Passer du prototype à l’industrialisation

Charles Bordet

16/06/2026

DATA CHAMP

« Ça marche. »

```
38999                'Are you sure you want to delete the project "',
39000                hold_project_name,
39001                '"?'
39002              )
39003            )
39004          ),
39005          br(),
39006          title = "Delete Project",
39007          footer = list(
39008            modalButton("Cancel"),
39009            actionButton(
39010              ns("delete_project"),
39011              "Delete",
39012              class = "btn-danger",
39013              style = "color: #FFF;"
39014            )
39015          ),
39016          size = "m"
39017        )
39018      )
39019    })
39020
39021    observeEvent(input$delete_project, {
39022      req(input$project_id_to_delete)
39023
39024      removeModal()
39025
39026      tryCatch({
39027        dbExecute(
39028          db_conn,
39029          "DELETE FROM aimm.projects WHERE uid=$1",
39030          params = list(input$project_id_to_delete)
39031        )
39032
39033        shinyFeedback::showToast('success', 'Project deleted')
39034        projects_trigger(projects_trigger() + 1)
39035      }, error = function(err) {
39036        print(err)
39037        shinyFeedback::showToast('error', 'Error deleting project')
39038      })
39039
39040    })
39041
39042        }
39043
```

39 043

lignes, un seul fichier

Script ≠ Produit

SCRIPT

- Code **jetable**
- Écrit et exécuté par **une** personne
- Code maintenable **non nécessaire**
- Réussi = **ça marche**

PRODUIT LOGICIEL

- Le code **vit des années**
- Maintenu par **plusieurs** personnes
- Code propre **indispensable**
- Réussi = **ça tient dans le temps**

DATA CHAMP

La checklist en 6 étapes

1. **Structure** du projet
2. **Reproductibilité**
3. **Qualité** du code
4. **Conception** produit
5. **Déploiement**
6. **Sécurité**

> Une **feuille de route** en 6 étapes, et pour chacune, de quoi **aller plus loin**.

1 · Structure du projet

RÉFLEXE SCRIPT

```
📂 mon-app/
   └── app.R  ← tout, 2000+ lignes (ou 39 043)
```

Personne ne sait où est quoi.

Tu touches un filtre → tu casses un graphe.

Personne d'autre que toi peut intervenir.

LA BONNE APPROCHE

```
📂 mon-app/
   ├── global.R · ui.R · server.R
   ├── 📂 R/
   │    ├── mod_*.R  ← 1 module / écran
   │    ├── model_*.R  ← accès données
   │    └── service_*.R  ← logique métier
   ├── 📂 tests/  ← testthat
   └── 📂 www/  ← assets
```

Séparer les responsabilités : le métier (`model` / `service`) d'un côté, l'interface (modules) de l'autre.

POUR ALLER PLUS LOIN : article « Les bonnes pratiques de développement Shiny » · **datachamp.fr/rr26**

2 · Reproductibilité

« Ça marche sur ma machine… »

« Alors on déploie ta machine. »

1. Même version de R.
2. Gestionnaire de packages comme `renv`
3. Fichier `.Renviron` pour les variables d'environnement

> POUR ALLER PLUS LOIN : article « Gérer ses dépendances avec renv » · **`datachamp.fr/rr26`**

3 · Qualité du code

RÉFLEXE SCRIPT

```
project_gap_table_prep23
project_gap_table_prepc2s3
project_gap_table_prepc3s3
# … 60 variantes, copié-collé
```

Noms cryptiques, indentation au hasard, **zéro test**. Le code « tombe en marche ».

