

BUT SD -
2023

Projet Rshiny – Vélo'v

vélo'v



Flavio Texier
Maxence Chan
Robin Mondou-Laurent

Table des matières

Documentation Technique	2
Schéma relationnel	2
Explication du code	2
Documentation application R Vélo'v	4
Page « Accueil »	5
Page « Données Brutes »	6
Page « KPI »	6
Page « Bouton »	7

Documentation Technique

Schéma relationnel

Pour mener ce projet à bien, nous avons opté pour une structure relationnelle plutôt simple, composée de 2 tables. La première nommée « Stations » contient toutes les informations récupérées grâce à l'API Vélo'v. Elle contient toutes les informations sur les stations en temps réel, leur nom, leur adresse... Puis, nous nous sommes servis d'une deuxième table nommée « Géocodage », que nous avons créée nous-même. Comme son nom l'indique, elle contient le code postal de chaque station. Ces deux tables sont en format CSV.

Explication du code

Pour développer cette application, nous travaillons sur Rshiny. Deux possibilités s'offraient à nous, à savoir créer un script en deux parties, l'une contenant l'UI, c'est-à-dire le visuel de l'application et l'autre contenant le Server, c'est-à-dire le code qui permet à l'application de fonctionner ; ou bien de tout faire dans le même script. Nous avons choisi la deuxième option.

Tout d'abord on installe tous les packages dont on a besoin en utilisant une condition « if », afin de vérifier que le package ne soit pas déjà installé. Il s'agit ici d'optimiser de la puissance de calcul et d'épargner du temps. Ensuite, on charge tous les packages dans Rstudio.

On charge les données depuis l'API Velo'v avec la commande GET, et on stocke le tout dans un dataframe appelé « df ». On lit également le fichier CSV contenant le géocodage, afin de récupérer les codes postaux correspondant à chaque station. La création de ce fichier CSV à part sert à ne pas avoir à relancer un géocodage à chaque fois que l'application se lance. En effet, le géocodage prend plusieurs minutes à se faire, ce qui serait une perte de temps conséquente. On lie ensuite les deux fichiers avec la commande « cbind() », afin d'avoir toutes les informations dans le même dataframe.

Ensuite, nous définissons l'UI de l'appli, en utilisant la librairie dashboard(). Nous estimions qu'elle donnait un rendu très propre et qui allait bien avec les besoins du commanditaire.

Chaque tabItems() correspond à un onglet. On définit tout d'abord la page d'accueil. fluidRow() permet de disposer les éléments en ligne. On affiche ensuite la carte en utilisant le package leaflet(), plus précisément la fonction leafletOutput().

Sur la même page, on va afficher les filtres qui seront définis plus tard dans le server. La fonction selectInput() sert donc à renvoyer un élément que nous avons défini dans le server. On retrouve des éléments similaires au langage HTML, avec par exemple h2() ou br(). InfoBox() sert à créer une petite zone d'informations. C'est dans ces infobox que nous avons affiché les KPI.

Ensuite vient la partie Server. C'est ici qu'on va définir toutes les fonctions de l'application ainsi que son fonctionnement. On va par exemple créer des objets auxquels on a fait appel dans la

partie de l'UI. On crée immédiatement les fonctions input, output et session, qui vont nous servir à définir chacun des éléments. Output sert à stocker des éléments qui sont plutôt inactifs. On les stocke donc dans une sorte de variable, de liste. On prendra donc soin de stocker ces éléments en les nommant de la sorte `output$[nom de l'élément] <-`

Input sert plutôt à stocker des éléments interactifs, comme des boutons par exemple. On va donc stocker dans la fonction input toutes les instructions qui doivent être réalisées dès lors qu'il y a une interaction.

