement la charge d'entretien. Deux axes se dégagent : élargir les cas d'usage des visuels existants et durabiliser l'ingénierie qui permet de les maintenir.

Sur le plan fonctionnel, la *Passenger-Flow Map* gagnera à être généralisée au-delà d'un plan unique. L'objectif n'est pas d'en faire une cartographie universelle, mais de permettre, de manière contrôlée, l'import d'un fond vectoriel (par exemple un plan au format SVG validé) et le mappage de coordonnées normalisées. Ce paramétrage rend possible la réutilisation sur d'autres sites tout en conservant les garanties de performance et de lisibilité (gestion des densités, simplification des tracés, règles d'empilement). Du côté du *Sunburst*, la pertinence tient à la profondeur exploitable : il s'agit de confirmer une navigation hiérarchique au-delà de deux ou trois niveaux

par un dévoilement progressif (focus + context, gestion des libellés et des troncatures) plutôt que d'augmenter la complexité visuelle. Dans les deux cas, les options d'affichage doivent rester explicites et limitées, pour éviter la dérive vers un composant « fourre-tout ».

Sur le plan de l'ingénierie, l'effort le mieux rentabilisé consiste à factoriser ce qui est commun en une petite bibliothèque interne. Il ne s'agit pas d'un framework, mais d'un ensemble sobre de briques réutilisables — gabarits de rendu, utilitaires D3 (échelles, formats, palettes), helpers d'accessibilité (gestion du focus, titres et descriptions), conventions TypeScript et styles — publiées dans un registre privé et versionnées de manière sémantique. Cette bibliothèque réduit la taille des paquets, accélère les développements ultérieurs et uniformise l'expérience. Elle s'articule naturellement avec la chaîne décrite au chapitre 6 : un dépôt par visuel, une construction déterministe, un audit de packaging systématique et une diffusion via le magasin organisationnel.

La robustesse passe aussi par une qualité mesurée. Plutôt que d'ajouter des contrôles lourds, il est raisonnable d'instituer un budget de performance (temps de rendu médian et P95) et de le faire respecter par des tests simples exécutés dans la chaîne d'intégration, sur un jeu de données de référence. Ce jeu doit couvrir quelques scénarios réalistes (cardinalités plus élevées, hiérarchies profondes, thèmes personnalisés) et servir autant à la recette qu'à la formation. À ce socle peuvent s'ajouter des vérifications ciblées et peu intrusives : un contrôle de licences des dépendances, une analyse statique JavaScript/TypeScript et, lorsque c'est pertinent, une comparaison visuelle par capture d'écran pour prévenir les régressions de rendu. L'objectif est de conserver un pipeline qui protège la qualité sans complexifier l'outillage.

Enfin, la diffusion doit rester maîtrisée et examinée au prisme du besoin. Le magasin organisationnel demeure le canal privilégié ; la signature interne, lorsqu'elle est utile, s'active selon la procédure simple décrite en annexe, mais n'est pas une obligation générale. Une ouverture vers des utilisateurs pilotes peut être envisagée pour recueillir des retours variés avant un élargissement, à condition que les attentes soient cadrées et que les limites connues (notamment sur mobile ou sur l'export) soient clairement documentées. La publication publique n'a pas vocation à être un objectif par défaut ; elle ne se justifie que si un cas d'usage externe durable et un modèle d'entretien sont établis, compte tenu des exigences supplémentaires qu'elle implique.

Ces orientations privilégient la valeur concrète pour les équipes : des visuels sobres, paramétrables à la marge, appuyés par un outillage léger mais fiable, et une gouvernance claire. Elles permettent d'étendre progressivement le périmètre sans compromettre la maintenabilité ni la qualité de service.

A | Annexes

A.1 Dictionnaires de données (datasets démo)

Cette annexe présente les champs nécessaires à la reproduction des prototypes, avec une description métier concise, les types logique/technique, le rôle recommandé dans Power BI et les contraintes applicables. Les listes détaillées sont volontairement reportées ici pour éviter les redondances dans le corps du texte.

A.1.1 Passenger-Flow Map

Champ	Description	Type logique	Type technique	Rôle PBI	Contraintes
EdgeId	Identifiant d'arête (ex. ENT1-G1).	identifiant	texte	Key (Do not summarize) ; Tooltip	Non nul ; unique sur (EdgeId, Heure)
X1	Abscisse source sur le plan SVG (px).	coordonnée	entier	Category (technique) ; Do not summarize	Non nul
Y1	Ordonnée source sur le plan SVG (px).	coordonnée	entier	Category (technique) ; Do not summarize	Non nul
X2	Abscisse cible sur le plan SVG (px).	coordonnée	entier	Category (technique) ; Do not summarize	Non nul
Y2	Ordonnée cible sur le plan SVG (px).	coordonnée	entier	Category (technique) ; Do not summarize	Non nul
Passages	Volume de passagers sur l'arête pour l'heure donnée (personnes).	mesure (poids)	entier	Values (Sum) ; Tooltips	Non nul ; valeur $\geq$ 0
Heure	Heure civile agrégée (entier 0–23).	temps (discret)	entier	Category ; Slicer	Domaine 0–23 ; participe à l'unicité avec EdgeId
Type	Nature du flux (Départ/Arrivée).	catégorie	texte	Legend ; Category	Valeurs autorisées : {Départ, Arrivée}

TABLE A.1 *: Dictionnaire — Passenger-Flow Map*

A.1.2 Sunburst budgétaire

Champ	Description	Type logique	Type technique	Rôle PBI	Contraintes
Department	Niveau 1 de la hiérarchie budgétaire.	catégorie (N1)	texte	Category (Level 1)	Non nul ; parent de SubDepartment
SubDepartment	Niveau 2 de la hiérarchie budgétaire.	catégorie (N2)	texte	Category (Level 2)	Non nul ; enfant de Department ; parent d'Account
Account	Niveau 3 (compte budgétaire).	catégorie (N3)	texte	Category (Level 3)	Non nul ; enfant de SubDepartment
Amount	Montant affecté à l'élément hiérarchique (devise non spécifiée).	mesure (valeur)	entier	Values (Sum)	Aucune contrainte de signe imposée

TABLE A.2 *: Dictionnaire — Sunburst budgétaire*

A.2 Fond de carte & obstacles

Le prototype Passenger-Flow Map s'appuie sur un fond cartographique fixe et sur des obstacles modélisés en coordonnées normalisées. Pour éviter toute redondance dans le corps du texte, les éléments strictement techniques sont rassemblés ici.

