Année universitaire 2024-2025

**RAPPORT DE STAGE**

*CRÉATION D’INTERFACE DE PILOTAGE DE SOLVEUR D’EMPLOIS DU TEMPS UNIVERSITÉ*

***Encadré par :***

*M. David Lesaint, M. David Genest, M. Marc Legeay,*

*M.Vincent Barichard, M. Aurélien Simon, M. Corentin Behuet*

***Réalisé par :***

*Mme. BUI Thi Vi*

*M. ANANI Messan Joseph*

*M. NDAYIZEYE Guy Keny*



# REMERCIEMENT

Nous tenons à exprimer notre sincère gratitude à l'Université d'Angers et au Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA) pour nous avoir offert l'opportunité de réaliser ce stage enrichissant.

Nos remerciements s'adressent particulièrement à notre équipe d'encadrants : M. David Lesaint, M. David Genest, M. Marc Legeay, M. Vincent Barichard, M. Aurélien Simon et M. Corentin Behuet. Leur expertise, leurs conseils pertinents et leur disponibilité ont été déterminants dans la réussite de ce projet. Leurs retours constructifs lors de nos réunions hebdomadaires nous ont permis d'améliorer continuellement notre travail et d'approfondir notre compréhension des problématiques liées à la génération d'emplois du temps universitaires.

Nous souhaitons également remercier l'ensemble du personnel du LERIA pour leur accueil chaleureux et le cadre de travail stimulant qu'ils nous ont offert tout au long de cette période.

Cette expérience a été particulièrement formatrice, tant sur le plan technique que sur celui de la gestion de projet en équipe, et constituera sans aucun doute un atout majeur dans notre parcours académique et professionnel.

# SOMMAIRE

[REMERCIEMENT 2](#_Toc198044413)

[CONTACTS 2](#_Toc198044414)

[LIENS DE NOTRE PROJET 2](#_Toc198044415)

[SOMMAIRE 3](#_Toc198044416)

[PRÉSENTATION DE NOTRE PROJET 6](#_Toc198044417)

[**I. Contexte : 6**](#_Toc198044418)

[**II. Objectifs du projet : 6**](#_Toc198044419)

[**III. Architecture technique : 7**](#_Toc198044420)

[RÉALISATION DE PROJET 7](#_Toc198044421)

[**I.** **Interface utilisateur :** 7](#_Toc198044422)

[1. Technologies utilisées 7](#_Toc198044423)

[2. Structure générale de l'interface 8](#_Toc198044424)

[2.1 Page d'accueil 8](#_Toc198044425)

[2.2 Section de définition du périmètre 9](#_Toc198044426)

[2.3 Section de filtrage des règles 10](#_Toc198044427)

[2.4 Section de filtrage des ressources 14](#_Toc198044428)

[2.5 Section de configuration du solveur 18](#_Toc198044429)

[2.6 Section des solutions 20](#_Toc198044430)

[**II.** **Interaction avec les API :** 22](#_Toc198044431)

[1. Communication avec l'API de données 22](#_Toc198044432)

[2. Communication avec l'API du solveur 23](#_Toc198044433)

[3. Gestion des données et transformation 24](#_Toc198044434)

[4. Gestion des erreurs et feedback utilisateur 24](#_Toc198044435)

[**III.** **Fonctionnalités implémentées :** 25](#_Toc198044436)

[**IV.** **Difficultés rencontrées et solutions apportées :** 26](#_Toc198044437)

[**V.** **Résultats obtenus :** 26](#_Toc198044438)

[1. Interface utilisateur intuitive 27](#_Toc198044439)

[2. Fonctionnalités complètes de sélection et filtrage 27](#_Toc198044440)

[3. Configuration flexible du solveur 27](#_Toc198044441)

[4. Visualisation des résultats 27](#_Toc198044442)

[5. Tests et validation 27](#_Toc198044443)

[6. Documentation 28](#_Toc198044444)

[**VI. Conclusion :** 28](#_Toc198044445)

[1. Bilan du projet 28](#_Toc198044446)

[2. Compétences développées 28](#_Toc198044447)

[3. Perspectives d'évolution 29](#_Toc198044448)

[4. Mot de la fin 29](#_Toc198044449)

[ANNEXES 30](#_Toc198044450)

**INTRODUCTION**

Ce rapport présente le travail réalisé par notre équipe de trois stagiaires au sein du Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA) durant notre période de stage du 24 mars 2025 à 20 mai 2025. Notre mission consistait à développer une interface web permettant de piloter un solveur d'emplois du temps universitaire développé par le laboratoire.

La génération d'emplois du temps universitaires est un problème complexe qui nécessite la prise en compte de nombreuses contraintes : disponibilité des enseignants et des salles, répartition cohérente des cours dans la semaine, respect des maquettes pédagogiques, etc. Le LERIA a développé un solveur performant pour traiter ce problème, mais son utilisation requiert actuellement des compétences techniques avancées.

Notre objectif principal était donc de créer une interface intuitive permettant aux utilisateurs non-spécialistes, comme les responsables de formation ou le personnel administratif, d'utiliser efficacement ce solveur. Cette interface devait permettre de sélectionner les données en entrée, configurer les paramètres du solveur, lancer les calculs et visualiser les résultats.

Dans ce rapport, nous présenterons d'abord le contexte et les objectifs détaillés du projet, puis nous décrirons notre démarche de conception et de réalisation. Nous exposerons ensuite les fonctionnalités développées, les difficultés rencontrées et les solutions apportées. Enfin, nous conclurons par un bilan de cette expérience et les perspectives d'évolution de l'application.

# PRÉSENTATION DU PROJET

1. **Contexte :**

Le Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA) travaille depuis plusieurs années sur le développement de solutions algorithmiques pour résoudre le problème complexe de la génération d'emplois du temps universitaires. Cette problématique constitue un défi majeur pour les établissements d'enseignement supérieur en raison des nombreuses contraintes à respecter : disponibilité des salles, des enseignants, cohérence des parcours étudiants, règles pédagogiques, etc.

Dans ce cadre, le LERIA a développé un solveur performant capable de calculer des solutions optimisées pour la création d'emplois du temps universitaires. Ce solveur, basé sur des techniques avancées de recherche opérationnelle et de programmation par contraintes, peut traiter des instances complexes comportant de nombreuses variables et contraintes.