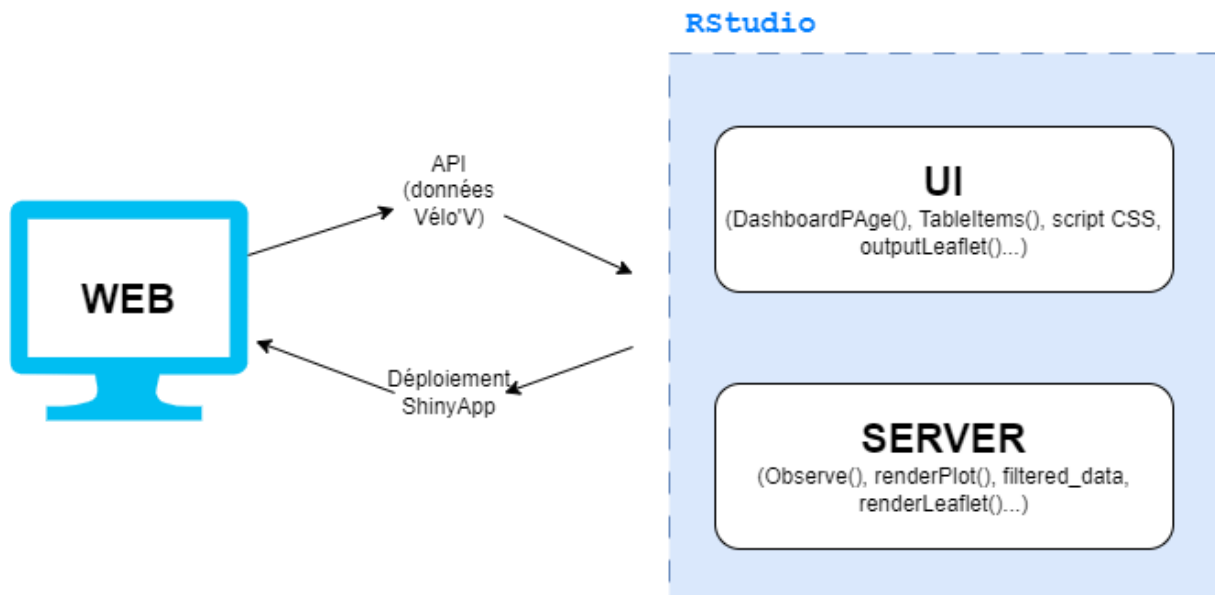
Filtered_data avec reactive sert à créer un filtre de données. C'est ce filtre qui est utilisé sur la page d'accueil et sur l'onglet des KPI.

La carte a été créée avec leaflet(), et on a pu ajouter des foyers de points (cluster). C'est à cette carte que l'on fait appel dans l'UI.

Les fonctions observe() et observeEvent() servent à mettre à jour les éléments. Elles vont surveiller ces éléments et les mettre à jour si elles constatent un changement. La fonction observeEvent() sert plus particulièrement à surveiller une situation en particulier.

Enfin, on peut lancer l'application en concluant notre script en appelant l'UI et le Server dans la fonction ShinyApp().

Diagramme décrivant le processus de création de l'application :