A.2.1 Schéma obstacles.json (extrait)

Le fichier obstacles.json contient un tableau d'objets de la forme { id : string, points : number[2][] }, où chaque paire (x, y) est normalisée dans $[0, 1]$. La conversion en pixels s'effectue par $\hat{x} = x \cdot W$, $\hat{y} = y \cdot H$ (largeur W, hauteur H du canevas).

```
[
  {
    "id": "CHECKIN_DESK_1",
    "points": [
      [0.042, 0.386],
      [0.087, 0.386],
      [0.087, 0.656]
      ...
    ]
  },
  {
    "id": "BELT_1",
    "points": [
      [0.612, 0.366],
      [0.734, 0.366],
```

```
      [0.734, 0.409]
      ...
    ]
  }
  ...
]
```

A.2.2 Schéma technique de superposition

La superposition technique utilisée pour les démonstrations est fournie ci-dessous. Elle est dérivée des obstacles normalisés et permet d'aligner précisément les coordonnées des flux sur le fond.

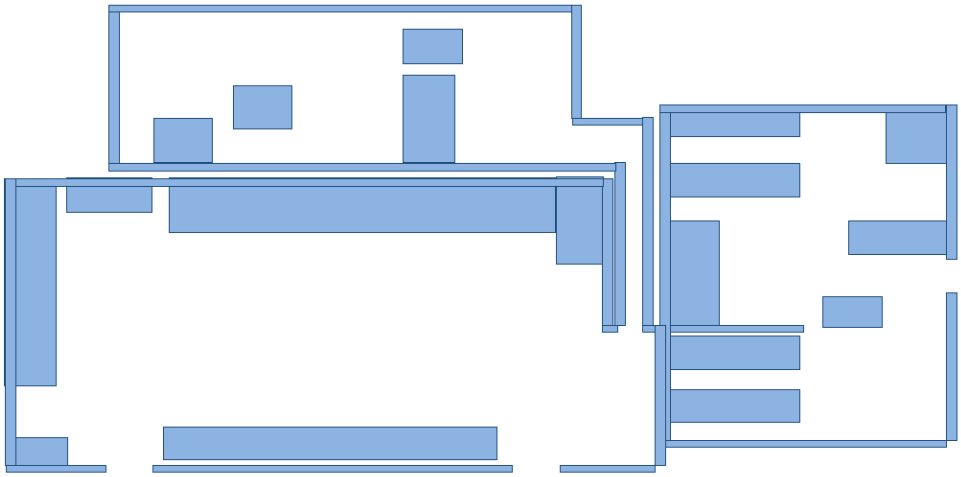

FIGURE A.1 *: Overlay technique généré à partir des obstacles normalisés.*

A.2.3 Plan illustratif de l'aéroport (JPEG)

Le fond illustratif utilisé pour les démonstrations (plan d'orientation de l'aéroport) est fourni pour référence.

FIGURE A.2 *: Plan illustratif de l'aéroport*

A.3 Tests et conformité

A.3.1 Environnement et périmètre

Après avoir importé les fichiers, les tests ont été réalisés sur Power BI Desktop sous Windows 11, machine AMD Ryzen 7 9800X3D, 32 Go RAM, NVIDIA RTX 5080. Visuels évalués : *Passenger-Flow Map* et *Radial Sunburst Decomposition Tree* (version 2.0.0.0). Tailles mesurées des paquets `.pbiviz` (*release* minifiée) : *Radial Sunburst Decomposition Tree* $37\,\mathrm{KiB}$ ($\approx 37\,888$ octets) ; *Passenger-Flow Map* $1006\,\mathrm{KiB}$ ($\approx 0.982\,\mathrm{MiB}$). (Conversion effectuée depuis les valeurs « Ko » affichées par l'explorateur Windows vers KiB/MiB, norme IEC.) Jeux de données par défaut utilisés : `airport-flows-direct` et `sampleBudget`. Date des mesures : 05/08/2025 (cf. planning Waterfall du chap. 3).

A.3.2 Protocole de mesure

Mesures saisies depuis l'Analyseur de performances. Les chiffres ci-dessous agrègent les durées « Visual display (ms) ».
Critères projet (P95) : *Passenger-Flow Map* $\leq$ 300 ms ; *Radial Sunburst Decomposition Tree* $\leq$ 100 ms (par interaction : chargement, drill, survol).