Jusqu'à présent, l'utilisation de ce solveur nécessitait des compétences techniques spécifiques et une compréhension approfondie de son fonctionnement interne, ce qui limitait son accessibilité aux utilisateurs non-spécialistes comme les responsables de formation ou le personnel administratif.

1. **Objectifs du projet :**

Notre mission consiste à développer une interface utilisateur intuitive et fonctionnelle permettant de piloter le solveur d'emplois du temps. Cette interface doit servir d'intermédiaire entre les utilisateurs finaux et le système algorithmique complexe sous-jacent. Plus précisément, notre application doit permettre de :

1. **Définir le périmètre des données** : Permettre aux utilisateurs de rechercher, sélectionner et combiner des formations, des périodes et d'autres paramètres pour délimiter précisément le périmètre de l'emploi du temps à générer.
2. **Filtrer les règles et contraintes** : Offrir une interface permettant de gérer les différentes règles et contraintes du solveur, tant fondamentales que métier, en adaptant le modèle aux besoins spécifiques de l'utilisateur.
3. **Gérer les ressources** : Faciliter la sélection et le filtrage des cours, enseignants et salles à inclure dans le processus de génération, optimisant ainsi le temps de calcul et la pertinence des résultats.
4. **Configurer le solveur** : Proposer une interface intuitive pour ajuster les paramètres techniques du solveur (stratégies des variables, temps d'exécution, etc.) sans nécessiter de connaissances avancées, tout en offrant des options pour les utilisateurs experts.
5. **Visualiser et exploiter les résultats** : Suivre l'avancement du processus de résolution, afficher les solutions générées sous forme de statistiques et d'aperçus structurés, et permettre leur visualisation détaillée, leur sauvegarde et leur export.
6. **Architecture technique :**

Notre application s'appuie sur une architecture client-serveur où le frontend interagit avec deux API REST distinctes :

**API de données** : Cette API permet de récupérer les informations nécessaires à la constitution des instances (formations, périodes, enseignants, salles, cours, etc.).

**API du solveur** : Cette seconde API permet de configurer le solveur, de lancer les calculs et de récupérer les solutions générées ainsi que des statistiques sur le processus de résolution.

# RÉALISATION DU PROJET

1. **Interface utilisateur :**

Pour le développement de l'interface utilisateur, nous avons adopté une approche centrée sur l'expérience utilisateur, visant à rendre l'utilisation du solveur accessible même aux personnes sans connaissances techniques approfondies. Nous avons opté pour une structure claire et intuitive, organisée autour de cinq sections principales qui suivent le flux de travail naturel des utilisateurs.

### **Technologies utilisées**

Conformément aux spécifications du projet, nous avons développé l'interface en utilisant les technologies web standards :

* **HTML5** pour la structure de base de l'application
* **CSS3** pour la mise en forme et les animations
* **JavaScript** pour l'interactivité et la communication avec les API
* **Bootstrap 5** comme framework CSS pour assurer une interface responsive et esthétique

**Ce framework a été retenu pour plusieurs raisons déterminantes.** Son système de grille responsive permet d'optimiser efficacement l'agencement du contenu sur tous les appareils. Il offre par ailleurs une collection de composants modulaires prêts à l'emploi, notamment pour la navigation, les cartes, les formulaires et les alertes, ce qui accélère significativement le développement. Enfin, ses nombreux utilitaires CSS intégrés facilitent la gestion des espacements et des alignements sans nécessiter de code supplémentaire.

**La personnalisation de l'apparence a été réalisée via des feuilles de style dédiées (style.css, tabs\_navigation.css, config\_tab.css), en exploitant les variables CSS de Bootstrap pour garantir une cohérence visuelle globale**

### **Structure générale de l'interface**

L'interface est organisée autour d'une page d'accueil servant de point d'entrée, suivie de cinq sections principales accessibles via une barre d'onglets horizontale:

1. **Page d'accueil** : Permet de choisir entre créer une nouvelle configuration ou charger une configuration existante.
2. **Définition du périmètre** : Permet de rechercher, filtrer et sélectionner les instances sur lesquelles travailler.
3. **Filtrage des règles** : Permet de gérer les contraintes fondamentales et métier du solveur.
4. **Filtrage des ressources** : Permet de sélectionner les cours, enseignants et salles à utiliser.
5. **Configuration du solveur** : Offre différentes options pour paramétrer le fonctionnement du solveur.
6. **Solutions** : Affiche les résultats générés et les statistiques associées.

Cette structure séquentielle guide l'utilisateur à travers les différentes étapes du processus de génération d'emploi du temps, depuis le choix initial du mode de travail jusqu'à la visualisation des résultats.

#### **Page d'accueil**

La page d'accueil constitue le point d'entrée de l'application et présente à l'utilisateur deux options principales, clairement présentées sous forme de cartes interactives :

* **Nouvelle configuration** : Permet de créer une configuration complètement nouvelle en définissant un périmètre, des règles et des ressources depuis le début.
* **Configuration existante** : Permet de charger une configuration précédemment sauvegardée afin de la consulter, la modifier ou l'utiliser comme base pour une nouvelle résolution.

Cette page d'accueil a été conçue avec une approche minimaliste et intuitive, utilisant des icônes (via Bootstrap Icons) et des descriptions concises pour guider rapidement l'utilisateur vers le choix qui correspond à son besoin. L'interface visuelle utilise des cartes avec des effets de survol pour améliorer l'expérience utilisateur.

La mise en page est structurée de manière à occuper tout l'écran, garantissant que l'utilisateur se concentre exclusivement sur ce choix initial important. Les deux options sont présentées côte à côte dans un conteneur central, chacune avec une icône distinctive, un titre, une description courte et un bouton d'action.

Lorsque l'utilisateur choisit l'option "Configuration existante", une fenêtre modale s'ouvre, affichant la liste des configurations disponibles avec leurs détails (nom, date), permettant ainsi une sélection éclairée.



*Figure 1 : Page d'accueil de l'application*

Sur le plan technique, la logique de cette page s'appuie sur une architecture simple mais efficace. Les fichiers dédiés gèrent respectivement la présentation visuelle et le comportement dynamique, offrant ainsi une séparation claire des responsabilités. Cette conception permet à l'utilisateur de choisir rapidement entre création et réutilisation, optimisant le flux de travail général de l'application.

#### **D****éfinition du périmètre**

La première section de notre interface permet aux utilisateurs de rechercher et sélectionner les formations et périodes sur lesquelles travailler. Cette section a été conçue pour être intuitive et offrir une expérience utilisateur fluide.

