Description du programme

Application web de gestion et d'édition de timeway pour installations de traitement de surface (galvanoplastie, phosphatation, etc.).

Version : 2.1.0 —

Description

VKTimeway permet de définir, visualiser et éditer graphiquement les cycles de transfert d'un ou de plusieurs robots sur une ligne de traitement de surface. L'application gère :

- Les **machines** (lignes de traitement) avec leur historique de versions
- Les **positions** (bains, postes de dépose/prise) par ligne
- Les **robots** (hoists) avec leurs paramètres cinématiques
- Les **séquences** de traitement par produit
- Les **timeway** (diagrammes espace-temps) générés automatiquement ou édités manuellement
- L'**éditeur graphique canvas** interactif des timeway
- Le **simulateur de production** avec visualisation multi-cycles
- La **génération de rapports PDF** pour chaque entité

Architecture

Application **PWA vanilla JS** sans framework, fonctionnant entièrement côté client.

```
App/
├── index.html                Point d'entrée unique (SPA)
├── manifest.json             Manifeste PWA
├── sw.js                     Service Worker (cache offline)
└── assets/
    ├── js/
    │   ├── VKApp.js          Routeur SPA + logique de toutes les pages
    │   ├── VKModels.js       Constructeurs des modèles de données
    │   ├── VKData.js         Persistance localStorage
    │   ├── VKDiagram.js      Moteur de rendu canvas du timeway
```

```
    │   ├── VKTimewayGen.js     Générateur automatique de timeway
    │   ├── VKReport.js         Génération de rapports PDF (pdfmake)
    │   ├── VKI18n.js           Internationalisation (FR / EN / DE / ZH)
    │   ├── pdfmake.min.js      Librairie PDF (embarquée, offline)
    │   └── vfs_fonts.min.js    Polices pdfmake (Roboto, embarquées)
    ├── css/
    │   └── app.css             Styles spécifiques à l'application
    └── img/
        └── icons/              Icônes PNG de la barre d'outils de l'éditeur
```

Modules JS

Fichier	Rôle
`VKApp.js`	Routeur hash (`#/`, `#/machines`, ...), rendu des pages, modales d'édition, simulateur
`VKModels.js`	Constructeurs `Machine`, `Version`, `Position`, `Hoist`, `Sequence`, `Timeway`, ...
`VKData.js`	Lecture / écriture dans `localStorage`
`VKDiagram.js`	Rendu canvas : axes, étapes, aides visuelles, détection collision
`VKTimewayGen.js`	Génération automatique d'un timeway depuis une séquence
`VKReport.js`	Rapports PDF via pdfmake : machine, séquence, timeway, diagramme
`VKI18n.js`	Dictionnaire quadrilingue + injection dans le DOM (`data-i18n`)

Modèle de données

```
Machine                 Id, Name, Number, Client, Location, Remarks, CreatedAt, UpdatedAt
└── Version[]           Id, Number, Comment, Remarks, CreatedAt, UpdatedAt
    ├── Positions[]     id, name, flag, group, tank, address, branch, lift, lower, distance
    ├── Hoists[]        id, pi, pf, width, front, back, speed1 (mm/s), accdec (ms), lifts[],
lowers[]
    ├── Sequences[]     id, name, remarks, steps[]
    ├── Timeways[]      id, name, remarks, cycletime (ms), visible, steps[]
    ├── LiftTypes[]     10 types de montée nommés (id, name, time)
    └── LowerTypes[]    10 types de descente nommés (id, name, time)
```

Toutes les données sont stockées en JSON dans `localStorage` sous les clés `tw_machines` et `tw_versions`.