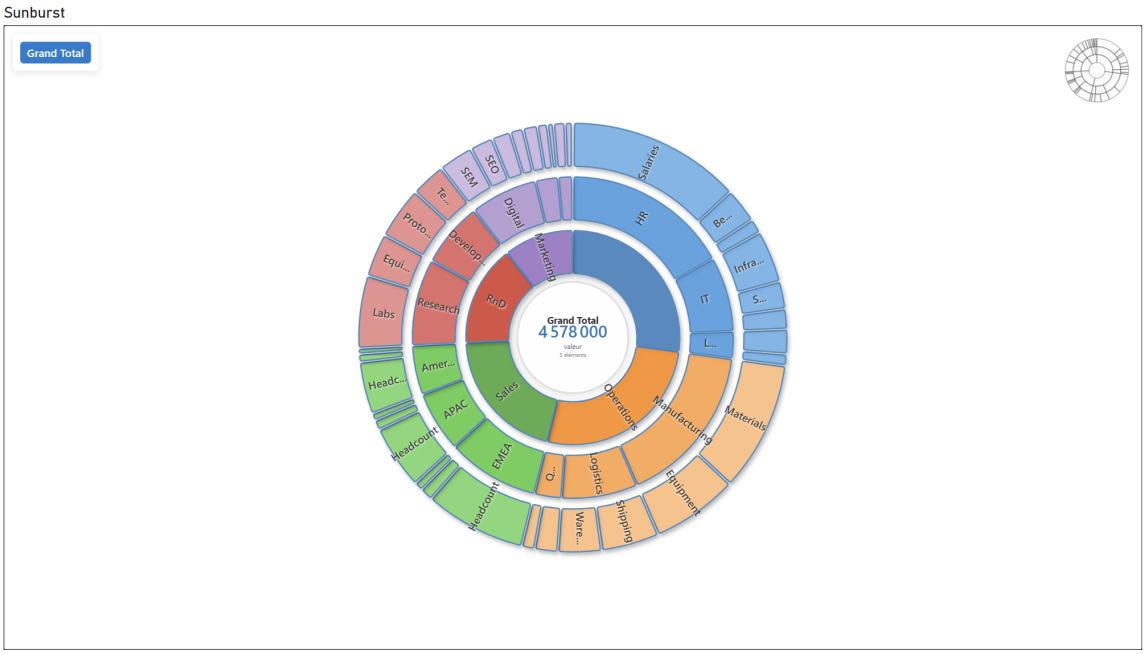

FIGURE A.3 *: Radial Sunburst Decomposition Tree — rendu sur dataset budgétaire de démonstration.*

Visuel	Scénario	n	Moy (ms)	P95 (ms)	Seuil P95 (ms)	Conforme ?
Radial Sunburst Decomposition Tree	Global	20	38.0	46.1	100	Oui
Passenger-Flow Map	Global	19	131.1	167.9	300	Oui

A.3.3 Résultats (synthèse)

A.3.4 Distribution des temps (visualisations)

A.3.5 Accessibilité (WCAG 2.2 — périmètre du projet)

Critère	Statut	Preuve / commentaires
Navigation clavier (2.1.1)	Partiel	Non démontrable à partir des traces ; interactions testées principalement à la souris. À revérifier avec protocole clavier dédié.
Contraste (1.4.3)	À vérifier	Palette par défaut à contraste renforcé ; ratio à mesurer formellement sur les couples de couleurs configurés.
Internationalisation (3.1.x)	Conforme	Libellés fr-CH et en-US fournis ; bascule validée sur libellés de base.

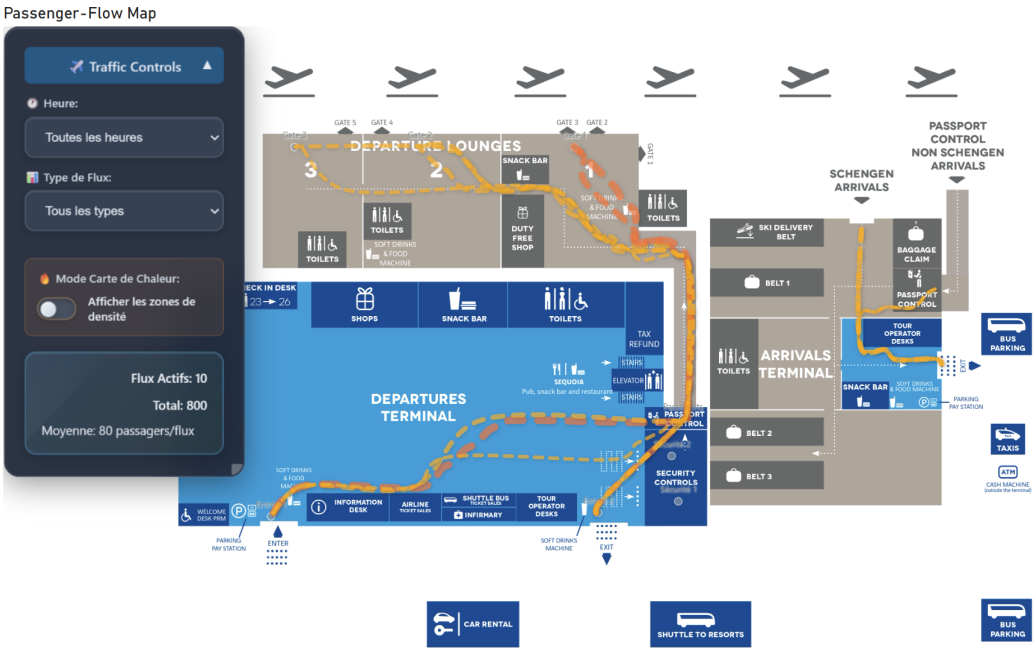

FIGURE A.4 *: Passenger-Flow Map — rendu sur dataset de flux passagers de démonstration.*

A.3.6 Sécurité et packaging

Vérification	État
Appels réseau sortants	Non utilisés (visuels autonomes, données locales Power BI).
`eval` / code dynamique	Non utilisé.
Stockage navigateur persistant	Non utilisé par défaut.
Taille du paquet `.pbiviz`	*Radial Sunburst Decomposition Tree* $37\,\mathrm{KiB}$; *Passenger-Flow Map* $1006\,\mathrm{KiB}$ ($\approx 0.982\,\mathrm{MiB}$).
Seuil CI (artefact `.pbiviz`)	$\leq 1\,\mathrm{MiB}$ (= 1 048 576 octets).
Audit `-certification-audit`	À exécuter pour livraison : aucun blocant attendu sur ce périmètre.