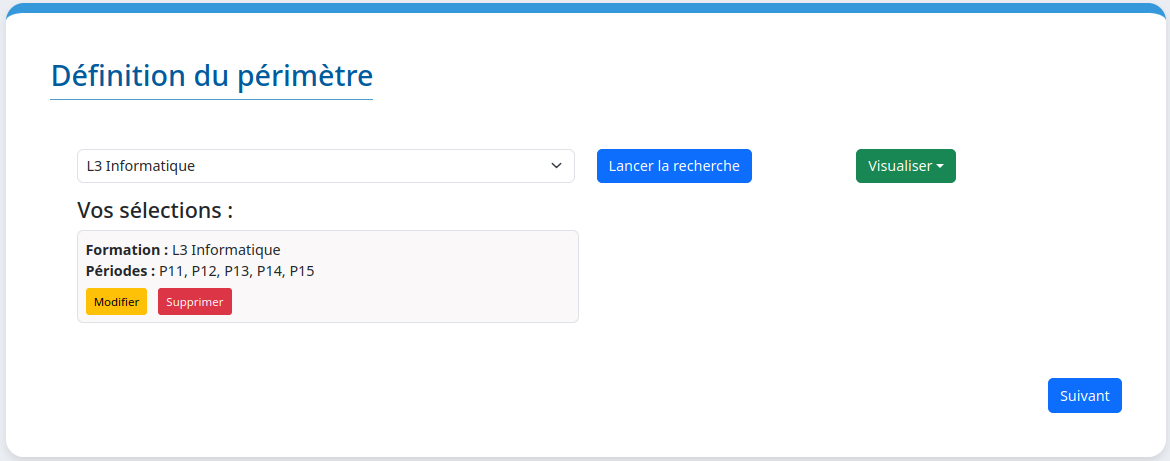
Nous avons implémenté une barre de recherche avec autocomplétion qui affiche des suggestions en temps réel. Cette fonctionnalité permet de trouver rapidement une formation spécifique sans parcourir toute la liste disponible.

En complément, un menu déroulant permet de filtrer les instances par type de formation. Une fois une formation sélectionnée, l'interface affiche automatiquement les périodes associées (ou semestres) que l'utilisateur peut sélectionner individuellement ou globalement grâce au bouton "Tout sélectionner".

Un résumé des choix effectués s'affiche ensuite dans une zone dédiée avec la possibilité de modifier ou supprimer des sélections existantes. Cette approche permet à l'utilisateur de visualiser clairement ses choix avant de lancer la recherche des données. Un bouton "Visualiser" permet d'accéder à un menu déroulant offrant deux options : "XML" pour afficher le fichier XML correspondant à la sélection et "JSON" pour visualiser les données au format JSON.

Le code JavaScript associé à cette section utilise des fonctions asynchrones pour communiquer avec l'API et récupérer les données nécessaires :

* fetchFormations() pour charger la liste des formations disponibles
* fetchPeriods(formationId) pour obtenir les périodes d'une formation spécifique
* updateSummaryDisplay() pour afficher le résumé des sélections
* fetchPerimetersData() pour récupérer les données complètes du périmètre
* generateCompleteTimetabling() pour générer le XML correspondant