Fonctionnalités principales

Gestion des machines

- Création / modification / suppression de machines
- Champs : nom, numéro, client, localisation, remarques
- Horodatage automatique de création et de dernière modification
- Export / import d'une machine au format JSON (transfert entre postes)

Export / Import des données

- **Export global** (page d'accueil) : télécharge un snapshot complet du `localStorage` (toutes les machines et versions) au format JSON
- **Import global** : restaure un snapshot précédemment exporté (remplace toutes les données existantes après confirmation)
- **Export machine** : exporte une machine et toutes ses versions dans un fichier JSON individuel
- **Import machine** : importe un fichier machine JSON dans l'application

Versions

- Chaque machine peut avoir plusieurs versions indépendantes
- Commentaire et remarques sur chaque version
- Copie et suppression de versions

Positions, robots (hoists), séquences

- Édition en liste avec validation
- Copie de séquences existantes
- Identifiants de séquences auto-incrémentés

Types montées / descentes

- 10 types nommés par version pour les montées et les descentes
- Le type de montée et de descente est conservé dans chaque étape de timeway généré (`liftType`, `lowerType`)

- Référencés par chaque position et chaque étape de séquence

Génération automatique de timeway

- Calcul des temps de transfert avec profil de vitesse trapézoïdal :
  - Distance longue : `t = (d/v + ta/2) × 1000` ms
  - Distance courte : `t = √(2 × ta × d / v) × 1000` ms
  - où `v` = vitesse en mm/s, `ta` = temps acc/déc en s
- Attribution automatique des robots aux étapes
- Calcul optionnel des mouvements à vide (retours)

Éditeur graphique de timeway

- Canvas interactif avec zoom et pan
- **Redimensionnement automatique** du canvas lorsque la fenêtre de l'éditeur change de taille (`ResizeObserver`)
- Sélection du timeway actif via menu déroulant (titre mis à jour dynamiquement)
- Outils : sélection, déplacement, déplacement bloc gauche/droite, ajout d'étapes, effacement, ajustement des temps
- Options d'affichage persistées dans `localStorage` :
  - Grille des temps (`timeGrid`)
  - Grille des positions (`posGrid`)
  - Marques montée/descente (`upDown`)
  - Aides mouvement vides (`emptyMove`)
  - Empreinte bras (`hoistRange`)
  - Liens (`showLink`)
  - **Regroupement des positions** (`collapseGroups`) — les positions d'un même groupe sont affichées sur une seule ligne
- Icônes actives en jaune, inactives en blanc, survol en gris
- Détection visuelle des collisions entre robots
- Cercles d'intersection sur les zones de croisement de temps clés
- Robot sélectionné mis en évidence en jaune (légende mise à jour en conséquence)
- **Curseur interactif** : nom de la position en jaune au survol + ligne verticale pointillée sur l'axe du temps

Simulateur de production

- Fenêtre plein écran de simulation multi-cycles
- Ajout de diagrammes dans la file de simulation avec vérification des conflits :
  - Détection d'occupation de bain (positions sans groupe)
  - Substitution automatique de position dans un groupe (flag `gf`) si la position cible est occupée
  - Collision de robots entre cycles

- **Algorithme de décalage de cycle** :
  - Les temps `ti`/`tf` sont arrondis à la seconde
  - Si un step dépose dans un groupe (`gf != 0`) et que `tminf > cycleTime`, le temps d'immersion effectif est calculé : `(tf−ti) + floor(tminf / cycleTime) × cycleTime`
  - L'ajout de cycles basé sur `tminf` ne s'applique qu'aux positions avec groupe
- **Positions avec flag 1, 2, 3 ou 7** : le test d'occupation de bain est ignoré
- Visualisation canvas interactive :
  - Axe du temps avec graduation en minutes (`#N` par slot de cycle)
  - Axe des positions avec défilement vertical (clic + glisser)
  - Position survolée mise en évidence en jaune
  - Barres d'immersion colorées (cyan = normal, orange = < tminf, rouge = > tmaxf)
  - Pan temporel par clic + glisser horizontal
  - Zoom temporel par molette de la souris
- Liste des cycles chargeables repliable (panneau latéral)
- Impression du diagramme de simulation

Rapports PDF

Générés côté client via **pdfmake** (aucun serveur requis, fonctionne offline). Chaque rapport est téléchargé directement en `.pdf`.