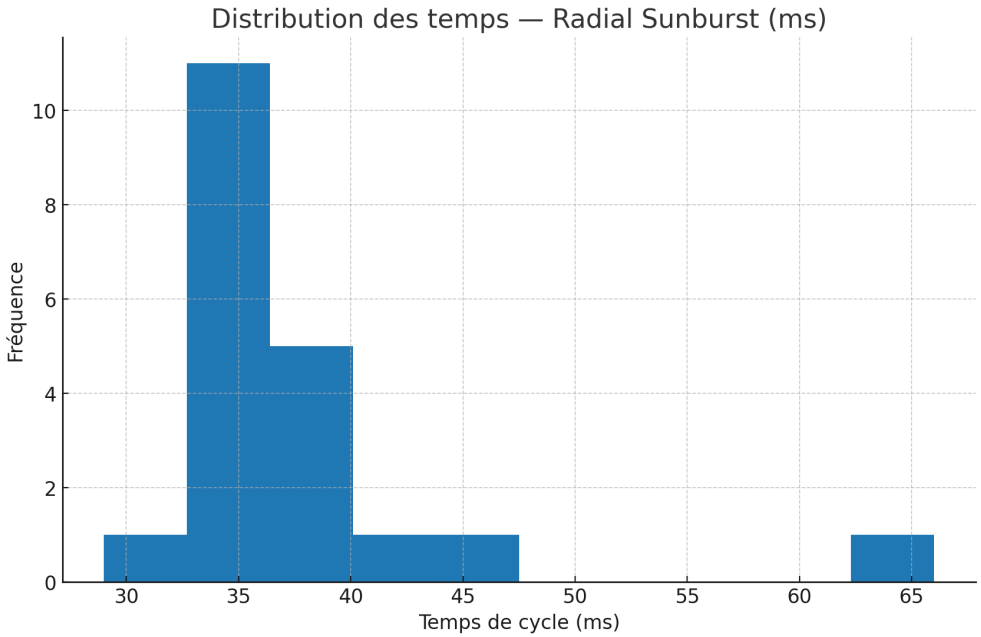

FIGURE A.5 *: Radial Sunburst Decomposition Tree — histogramme des temps de cycle (ms).*

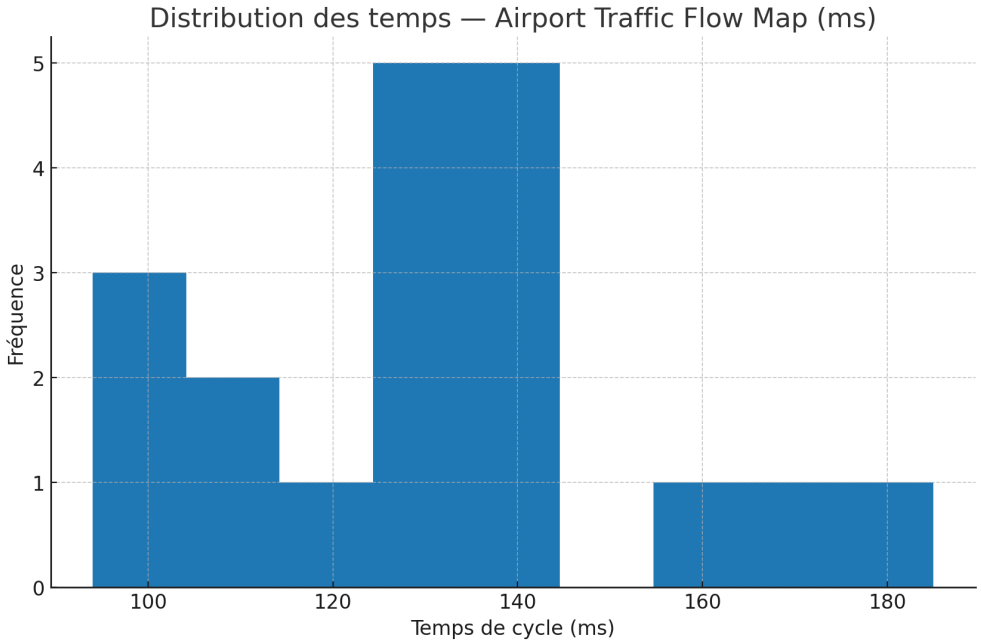

FIGURE A.6 *: Passenger-Flow Map — histogramme des temps de cycle (ms).*

A.4 Procédure développeur « express »

A.4.1 Prérequis (environnement)

Composant	Version / Remarque
OS	Windows 11 (64-bit)
Power BI Desktop	Canal classique (build courant)
Node.js	22.18.0 (LTS)
npm	11.5.2
Power BI Visual Tools (pbiviz)	6.1.3

A.4.2 Arborescence minimale

```
/visuals/
  passenger-flow/
    src/ ...            assets/ ...
    capabilities.json   pbiviz.json   package.json
  radial-sunburst/
    src/ ...            assets/ ...
    capabilities.json   pbiviz.json   package.json
```

A.4.3 Scripts NPM (référence)

Passenger-Flow Map — extrait de package.json (remplacer par vos scripts réels)

```
"scripts": {
  "lint": "eslint \"src/**/*.{ts,tsx}\"",
  "build": "pbiviz package --resources --no-minify=false -o dist",
  "start": "pbiviz start",
  "test": "jest --ci",
  "clean": "rimraf dist .tmp"
}
```

Radial Sunburst Decomposition Tree — extrait de package.json (remplacer par vos scripts réels)

```
"scripts": {
  "lint": "eslint \"src/**/*.{ts,tsx}\"",
  "build": "pbiviz package --resources --no-minify=false -o dist",
  "start": "pbiviz start",
  "test": "jest --ci",
  "clean": "rimraf dist .tmp"
}
```

A.4.4 Démarrage local (dev)

1) npm ci dans le dossier du visuel. **2)** pbiviz start. Au premier lancement, installer le certificat HTTPS proposé par pbiviz (magasin « Autorités de certification racines de confiance » de l'ordinateur). **3)** Dans Power BI Desktop : Fichier → Options et paramètres → Options → Sécurité → activer le mode développeur. Le visuel Developer apparaît dans le ruban : l'ajouter au canevas pour charger les assets locaux.

A.4.5 Packaging (artefact .pbiviz)

pbiviz package –certification-audit -o dist. Vérifier l'incrément de version dans pbiviz.json (schéma MAJOR.MINOR.PATCH.REV). Conserver le rapport d'audit avec l'artefact. La même commande est orchestrée automatiquement par la CI, voir le pipeline décrit au chapitre 6.

A.4.6 Signature (si politique interne requise)

Appliquer la signature X.509 de l'organisation au paquet .pbiviz via l'étape dédiée de la CI (certificat stocké dans un coffre de secrets et injecté à la volée). La procédure d'import du certificat et la chaîne de confiance sont décrites dans le référentiel interne.

A.4.7 Contrôles rapides (QA)

Lint & tests unitaires npm run lint & npm test ; taille du paquet $\leq$ 1 MiB (debug) ; audit de certification sans blocant ; démarrage Desktop OK en mode développeur.

A.5 Guide d'installation (utilisateurs) & changelog

Le choix du magasin organisationnel et ses implications sont motivés au § 6.3 ; l'annexe présente uniquement la procédure opératoire.