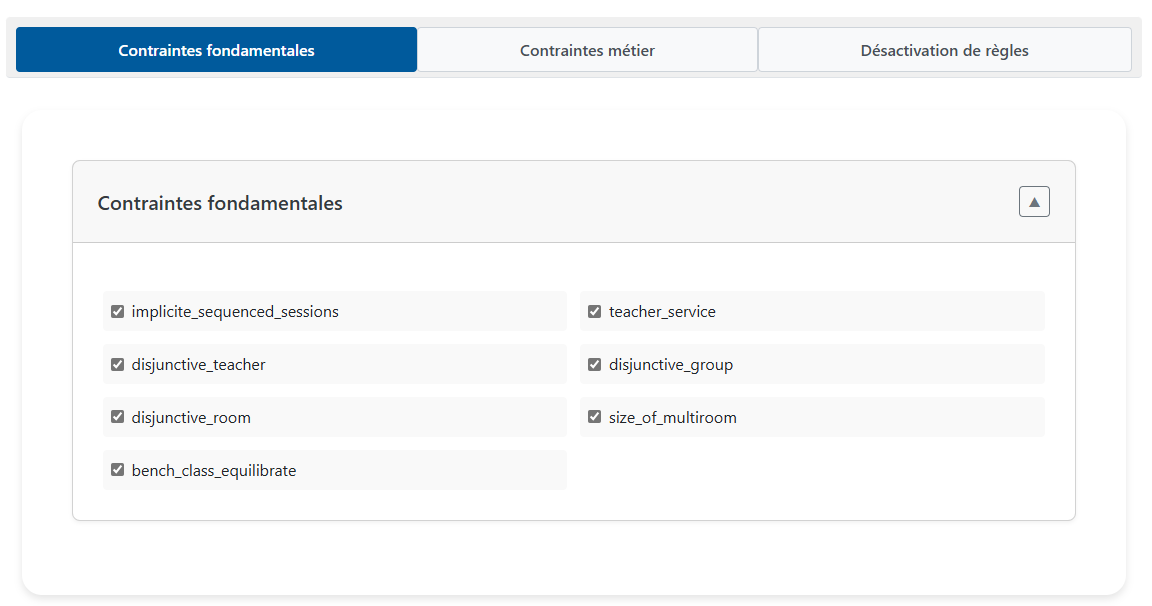


***Figure 2 :*** *Interface de définition du périmètre*

#### **F****iltrage des règles**

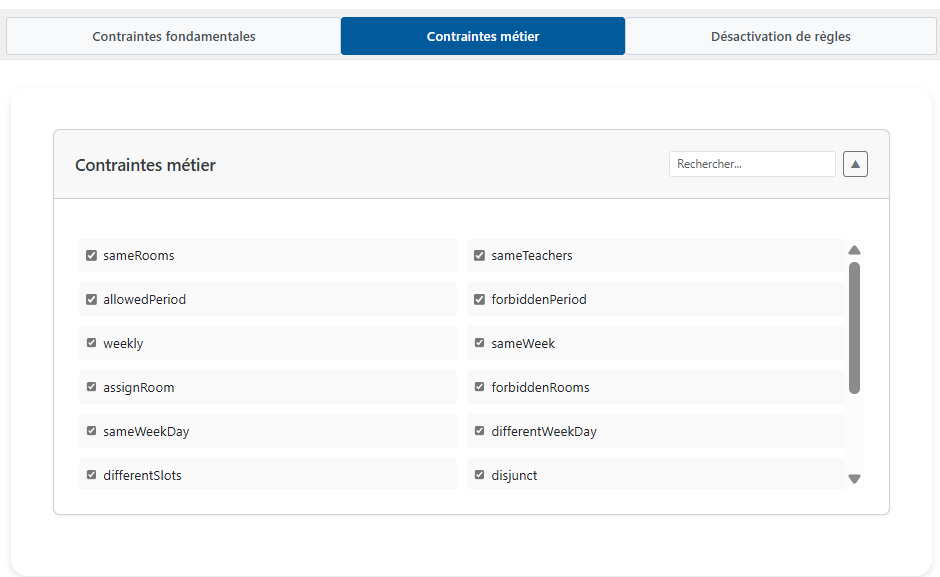
La deuxième section de notre interface permet aux utilisateurs de gérer les règles et contraintes du solveur. Compte tenu du nombre important d'options disponibles, nous avons organisé cette section en trois sous-onglets pour une navigation claire et intuitive :

1. **Contraintes fondamentales** : Cette partie permet d'activer ou désactiver les contraintes de base nécessaires au bon fonctionnement du solveur comme disjunctive\_teacher, disjunctive\_group, disjunctive\_room, teacher\_service, etc. Ces contraintes sont essentielles pour assurer la cohérence des emplois du temps générés.

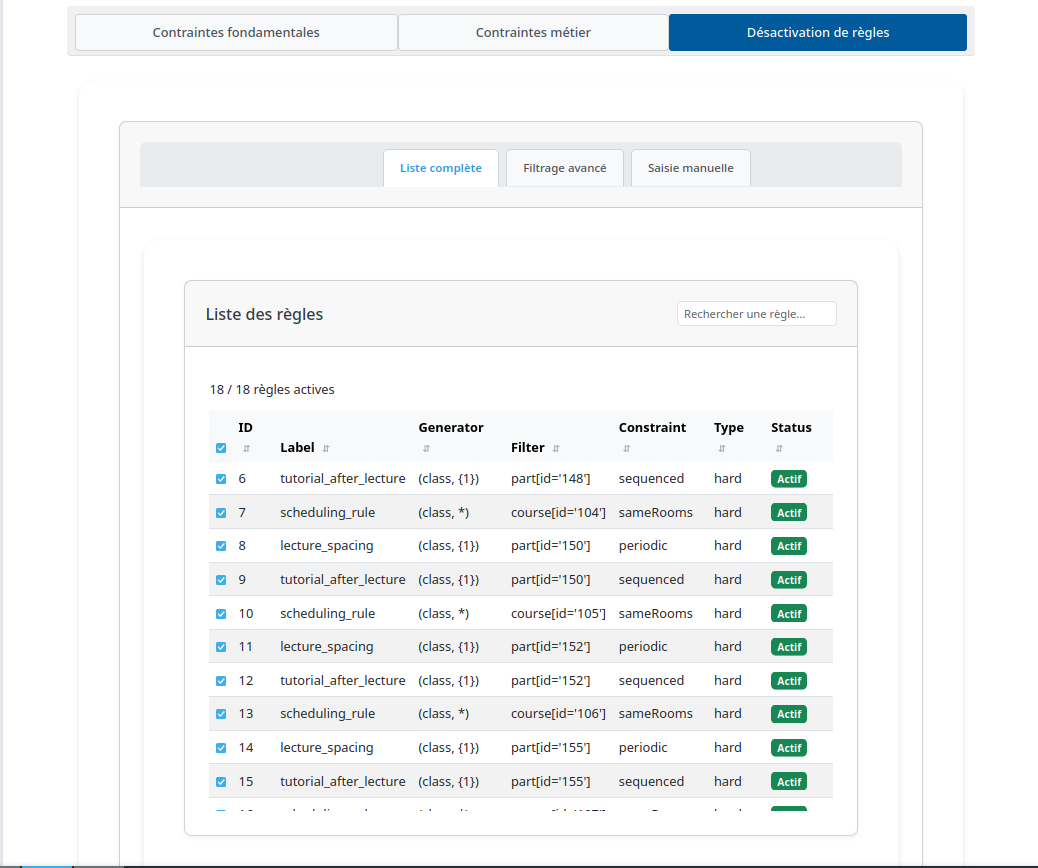


*Figure 3 : Interface des contraintes fondamentales*

1. **Contraintes métier** : Cet onglet offre la possibilité de gérer les contraintes spécifiques au domaine de l'emploi du temps universitaire, comme sameRooms, allowedPeriod, weekly, forbiddenPeriod, etc. Un champ de recherche dédié facilite la navigation parmi ces nombreuses contraintes, permettant aux utilisateurs de trouver rapidement des contraintes spécifiques.

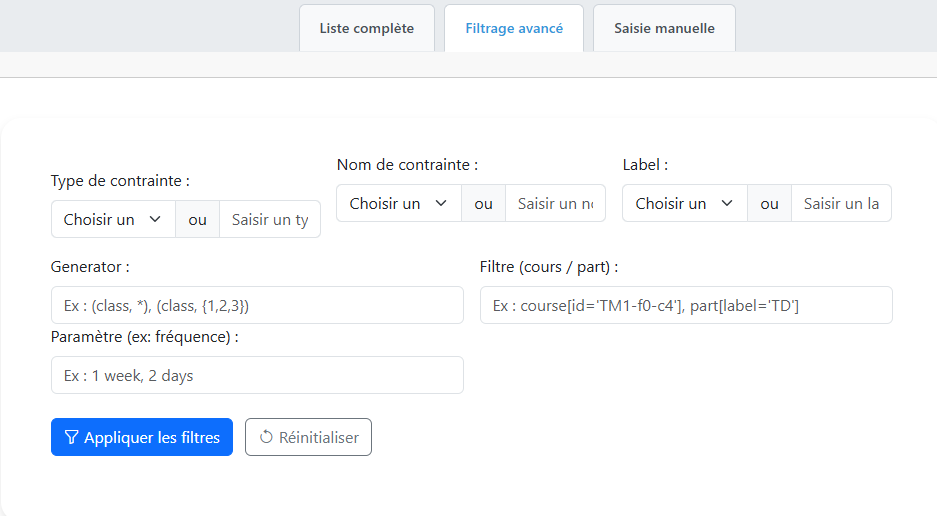


*Figure 4 : Interface des contraintes métier*

1. **Désactivation de règles** : Ce dernier sous-onglet propose trois modes différents de désactivation de règles :
   * **Liste complète** : Affiche toutes les règles dans un tableau interactif avec possibilité de filtrage par texte et tri par colonne