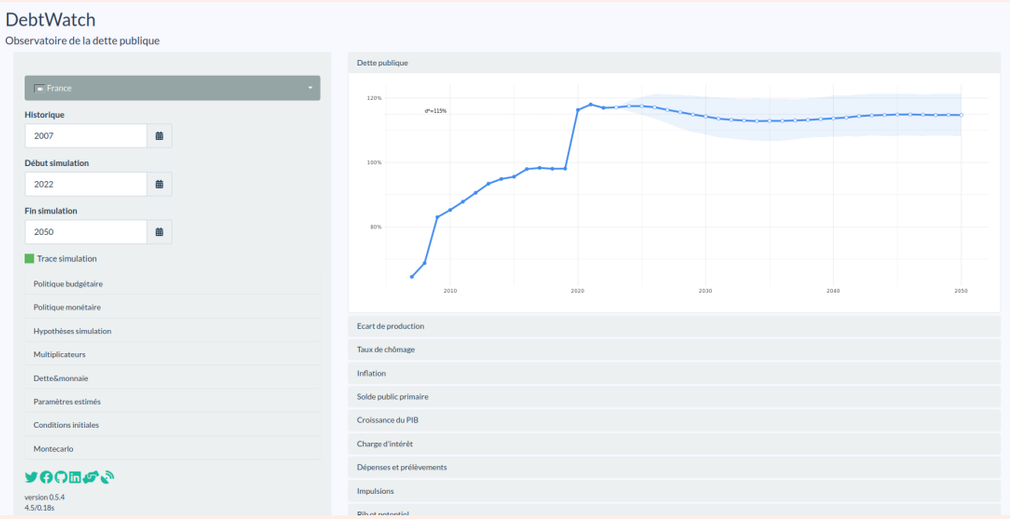
LA BONNE APPROCHE

- **Standards** appliqués par `lintr` + `styler` en **CI**
- **Code propre** : noms parlants, **factoriser** (DRY)
- Tests : **bug → fix → test**

39 043 lignes → ~15 000, mêmes features.

POUR ALLER PLUS LOIN : article « Assurer la qualité du code » · **`datachamp.fr/rr26`**

4 · Conception produit



Sobre, mais **générique et sans identité** : on **doute de l'outil.**

Une **vraie identité** (ici l'OFCE) : maquette + Sass. Perçue plus **pro.**

POUR ALLER PLUS LOIN : article « Concevoir une UI Shiny moderne » *(à paraître)* · **`datachamp.fr/rr26`**

5 · Déploiement

shinyapps.io

Rapide, zéro infra.
Mais : pas de stockage persistant, machines limitées, **non-souverain**.
→ prototype, partage rapide.

VM ~10 €/mois

nginx (HTTPS) + déploiement auto (`git push`).
Mais : héberger = **un vrai métier** (sécu, MAJ, maintenance).
→ bon défaut pour 90 % des cas.

AWS / cloud

Puissant, scalable.
Mais : **coûteux ET complexe**.
→ un piège dans la plupart des cas.

> Pas de réponse unique : **ça dépend du contexte**.

POUR ALLER PLUS LOIN : article « Déployer une app Shiny sur une VM » · **`datachamp.fr/rr26`**

6 · Sécurité

Objectif : protéger l'accès à l'appli (vraies données derrière), sans devenir soi-même expert en sécurité.

LA SOLUTION DE FACILITÉ

Un package "maison" (ex. `shinymanager`).
Très bien en **quick & dirty**, ou si l'accès est déjà protégé autrement (réseau, DSI).
Mais en **public** : pas maintenu par des experts sécu, et **il manque des features**.

CE QUE JE CONSEILLE

Auth0 : un service tiers où **tout est déjà là**. La sécu, c'est leur métier.
Gestion des accès et des **rôles**.
Portail complet : login, signup, mot de passe perdu, tous les flows.
Bémol : dépendance à un service tiers US.

POUR ALLER PLUS LOIN : article « Sécuriser une app Shiny avec Auth0 » · **`datachamp.fr/rr26`**

Votre feuille de route

Étape	Pour aller plus loin
1 · Structure	Bonnes pratiques de développement
2 · Reproductibilité	Gérer ses dépendances avec renv
3 · Qualité	Assurer la qualité du code
4 · Conception	Concevoir une UI Shiny
5 · Déploiement	Déployer sur une VM
6 · Sécurité	Sécuriser avec Auth0

Vous repartez avec la **carte**. À vous de **creuser** chaque sujet, à votre rythme.

`datachamp.fr/rr26`

On est sur le **stand** toute la conf.

Merci !

DATA CHAMP