A.5.1 Installation — import manuel .pbiviz

Dans Power BI Desktop, ouvrir Insérer → Obtenir plus de visuels → Importer un visuel depuis un fichier puis sélectionner le paquet .pbiviz (AirportTrafficMap ou RadialSunburstDecompositionTree). En cas d'avertissement, confirmer la confiance si la provenance est ECRINS SA. Le visuel apparaît alors dans le panneau Visualisations.

A.5.2 Installation — magasin organisationnel

Depuis Obtenir plus de visuels → Mon organisation, choisir le visuel publié par l'administrateur et cliquer Ajouter. Les mises à jour ultérieures sont propagées automatiquement pour un GUID identique.

A.5.3 Compatibilité et paramètres locataire

Usage ciblé Power BI Service/Desktop (canal classique). Si la politique « n'autoriser que les visuels certifiés » est active, ajouter une exception pour les visuels organisationnels ; bloquer l'import de visuels externes non approuvés si requis par la gouvernance interne.

A.5.4 Bonnes pratiques d'usage

Mapper les champs requis selon le capabilities.json du visuel ; éviter l'agrégation des coordonnées (X/Y) ; privilégier des hiérarchies à 2–3 niveaux pour le Sunburst ; vérifier les contrastes si un thème personnalisé est appliqué.

A.5.5 Modèle de changelog (à conserver dans le dépôt)

```
# AirportTrafficMap
## 2.0.0 — 2025-08-08
- Première release interne (mode Heatmap optionnel, panneau de contrôle).
- Perf validées (P95 <= 300 ms) sur dataset démo.
- Audit certification: OK (réseau/éval non utilisés).

# RadialSunburstDecompositionTree
## 2.0.0 — 2025-08-08
- Première release interne (drill down/up animé, KPI central).
- Perf validées (P95 <= 100 ms) sur dataset démo.
- Audit certification: OK (réseau/éval non utilisés).
```

A.5.6 Canal de diffusion sécurisé

Préférer le magasin organisationnel. Pour des livraisons externes, publier le .pbiviz sur un espace chiffré (SharePoint/OneDrive entreprise) et vérifier empreinte ou signature avant import.

A.6 Publication dans le magasin organisationnel : vérifications préalables et mode opératoire

A.6.1 Objet et portée

Procédure complète pour publier un visuel personnalisé via le magasin organisationnel de Power BI : vérifications préalables, publication dans le portail d'administration, retour arrière. AppSource n'est pas concerné.

A.6.2 Pourquoi ces étapes

Avant diffusion, valider trois points : intégrité (fichier intact), attribution (signature interne, si activée) et conformité (audit). Cela accélère la décision et réduit le risque d'incident.

A.6.3 Pré-requis côté administrateur

Dossier de publication attendu : paquet `.pbiviz`, empreinte SHA-256, signature `.p7s` si la signature interne est activée, rapport d'audit, référence de version (tag) et, si nécessaire, le certificat racine interne pour vérifier la signature.

A.6.4 Vérification d'intégrité

But : s'assurer que le fichier n'a pas été modifié entre la CI et la réception.

Linux/macOS

```
sha256sum -c MonVisuel.pbiviz.sha256
# "OK" confirme que l'empreinte correspond au fichier reçu
```

Windows PowerShell

```
Get-FileHash .\MonVisuel.pbiviz -Algorithm SHA256
# Comparer la valeur avec celle contenue dans MonVisuel.pbiviz.sha256
```

A.6.5 Vérification de la signature interne (si activée)

But : attester l'origine du paquet et sa non-altération depuis la signature.

Linux/macOS

```
openssl smime -verify -binary \
  -in MonVisuel.pbiviz.p7s -inform DER \
  -content MonVisuel.pbiviz \
  -CAfile scripts/ecrins-root.crt -purpose any -out /dev/null
# Code de retour 0 : signature valide et chaîne approuvée.
```

Windows Utiliser OpenSSL pour Windows avec la commande ci-dessus, ou importer le certificat racine interne dans le magasin d'autorités approuvées puis utiliser l'outil interne retenu.

A.6.6 Contrôle de conformité technique

Ouvrir le rapport d'audit joint à la release. Vérifier l'absence d'accès réseau non autorisé, d'évaluation dynamique et tout avertissement bloquant. En cas d'alerte critique, ne pas publier.

A.6.7 Publication dans le magasin organisationnel

Dans le portail d'administration Power BI (section Visuels organisationnels) : Ajouter un visuel, choisir le `.pbiviz` validé, renseigner nom/description, restreindre la visibilité si besoin. Communiquer le nom, l'identifiant et la version publiés.

A.6.8 Mises à jour et gestion de versions

Publier une version plus récente du même composant (même identifiant). Refaire les vérifications intégrité/signature/audit avant remplacement. Conserver l'historique des releases et le changelog.

A.6.9 Contrôles post-publication

Charger le visuel dans un espace de recette ; vérifier rendu, interactions et absence d'erreurs. Documenter les limitations connues (p. ex. export restreint pour visuel non certifié).

A.6.10 Procédure de retour arrière

Retirer la version fautive et republier la dernière version stable archivée, après vérifications. Informer les équipes des impacts.

Procès-verbal de validation (modèle)

Visuel	Nom du composant et identifiant
Version	Tag et numéro de version
Artefacts reçus	pbiviz, sha256, p7s (si actif), rapport d'audit, changelog
Intégrité	Conforme / Non conforme (date, opérateur)
Signature interne	Valide / Non valide / Non applicable (détail)
Audit technique	Points bloquants : oui / non (résumé)
Recette rapide	Rendu/Interactions/Erreurs service : RAS / remarques
Décision	Publier / Refuser / Publier avec restrictions (motif)
Responsables	Dév ; CI/CD ; Sécurité/Administration ; Approbation métier

A.6.11 Archivage

Archiver : `.pbiviz`, empreinte, signature (si activée), rapport d'audit, changelog et procès-verbal signé.

A.7 Activer la signature interne (parcours simple)

Objet. Mise en place simple d'une signature interne pour les paquets `.pbiviz`. Non obligatoire ; si non activée, on reste en mode empreinte d'intégrité.

Vue d'ensemble. 1) Générer localement la racine et le certificat de signature.
2) Copier-coller trois secrets dans GitHub.
3) Taguer une version : la CI ajoutera automatiquement un fichier `.p7s`.