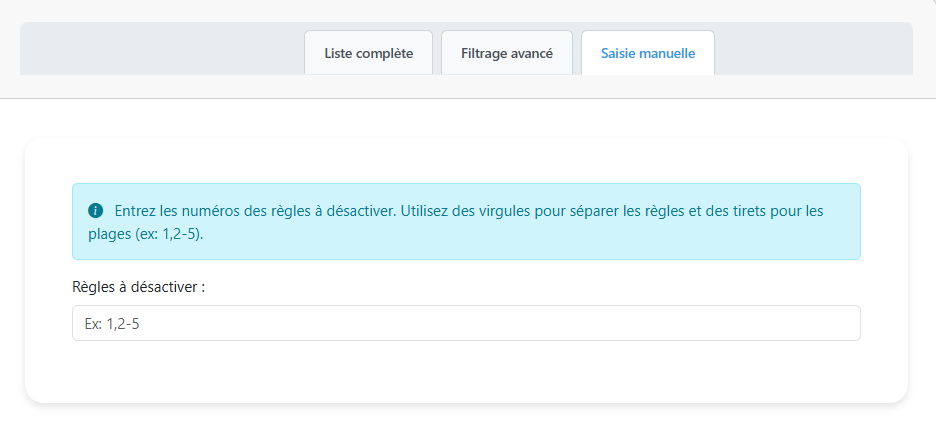
***Figure 5 :*** *Interface de désactivation de règles - Liste complète*

* + **Filtrage avancé** : Permet de filtrer les règles selon divers critères (type de contrainte, nom, label, générateur, etc.)



*Figure 6 : Interface de désactivation de règles - Filtrage avancé*

* + **Saisie manuelle** : Permet de désactiver des règles en entrant directement leurs numéros, avec support des plages (ex: 1-5,8,10-12)



*Figure 7 : Interface de désactivation de règles - Saisie manuelle*

Pour la liste complète, nous avons implémenté un tableau interactif avec des fonctionnalités de tri pour chaque colonne (ID, label, générateur, filtre, contrainte, type, statut), ainsi qu'une barre de recherche pour filtrer rapidement les règles. Un compteur indique le nombre de règles actives par rapport au total.

Chaque règle est représentée par une ligne dans le tableau avec une case à cocher permettant de l'activer ou la désactiver facilement. Le statut actif/inactif est clairement visible grâce à un badge coloré (vert pour actif, rouge pour inactif) et une opacité réduite pour les règles inactives.

La fonctionnalité de filtrage avancé est particulièrement utile pour les utilisateurs travaillant avec un grand nombre de règles. Elle permet de combiner plusieurs critères pour cibler précisément les règles à modifier :

* Type de contrainte (hard/soft)
* Nom de contrainte avec suggestion des valeurs courantes
* Labels associés aux règles
* Générateurs et filtres pour les utilisateurs avancés

Le mode de saisie manuelle offre une alternative rapide pour les utilisateurs qui connaissent déjà les identifiants des règles à désactiver. Il supporte aussi bien les identifiants individuels que les plages, permettant par exemple de désactiver rapidement un ensemble de règles connexes.

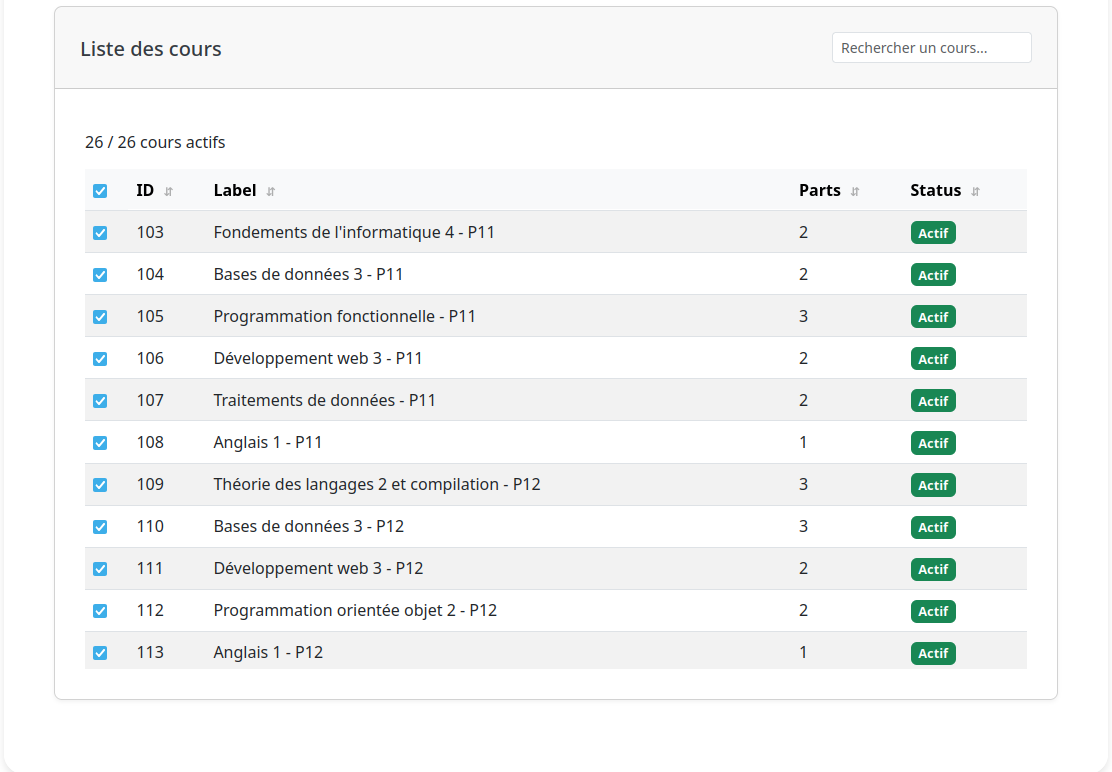
Le code JavaScript associé à cette section inclut des fonctions comme :