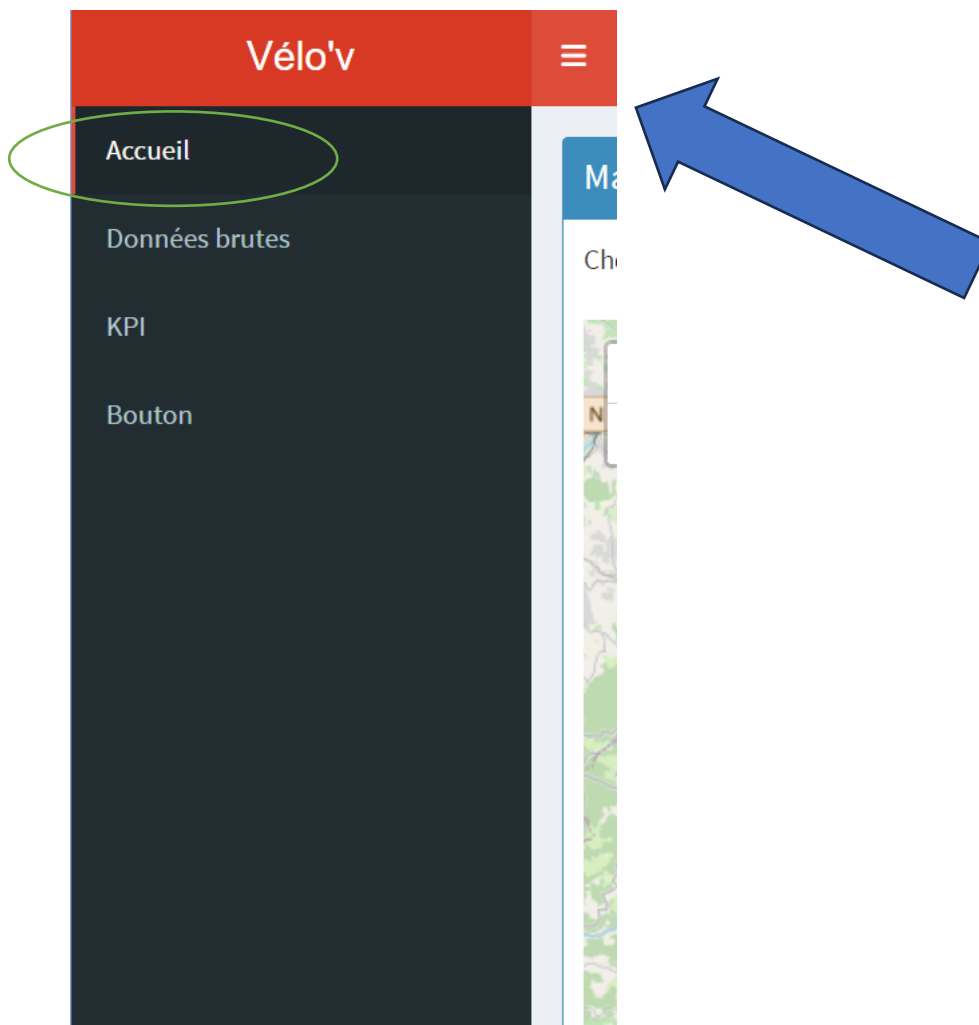
Documentation application R Vélo'v

Voici le lien de notre application

Rshiny: https://maxencechan.shinyapps.io/application_rshiny/

Dans cette partie, nous allons vous donner **une notice, un guide d'utilisation** de notre application Rshiny sur les données Vélo'v.

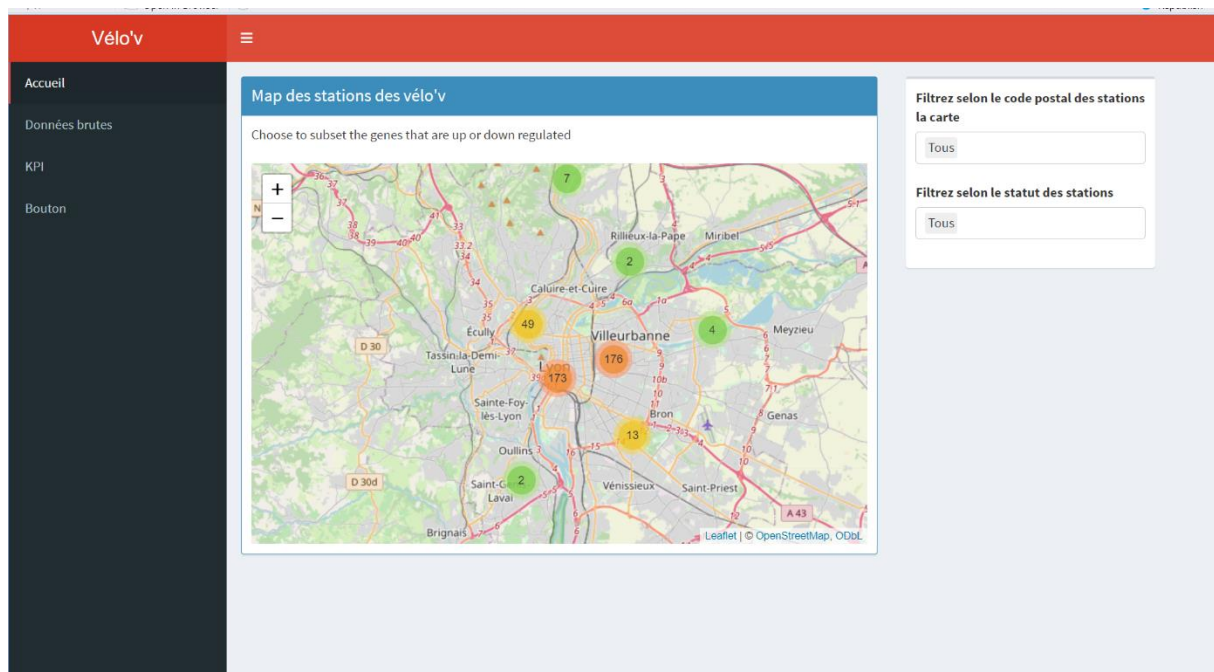
Une barre de navigation permet à l'utilisateur de naviguer entre 4 onglets : « Accueil », « Données brutes », « KPI », « Bouton » :