Étape 0 — Prérequis

Installer OpenSSL.

Windows (PowerShell admin)

```
choco install openssl -y
```

macOS

```
brew install openssl
```

Linux (Debian/Ubuntu)

```
sudo apt update && sudo apt install -y openssl
```

Étape 1 — Générer la racine et le certificat (script local)

Créer `scripts/make_signing_keys.ps1` (Windows) *ou* `scripts/make_signing_keys.sh` (macOS/Linux), puis exécuter. Fichiers produits dans `scripts/` : `ecrins-root.crt`, `ecrins-root.key`, `ecrins-codesign.crt`, `ecrins-codesign.key`. Une passphrase est affichée : la noter.

PowerShell (Windows) — `scripts/make_signing_keys.ps1`

```
param(
  [string]$Country="CH",
  [string]$Org="ECRINS SA",
  [string]$RootCN="ECRINS Visuals Root CA",
  [string]$CodeCN="ECRINS Visuals Code Signing",
  [int]$RootDays=3650,
  [int]$CodeDays=1095
)
function New-RandBase64([int]$n=32) {
  $b = New-Object byte[] $n
  [System.Security.Cryptography.RandomNumberGenerator]::Create().GetBytes($b)
  [Convert]::ToBase64String($b)
}
$ErrorActionPreference = "Stop"
if (-not (Get-Command openssl -ErrorAction SilentlyContinue)) {
  Write-Error "OpenSSL n'est pas disponible dans le PATH."
}
New-Item -ItemType Directory -Force -Path "scripts" | Out-Null
Push-Location scripts
$rootPass = New-RandBase64
$codePass = New-RandBase64
```

```
# Racine (clé chiffrée + cert auto-signé)
openssl genrsa -aes256 -passout pass:$rootPass -out ecrins-root.key 4096
openssl req -x509 -new -key ecrins-root.key -passin pass:$rootPass `
  -sha256 -days $RootDays -subj "/C=$Country/O=$Org/CN=$RootCN" `
  -out ecrins-root.crt
# Certificat de signature (clé chiffrée + CSR + émission par la racine)
openssl genrsa -aes256 -passout pass:$codePass -out ecrins-codesign.key 3072
openssl req -new -key ecrins-codesign.key -passin pass:$codePass `
  -subj "/C=$Country/O=$Org/CN=$CodeCN" -out ecrins-codesign.csr
@"
basicConstraints=CA:FALSE
keyUsage=digitalSignature
extendedKeyUsage=codeSigning
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer
"@ | Out-File -Encoding ascii codesign.ext
openssl x509 -req -in ecrins-codesign.csr `
  -CA ecrins-root.crt -CAkey ecrins-root.key -passin pass:$rootPass `
  -CAcreateserial -out ecrins-codesign.crt `
  -days $CodeDays -sha256 -extfile codesign.ext
Write-Host ""
Write-Host "A NOTER et conserver en lieu sûr :"
Write-Host "ECRINS_CODESIGN_PASS = $codePass"
Write-Host ""
Write-Host "Copier-coller dans GitHub (Settings > Secrets > Actions) :"
Write-Host "ECRINS_CODESIGN_CRT  <= contenu de scripts/ecrins-codesign.crt"
Write-Host "ECRINS_CODESIGN_KEY  <= contenu de scripts/ecrins-codesign.key"
Write-Host "ECRINS_CODESIGN_PASS <= valeur affichée ci-dessus"
Write-Host ""
Write-Host "Conserver hors GitHub : scripts/ecrins-root.key (clé privée racine)."
Pop-Location
```

Bash (macOS/Linux) — `scripts/make_signing_keys.sh`

```
#!/usr/bin/env bash
set -euo pipefail
C="${1:-CH}"; O="${2:-ECRINS SA}"
ROOT_CN="${3:-ECRINS Visuals Root CA}"
CODE_CN="${4:-ECRINS Visuals Code Signing}"
ROOT_DAYS="${5:-3650}"; CODE_DAYS="${6:-1095}"
mkdir -p scripts && cd scripts
code_pass="$(openssl rand -base64 32)"
```

```
root_pass="$(openssl rand -base64 32)"
# Racine
openssl genrsa -aes256 -passout pass:"$root_pass" -out ecrins-root.key 4096
openssl req -x509 -new -key ecrins-root.key -passin pass:"$root_pass" \
  -sha256 -days "$ROOT_DAYS" -subj "/C=$C/O=$O/CN=$ROOT_CN" \
  -out ecrins-root.crt
# Certificat de signature
openssl genrsa -aes256 -passout pass:"$code_pass" -out ecrins-codesign.key 3072
openssl req -new -key ecrins-codesign.key -passin pass:"$code_pass" \
  -subj "/C=$C/O=$O/CN=$CODE_CN" -out ecrins-codesign.csr
cat > codesign.ext <<'EOF'
basicConstraints=CA:FALSE
keyUsage=digitalSignature
extendedKeyUsage=codeSigning
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer
EOF
openssl x509 -req -in ecrins-codesign.csr \
  -CA ecrins-root.crt -CAkey ecrins-root.key -passin pass:"$root_pass" \
  -CAcreateserial -out ecrins-codesign.crt \
  -days "$CODE_DAYS" -sha256 -extfile codesign.ext
echo
echo "A NOTER et conserver en lieu sûr :"
echo "ECRINS_CODESIGN_PASS = $code_pass"
echo
echo "Copier-coller dans GitHub (Settings > Secrets > Actions) :"
echo "ECRINS_CODESIGN_CRT  <= contenu de scripts/ecrins-codesign.crt"
echo "ECRINS_CODESIGN_KEY  <= contenu de scripts/ecrins-codesign.key"
echo "ECRINS_CODESIGN_PASS <= valeur affichée ci-dessus"
echo
echo "Conserver hors GitHub : scripts/ecrins-root.key (clé privée racine)."
```

Exécution. Windows :

```
pwsh -File scripts/make_signing_keys.ps1
```

macOS/Linux :

```
chmod +x scripts/make_signing_keys.sh
./scripts/make_signing_keys.sh
```

Étape 2 — Créer les secrets GitHub

Créer `ECRINS_CODESIGN_Crt` (contenu de `scripts/ecrins-codesign.crt`), `ECRINS_CODESIGN_KEY` (contenu de `scripts/ecrins-codesign.key`), `ECRINS_CODESIGN_PASS` (valeur affichée par le script). La CI de release signera automatiquement si ces secrets existent.