* renderRulesDetails(data) pour afficher les règles dans le tableau
* sortRulesTable(field, direction) pour le tri des colonnes
* applyAdvancedFilters() pour le filtrage multicritère
* parseRuleInput(input) pour la saisie manuelle des règles à désactiver

#### **F****iltrage des ressources**

La troisième section de notre interface est dédiée à la gestion des ressources utilisées par le solveur. Cette étape est cruciale car elle permet d'optimiser la performance du solveur en réduisant le périmètre des ressources à considérer. Nous avons organisé cette section en trois onglets correspondant aux trois types principaux de ressources :

1. **Cours** : Affiche la liste complète des cours disponibles pour le périmètre sélectionné. Chaque cours est présenté avec son identifiant unique, son intitulé extrait du label, et le nombre de parties qui le composent. Une interface de tableau interactif permet d'activer ou désactiver individuellement chaque cours.

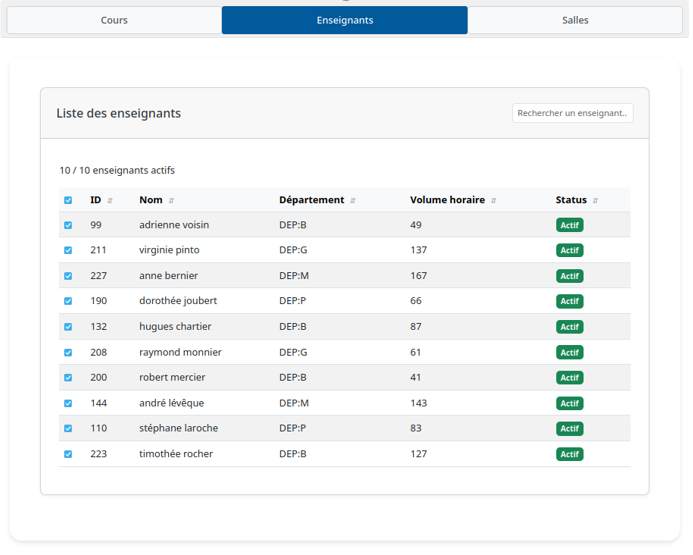


*Figure 8 : Interface de gestion des cours*

**2. Enseignants** : Présente tous les enseignants disponibles avec leurs informations détaillées, extractées intelligemment à partir des labels complexes fournis par l'API :

* Nom et prénom de l'enseignant
* Département ou unité d'appartenance
* Volume horaire associé

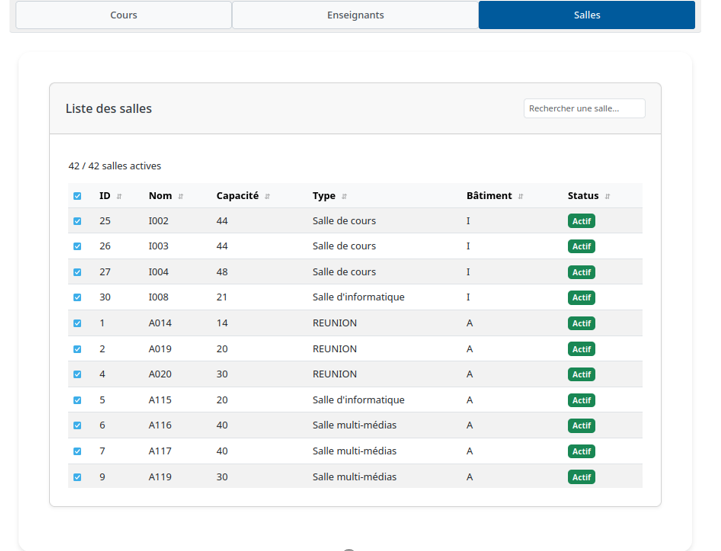
Ces informations facilitent l'identification des enseignants et permettent une sélection plus pertinente.



***Figure 9 :*** *Interface de gestion des* ***enseignants***

**3. Salles** : Liste l'ensemble des salles disponibles avec leurs caractéristiques principales :

* Identifiant et nom de la salle
* Capacité en nombre d'étudiants
* Type de salle (amphithéâtre, salle de TD, laboratoire, etc.)
* Localisation/bâtiment



*Figure 10 : Interface de gestion des salles*

Pour chacune de ces catégories, nous avons implémenté un ensemble cohérent de fonctionnalités :

**Tableaux interactifs** : Chaque ressource est présentée dans un tableau avec une structure claire et uniforme. Les tableaux offrent :

* Des cases à cocher individuelles pour activer/désactiver chaque ressource
* Un en-tête avec case à cocher générale pour tout sélectionner/désélectionner
* Un système de tri pour chaque colonne (ascendant/descendant) avec indicateur visuel
* Un badge coloré indiquant l'état actif (vert) ou inactif (rouge) de chaque ressource
* Une mise en forme visuelle différente (opacité réduite) pour les ressources inactives

**Recherche et filtrage** : Pour faciliter la gestion d'un grand nombre de ressources, chaque onglet dispose :

* D'une barre de recherche en temps réel qui filtre instantanément les ressources affichées
* D'un compteur de ressources actives/totales (par exemple "15/20 enseignants actifs")
* D'un système de pagination pour les jeux de données volumineux

**Traitement intelligent des données** : La fonction extractInfoFromLabel permet d'extraire des informations structurées à partir des labels complexes fournis par l'API. Par exemple, pour un enseignant dont le label est "name,lastname,code\_unite ,hourly\_volume:24", notre système extrait automatiquement le nom, prénom, département et volume horaire pour les afficher dans des colonnes séparées.

**Gestion de l'état** : Les ressources sélectionnées sont suivies à l'aide de trois ensembles JavaScript (Sets) :

* selectedCourses pour les cours actifs
* selectedTeachers pour les enseignants actifs
* selectedRooms pour les salles actives

À l'inverse, les ressources désactivées sont suivies dans trois ensembles distincts :

* inactiveCourses pour les cours désactivés
* inactiveTeachers pour les enseignants désactivés
* inactiveRooms pour les salles désactivées

Cette structure de données optimisée permet une gestion efficace de l'état et une mise à jour rapide de l'interface utilisateur.