Page « Accueil »

L'application s'ouvre sur la page d'accueil avec 3 utilités :

- **Une carte interactive de Lyon avec toutes les stations existantes**, la carte peut être zoomer pour mieux visualisé les stations. De plus en cliquant sur un point de station, des informations supplémentaires sont accessibles (nom de la station, nombre de vélos disponibles, nombre de places disponibles et si la station est ouverte)
- Sur la droite de la carte, l'utilisateur peut retrouver un **filtre selon le code postal** pour rechercher plus facilement une station précise s'il le souhaite
- Un deuxième filtre est présent, celui-ci permet de **filtrer les stations en fonction de si elles sont ouvertes ou si elles sont fermées**.



Page « Données Brutes »

Le second onglet « Données brutes » situé en dessous de l'onglet « Accueil », dans la barre de navigation, permet **à l'utilisateur de voir les données brutes en temps-réelles qui sont utilisées dans l'application**. Toutes les données présentent donnent des informations plus détaillées de chaque station.

Vélo'v

☰

Accueil

Données brutes

KPI

Bouton

Show

10

▼

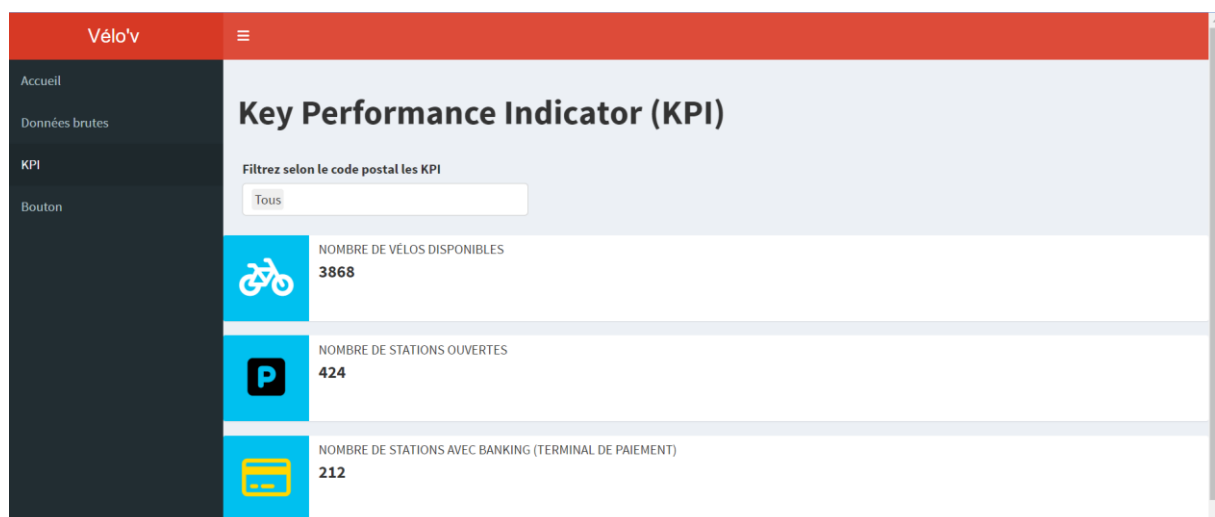
entries




Search:

number	contractName	name	address	banking	bonus	status	lastUpdate	connected	overflow	
1	2010	lyon	2010 - CONFLUENCE / DARSE	ANGLE ALLEE ANDRE MURE ET QUAI ANTOINE RIBOUD	true	false	OPEN	2023-10-29T22:17:51Z	true	false
2	5015	lyon	5015 - FULCHIRON	Devant le n°41 rue de la Quarantaine	true	false	OPEN	2023-10-29T22:19:58Z	true	false
3	32001	lyon	32001 - COUZON - CENTRE		true	false	OPEN	2023-10-29T22:14:31Z	true	false
4	6004	lyon	6004 - FOCH	Angle rue Vendôme	true	false	OPEN	2023-10-29T22:17:09Z	true	false
5	7035	lyon	7035 - MARSEILLE / UNIVERSITE	Angle rue de Marseille	true	false	OPEN	2023-10-29T22:23:21Z	true	false
6	10120	lyon	10120 - SALENGRO / DESCARTES	41 AV. ROGER SALENGRO (VILLEURBANNE)	true	false	OPEN	2023-10-29T22:19:10Z	true	false
7	5016	lyon	5016 - POINT DU JOUR / GRANGES	Devant la mairie du 5°	false	true	OPEN	2023-10-29T22:17:28Z	true	false
8	9013	lyon	9013 - QUAI PAUL SEDALLIAN	Angle rue de la Navigation	false	false	OPEN	2023-10-29T22:22:19Z	true	false
Cours de la										

Page « KPI »

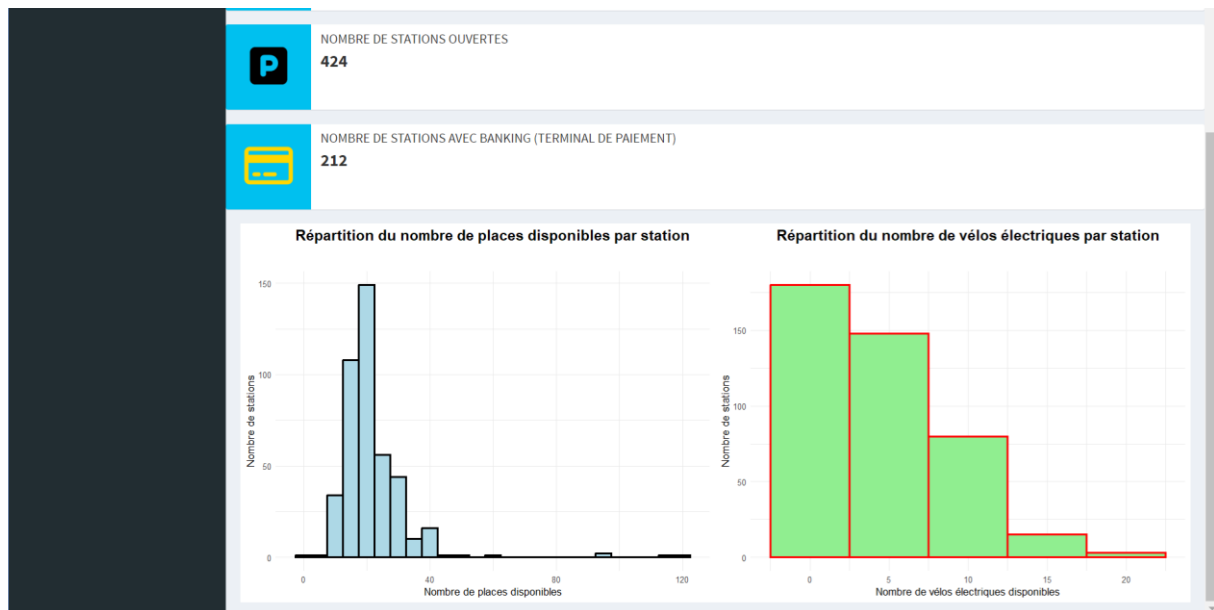
La page KPI donne des informations sur le nombre de vélos disponible, le nombre de stations ouvertes ainsi que le nombre de stations avec un terminal de paiement.



Key Performance Indicator (KPI)	
Filtrez selon le code postal les KPI	
Tous	
	NOMBRE DE VÉLOS DISPONIBLES 3868
	NOMBRE DE STATIONS OUVERTES 424
	NOMBRE DE STATIONS AVEC BANKING (TERMINAL DE PAIEMENT) 212

De plus, nous pouvons retrouver 2 graphiques :

- **La répartition du nombre de places disponibles par station**
- **La répartition du nombre de vélos électriques par station.**



Page « Bouton »

Cette dernière page a deux boutons que l'utilisateur peut exploiter :

- **Un bouton pour exporter les graphiques de la page KPI.**
- **Un bouton pour relancer l'application avec des données plus récentes en cas de changements au niveau des stations et des vélos.**