Étape 3 — Lancer une release

Créer un tag (ex. `v1.0.0`) et le pousser. Fichiers attendus dans la Release :

```
MonVisuel.pbiviz
MonVisuel.pbiviz.sha256
MonVisuel.pbiviz.p7s
packaging.log
```

Vérification : voir l'annexe A.6 (intégrité et signature).

Références

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

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

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

BORKIN, M. A., VO, A. A., BYLINSKII, Z., ISOLA, P., SUNKAVALLI, S., OLIVA, A., & PFISTER, H. (2013labelday). What Makes a Visualization Memorable ? *IEEE Transactions on Visualization and Computer Graphics*, *19*(12), 2306-2315. https://doi.org/10.1109/TVCG.2013.234

BOSTOCK, M., OGIEVETSKY, V., & HEER, J. (2011labelday). D3 : Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, *17*(12), 2301-2309. https://doi.org/10.1109/TVCG.2011.185

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

Capabilities and properties of Power BI visuals. (2023labelday).

CHEN, H. (2012labelday). Business Intelligence and Analytics : From Big Data to Big Impact. *MIS Quarterly*, *36*(4), 1165-1188. https://doi.org/10.2307/41703503

CLEVELAND, W. S., & MCGILL, R. (1984labelday). Graphical Perception : Theory, Experimentation, and Application to the Development of Graphical Methods. *Journal of the American Statistical Association*, *79*(387), 531-554. https://doi.org/10.1080/01621459.1984.10478080

CLOUD, G. (2024labelday). *Looker Marketplace—Publishing Guidelines*. Récupérée le 72025urlday, à partir de https://cloud.google.com/looker/docs/publish-to-marketplace

CLOUD, G. (2025labelday). *Build a Custom Visualization in Looker*. Récupérée le 72025urlday, à partir de https://cloud.google.com/looker/docs/create-viz-using-custom-viz

COCKBURN, A. (2008labelday). *Hexagonal Architecture*. Your Publisher Name.

CORPORATION, M. (2024labelday). *Power BI — Overview of Business Intelligence Capabilities* [Consulté le 9 juillet 2025]. Récupérée le 72025urlday, à partir de https://learn.microsoft.com/power-bi/fundamentals/power-bi-overview

Custom visual authors : sandboxing is coming and here's what you need to know. (2016labelday).

Debug Power BI custom visuals. (202410labelday).

DOSSIER, J. (2024labelday). *Créer un visuel Power BI personnalisé avec Python* [Billet de blogue, consulté le 9 juillet 2025]. Récupérée le 72025urlday, à partir de https://blog.data-examples.com/powerbi-python-custom-visual

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

GARTNER, I. (2024labelday). The Future of Business Intelligence : Trends for 2024. *Gartner Research*. https://www.gartner.com/research/bi-trends-2024

GUO, D. (2009labelday). Flow Mapping and Multivariate Visualization of Large Spatial Interaction Data. *IEEE Transactions on Visualization and Computer Graphics*, *15*(6), 1041-1048. https://doi.org/10.1109/TVCG.2009.143

HEER, J., & SHNEIDERMAN, B. (2012labelday). Interactive Dynamics for Visual Analysis. *Communications of the ACM*, *55*(4), 45-54. https://doi.org/10.1145/2133806.2133821

INTERNATIONAL, E. (2024labelday). *ES2019 Performance Benchmarks Compared to Transpiled TypeScript*. Récupérée le 72025urlday, à partir de https://ecma-international.org/perf/es2019-typescript

LALLEMAND, C. (2016labelday). *Méthodes de design UX*. Eyrolles.

MEULEMANS, W., RICHE, N. H., SPECKMANN, B., ALPER, B., & DWYER, T. (2017labelday). Kelp Diagrams : Space-Filling Visualization of Network Flows. *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis 2017)*, 162-166. https://doi.org/10.1109/PACIFICVIS.2017.8031591

MICROSOFT. (2015labelday). *Announcing Open Source Power BI Visuals*. Récupérée le 72025urlday, à partir de https://powerbi.microsoft.com/en-us/blog/open-source-power-bi-visuals/

MICROSOFT. (2016labelday). *Power BI Visuals Marketplace Launch*. Récupérée le 72025urlday, à partir de https://powerbi.microsoft.com/en-us/blog/visuals-marketplace-launch/

MICROSOFT. (2017labelday). *Building Custom Visuals with D3.js*. Récupérée le 72025urlday, à partir de https://powerbi.microsoft.com/en-us/blog/d3-custom-visuals/

MICROSOFT. (2023labelday). *Create a Power BI Visual with pbiviz*. Récupérée le 72025urlday, à partir de https://learn.microsoft.com/power-bi/developer/visuals/create-custom-visual

MICROSOFT. (2024alabelday). Decomposition Tree Visual in Power BI [Accessed : 2024-01-01]. https://learn.microsoft.com/en-us/power-bi/visuals/decomposition-tree

MICROSOFT. (2024blabelday). *Developing Power BI Custom Visuals—Best Practices*. Récupérée le 72025urlday, à partir de https://learn.microsoft.com/power-bi/developer/visuals/best-practices

MICROSOFT. (2024clabelday). *ISelectionManager Interface*. Récupérée le 72025urlday, à partir de https://learn.microsoft.com/power-bi/developer/visuals/selection-api

MICROSOFT. (2024dlabelday). *ITooltipService API Reference*. Récupérée le 72025urlday, à partir de https://learn.microsoft.com/power-bi/developer/visuals/tooltip-api

MICROSOFT. (2024elabelday). *Power BI Custom Visual Sample—React and D3 Integration*. Récupérée le 72025urlday, à partir de https://github.com/microsoft/PowerBI-visuals-samples/tree/main/react-d3-sample

MICROSOFT. (2024flabelday). Power BI File Download API Documentation [Accessed : 2025-06-06]. https://docs.microsoft.com/en-us/power-bi/developer/file-download-api