**Intégration avec la configuration du solveur** : Les sélections effectuées dans cette section sont automatiquement intégrées à la configuration du solveur via la fonction getFilteredResourcesData() qui :

* Filtre les données complètes selon les sélections utilisateur
* Génère un sous-ensemble optimisé des données pour le solveur
* Met à jour le JSON de configuration avancée si ce mode est activé

Le code JavaScript principal de cette section est contenu dans le fichier filter\_ressource.js, qui implémente une architecture modulaire avec des fonctions dédiées pour chaque type de ressource :

* renderCoursesTable(), renderTeachersTable() et renderRoomsTable() pour l'affichage initial
* setupCoursesSearch(), setupTeachersSearch() et setupRoomsSearch() pour les fonctionnalités de recherche
* setupCoursesButtons(), setupTeachersButtons() et setupRoomsButtons() pour la gestion des sélections en masse
* sortTable() pour le tri générique applicable à tous les tableaux
* updateResourceCounter() pour la mise à jour des compteurs de ressources

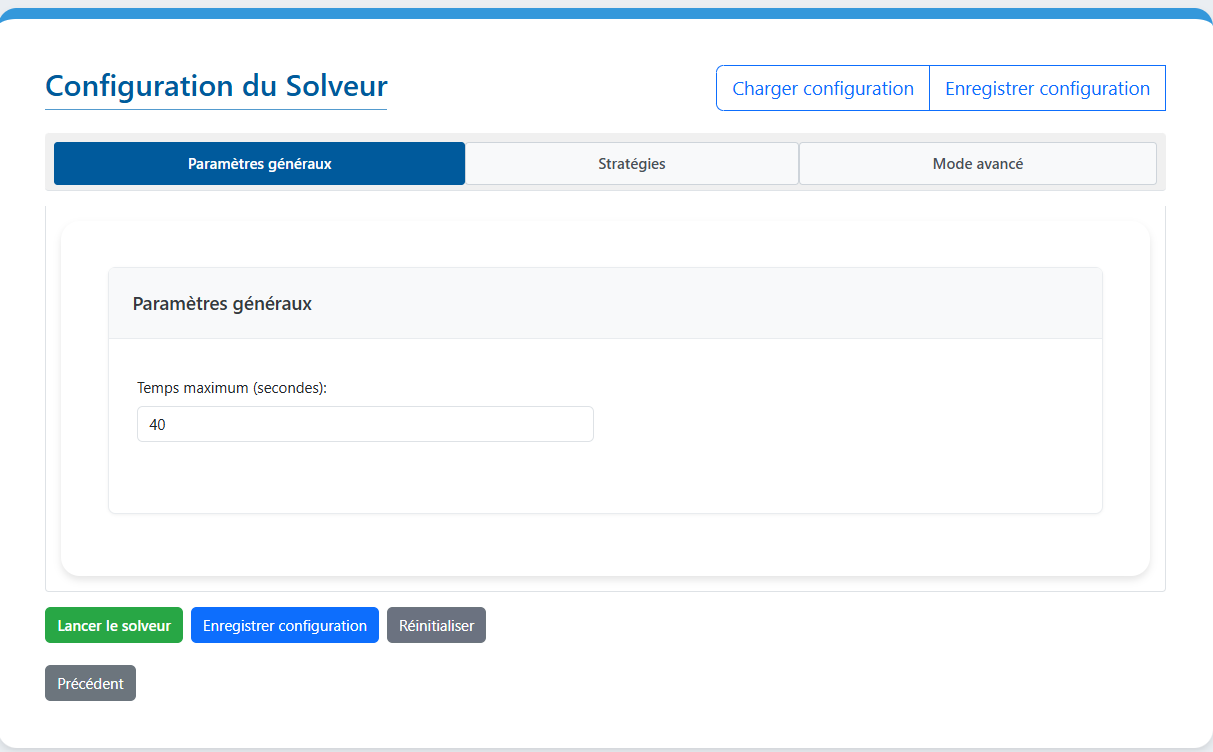
L'initialisation de cette section est synchronisée avec le chargement des données universitaires via un système d'événements personnalisés. Dès que les données sont disponibles, les tableaux sont automatiquement rendus avec toutes les ressources sélectionnées par défaut.

Cette approche de filtrage des ressources permet aux utilisateurs de réduire efficacement le périmètre de travail du solveur, ce qui améliore significativement les performances de calcul tout en assurant que seules les ressources pertinentes sont prises en compte dans la génération des emplois du temps.

#### **C****onfiguration du solveur**

La quatrième section permet de paramétrer finement le solveur selon les besoins spécifiques de l'utilisateur. Pour rendre accessible cette configuration technique, nous avons créé une interface structurée en trois sous-onglets principaux :

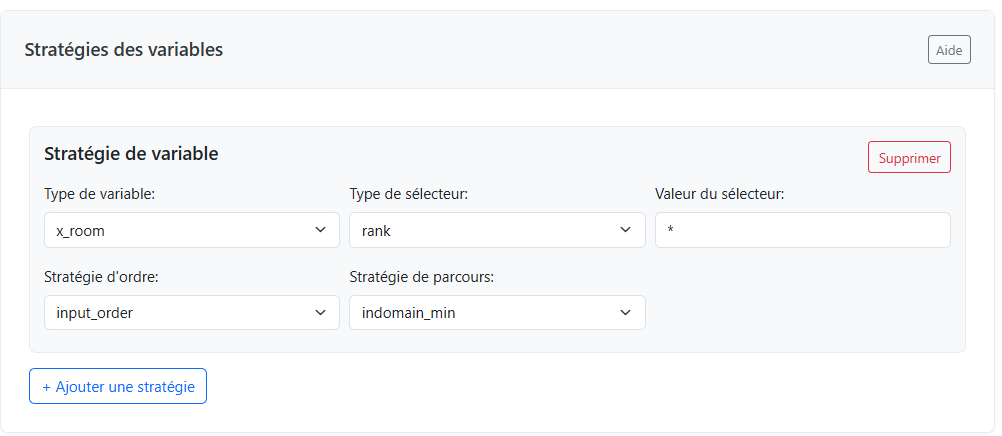
1. **Paramètres généraux** : Permet de définir les configurations de base, notamment le temps maximum d'exécution du solveur en secondes. Cette limite temporelle est cruciale pour éviter que le solveur ne s'exécute indéfiniment sur des problèmes complexes.