Rapport	Déclencheur	Contenu
Fiche machine	Bouton dans l'onglet Positions	Positions · Robots · Types montées/descentes · Remarques
Fiche séquence	Bouton par ligne dans l'onglet Séquences	Étapes (position, groupe, montée, descente, Tmin/Tmax/Tégouttage) · Remarques
Fiche timeway	Bouton par ligne dans l'onglet Timeway	Étapes avec tous les temps (Ti, Tf, Tégouttage, Tmontée, Tdescente, Ttransfert) · Remarques
Rapport diagramme	Bouton dans la toolbar de l'éditeur	Capture du canvas + table des étapes · Remarques (orientation paysage si nécessaire)

Structure commune des rapports :

- Bandeau d'en-tête (machine, version, titre du rapport, client/localisation)
- Tables avec lignes alternées et en-têtes répétés sur chaque page
- Footer : copyright · numéro de page · date de génération

Internationalisation

Quatre langues supportées : **Français** (défaut), **English**, **Deutsch**, □□ (chinois simplifié).

La langue est sélectionnable depuis la **page d'accueil** et persistée dans `localStorage`. Le changement est instantané sans rechargement de page.

Utilisation dans le HTML : `data-i18n="clé"` sur n'importe quel élément. Utilisation dans le JS : `VKI18n.t('clé')`.

Installation / Démarrage

Développement

Aucune dépendance serveur. L'application fonctionne directement depuis un navigateur.

1. Cloner le dépôt
2. Servir le dossier `App/` via un serveur HTTP local (requis pour le Service Worker) :

   ```
   npx serve App/
   # ou
   python -m http.server 8080 --directory App/
   ```

3. Ouvrir `http://localhost:8080` dans un navigateur moderne

> **Note :** L'ouverture directe du fichier `index.html` (`file://`) ne fonctionnera pas avec le Service Worker. Un serveur HTTP est nécessaire.

Build de distribution (version minifiée)

Un script de build génère une version optimisée dans `Dist/` :

```
npm install       # une seule fois
npm run build     # génère Dist/
```

Le build effectue :

- Concaténation + minification des 7 fichiers JS applicatifs → `app.bundle.min.js` (~54% de réduction)
- Minification de `app.css` → `app.min.css` (~37% de réduction)
- Copie des librairies tierces déjà minifiées
- Mise à jour de `index.html` pour référencer les fichiers minifiés

- Génération d'un `sw.js` avec un nom de cache dédié (`vktimeway-dist-v*`)

> **Important :** Incrémenter `SW_CACHE_VERSION` dans `build.js` à chaque nouveau déploiement pour forcer la mise à jour du cache chez les utilisateurs.

Pour servir la version de distribution :

```
npx serve Dist/
```

Dépendances front-end (embarquées)

Librairie	Version	Usage
Bootstrap	5.x	UI composants, grille, modales
Font Awesome	6.x	Icônes vectorielles
pdfmake	0.2.10	Génération de rapports PDF côté client

Toutes les dépendances sont incluses dans `App/assets/` — aucune connexion CDN requise.

Dépendances de build (développement uniquement)

Package	Usage
`terser`	Minification JavaScript
`clean-css`	Minification CSS
`fs-extra`	Opérations fichiers avancées

Structure des routes

Hash	Page
`#/`	Accueil (sélection de langue, export/import données)
`#/machines`	Liste des machines

Hash	Page
`#/machines/create`	Nouvelle machine
`#/machines/:id`	Détail machine + liste des versions
`#/machines/:id/version/:vid`	Édition d'une version (positions, robots, séquences, timeway)

Versioning de l'application

La version est définie par la constante `VK_VERSION` en tête de `VKApp.js` :

```
var VK_VERSION = 'ver. 2.1.0';
```

Elle s'affiche automatiquement dans le footer de l'application et est intégrée dans les métadonnées et les noms de fichiers des rapports PDF générés.

Revision #5
Created 2026-03-24 09:25:49 UTC by Jean-Noël Voirol
Updated 2026-03-24 09:42:36 UTC by Jean-Noël Voirol