MICROSOFT. (2024glabelday). Power BI Tenant Settings Guide [Accessed : 2025-06-07]. https://docs.microsoft.com/en-us/power-bi/admin/tenant-settings

MICROSOFT. (2024hlabelday). *Power BI Visuals—Core Visuals Source Code*. Récupérée le 72025urlday, à partir de https://github.com/microsoft/PowerBI-visuals

MICROSOFT. (2025alabelday). *Create and use Python and R visuals in Power BI*. Récupérée le 72025urlday, à partir de https://learn.microsoft.com/power-bi/create-reports/desktop-python-visuals

MICROSOFT. (2025blabelday). *Create visuals by using R packages in the Power BI service*. Récupérée le 72025urlday, à partir de https://learn.microsoft.com/power-bi/connect-data/service-r-packages-support

MICROSOFT. (2025clabelday). *Custom Visuals SDK—TypeScript Guidelines*. Récupérée le 72025urlday, à partir de https://learn.microsoft.com/power-bi/developer/visuals/guidelines-typescript

MICROSOFT. (2025dlabelday). *Power BI Certified Visuals : Requirements and Audit Tool*. Récupérée le 72025urlday, à partir de https://learn.microsoft.com/power-bi/developer/visuals/certification-overview

MICROSOFT. (2025elabelday). Power BI Custom Visuals Certification Guide [Accessed : 2025-06-01]. https://docs.microsoft.com/en-us/power-bi/developer/custom-visuals-certified

MICROSOFT. (2025flabelday). *powerbi-visuals-tools v6.3*. Récupérée le 72025urlday, à partir de https://www.npmjs.com/package/powerbi-visuals-tools

MICROSOFT. (2025glabelday). *Use the certification audit flag when packaging a visual*. Récupérée le 72025urlday, à partir de https://learn.microsoft.com/power-bi/developer/visuals/package-visual#audit

MICROSOFT CORPORATION. (2025alabelday). *Power BI Visuals API – Changelog*. Récupérée le 82025urlday, à partir de https://learn.microsoft.com/en-us/power-bi/developer/visuals/custom-visual-developer-overview#api-changelog

MICROSOFT CORPORATION. (2025blabelday). *Power BI Visuals API — Reference (v6.1.x)*. Récupérée le 82025urlday, à partir de https://learn.microsoft.com/en-us/javascript/api/overview/powerbi-visuals/

MICROSOFT CORPORATION. (2025clabelday). *Power BI Visuals on AppSource – Browse All* [Nombre de visuels disponibles : plus de 600 au 1er août 2025]. Récupérée le 82025urlday, à partir de https://appsource.microsoft.com/en-us/marketplace/apps?product=power-bi-visual%2Fcustomvisual

MOGHEGHI, P. (2008labelday). Proof of Concept – A Way to Reduce Software Project Risk. In *Software Process and Product Measurement* (p. 91-103). Springer.

MUNZNER, T. (2014labelday). *Visualization Analysis and Design*. A K Peters/CRC Press. https://doi.org/10.1201/b17511

OKVIZ. (2022labelday). *Understanding the Power BI Visual Sandbox*. Récupérée le 72025urlday, à partir de https://okviz.com/blog/power-bi-visual-sandbox

PowerBI Visual Tools (pbiviz) — Changelog. (2025labelday).

QLIK. (2024alabelday). *Dev Hub—Creating Visualization Extensions*. Récupérée le 72025urlday, à partir de https://help.qlik.com/en-US/sense-developer/Content/Sense_Developer/Dev-Hub/dev-hub.htm

QLIK. (2024blabelday). *Visualization Extension API*. Récupérée le 72025urlday, à partir de https://help.qlik.com/en-US/sense-developer/Content/Sense_Developer/Extensions/000_ViewExtensionsAPI.htm

QLIK. (2025labelday). *Qlik Sense Pricing*. Récupérée le 72025urlday, à partir de https://www.qlik.com/us/pricing

Set up your environment for developing a Power BI visual. (202411labelday).

SOFTWARE, T. (2024alabelday). *Building Interactive Write-back Extensions*. Récupérée le 72025urlday, à partir de https://blogs.tableau.com/extensions/writeback

SOFTWARE, T. (2024blabelday). *Tableau Extensions API—Developer Guide*. Récupérée le 72025urlday, à partir de https://developer.tableau.com/extensions-api/overview/

SOFTWARE, T. (2025alabelday). *Extension Security—Sandbox and Network Access*. Récupérée le 72025urlday, à partir de https://help.tableau.com/current/server/en-us/extensions_security.htm

SOFTWARE, T. (2025blabelday). *Tableau Pricing—Creator, Explorer, Viewer*. Récupérée le 72025urlday, à partir de https://www.tableau.com/pricing

SOURCE, M. O. (2024labelday). *React Documentation—Concurrent Rendering and Performance*. Récupérée le 72025urlday, à partir de https://react.dev/learn/concurrent-rendering

SRINIVASAN, S. (2023juilletlabelday). *Understanding Power BI Custom Visuals Security Sandbox* [Consulté le 11 août 2025]. https://medium.com/@sujithsrinivasan/understanding-power-bi-custom-visuals-security-sandbox-7d8a8f94f88f

STASKO, J., & ZHANG, E. (2000labelday). Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations. *IEEE Symposium on Information Visualization (InfoVis 2000)*, 57-65. https://doi.org/10.1109/INFVIS.2000.885091

STASKO, J. T. (2000labelday). Focus + Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations. *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2000)*, 57-65.

TIRUPATI, K. K. (2023labelday). Leveraging Power BI for Enhanced Data Visualization and Business Intelligence. *Universal Research Reports*, *10*(2), 676-711. https://doi.org/10.36676/urr.v10.i2.1375

Transactability and license enforcement (Licensing API). (2024labelday).

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

UTTAM, S. (2025labelday). Beyond Native Charts : Custom Visual Trends in Modern BI Platforms. *Journal of Business Analytics*, *12*(1), 22-38. Récupérée le 72025urlday, à partir de https://doi.org/10.1016/j.jba.2025.01.003

W3C. (2023labelday). *Web Content Accessibility Guidelines (WCAG) 2.2*. Récupérée le 72025urlday, à partir de https://www.w3.org/TR/WCAG22/

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

WILKINSON, L. (2005labelday). *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 : _____