***Figure 11 :*** *Interface des* ***paramètre généraux***

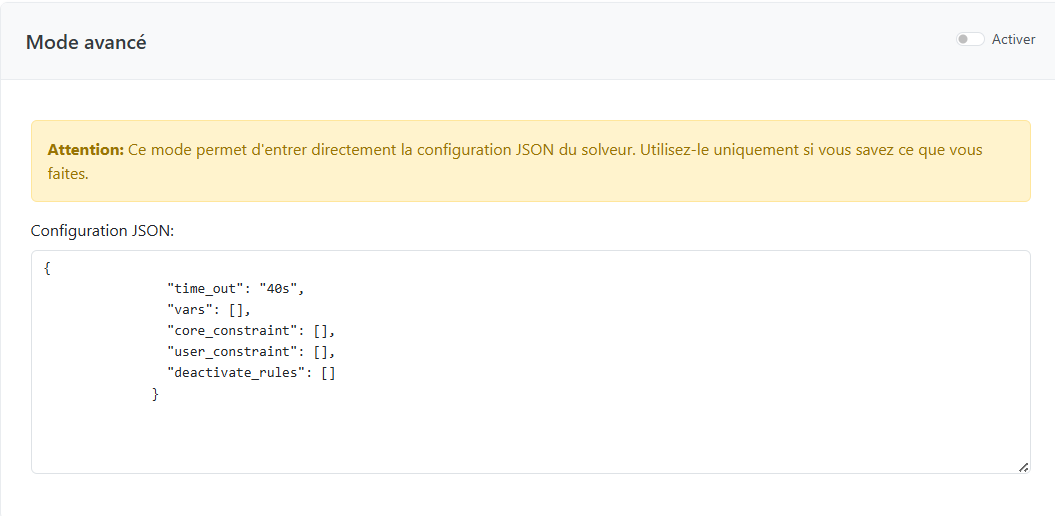
1. **Stratégies des variables** : Offre un contrôle avancé sur la manière dont le solveur traite les différentes variables du problème. Nous avons développé une interface intuitive permettant d'ajouter et configurer différentes stratégies pour chaque type de variable :
   * **Type de variable** : Sélection parmi plusieurs options comme x\_room (salles), x\_teachers (enseignants), x\_slot (créneaux), etc.
   * **Type de sélecteur** : Choix entre rank, label ou id pour définir comment cibler les variables
   * **Valeur du sélecteur** : Champ avec validation en temps réel permettant de définir les valeurs spécifiques
   * **Stratégie d'ordre** : Définit l'ordre dans lequel les variables sont traitées (first\_fail, input\_order, etc.)
   * **Stratégie de parcours** : Détermine comment les valeurs sont attribuées aux variables (indomain\_min, indomain\_random, etc.)

Chaque stratégie est représentée visuellement par une carte avec tous ces paramètres, et l'utilisateur peut ajouter autant de stratégies que nécessaire via un bouton dédié. Pour chaque stratégie ajoutée, un bouton de suppression permet de la retirer si besoin.



*Figure 12 : Interface de statégies des variables*

1. **Mode avancé** : Destiné aux utilisateurs experts, cet onglet permet d'éditer directement le JSON de configuration du solveur. Cette option offre un accès complet à toutes les possibilités de paramétrage, mais est clairement identifiée comme avancée avec un avertissement approprié. L'activation de ce mode se fait via un interrupteur qui affiche alors un éditeur JSON avec la configuration actuelle.



*Figure 13 : Interface du mode avancé*

En plus de ces onglets, nous avons implémenté des fonctionnalités de gestion des configurations :

* **Bouton "Charger configuration"** : Permet d'importer un fichier JSON de configuration existant
* **Bouton "Enregistrer configuration"** : Exporte la configuration actuelle dans un fichier JSON, avec nom automatique incluant la date et l'heure
* **Validation des entrées** : Les champs sensibles comme les valeurs de sélecteur sont validés en temps réel avec retour visuel immédiat en cas d'erreur

Le bouton "Lancer le solveur" est particulièrement important : il déclenche la génération de la configuration finale au format JSON, l'envoie à l'API du solveur, et redirige automatiquement l'utilisateur vers la section des résultats où il pourra suivre l'avancement de la résolution.

Le code JavaScript associé à cette section est principalement contenu dans les fichiers app.js et config\_Tabs.js. Les fonctions clés incluent :

* generateConfigJSON() pour créer le JSON de configuration à partir des paramètres de l'interface
* getVarsStrategies() pour extraire les stratégies de variables définies par l'utilisateur
* addNewStrategy() pour ajouter dynamiquement une nouvelle stratégie à l'interface
* loadConfigIntoForm(config) pour remplir l'interface à partir d'un JSON de configuration existant

Cette approche modulaire de la configuration permet de répondre aux besoins de différents profils d'utilisateurs, des moins techniques qui se contenteront des paramètres généraux, aux plus avancés qui pourront définir des stratégies personnalisées ou même éditer directement le JSON.

#### **Section des solutions**

La cinquième section présente les résultats générés par le solveur de manière structurée et informative. Elle est organisée pour offrir à la fois une vision synthétique des résultats et des métriques détaillées sur les performances du solveur.

La partie supérieure affiche le statut actuel du solveur (inactif, en cours, terminé, échec) accompagné du temps écoulé et d'une barre de progression visuelle. Un bouton "Arrêter le solveur" permet à l'utilisateur d'interrompre le processus de résolution à tout moment si nécessaire, offrant ainsi un contrôle total sur l'exécution. Ces indicateurs permettent à l'utilisateur de suivre en temps réel l'avancement du processus de résolution.

Une fois le calcul terminé, un résumé statistique complet s'affiche, structuré en deux colonnes principales :

1. **Résultats du planning** : Cette colonne présente des indicateurs clés sous forme de cartes visuelles colorées, incluant :

* Nombre de séances planifiées
* Nombre de groupes constitués
* Nombre de classes
* Nombre d'enseignants impliqués
* Nombre de salles utilisées

1. **Performance du solveur** : Cette colonne présente des métriques techniques détaillées pour comprendre l'efficacité du processus de résolution :

* Temps d'exécution total, décomposé en temps de création et de résolution
* Nombre de solutions trouvées
* Ratio d'efficacité (nœuds/décisions)
* Nombre d'échecs
* Métriques avancées comme le nombre de nœuds, de décisions, de contraintes et de variables

L'interface propose ensuite plusieurs options pour exploiter la solution :

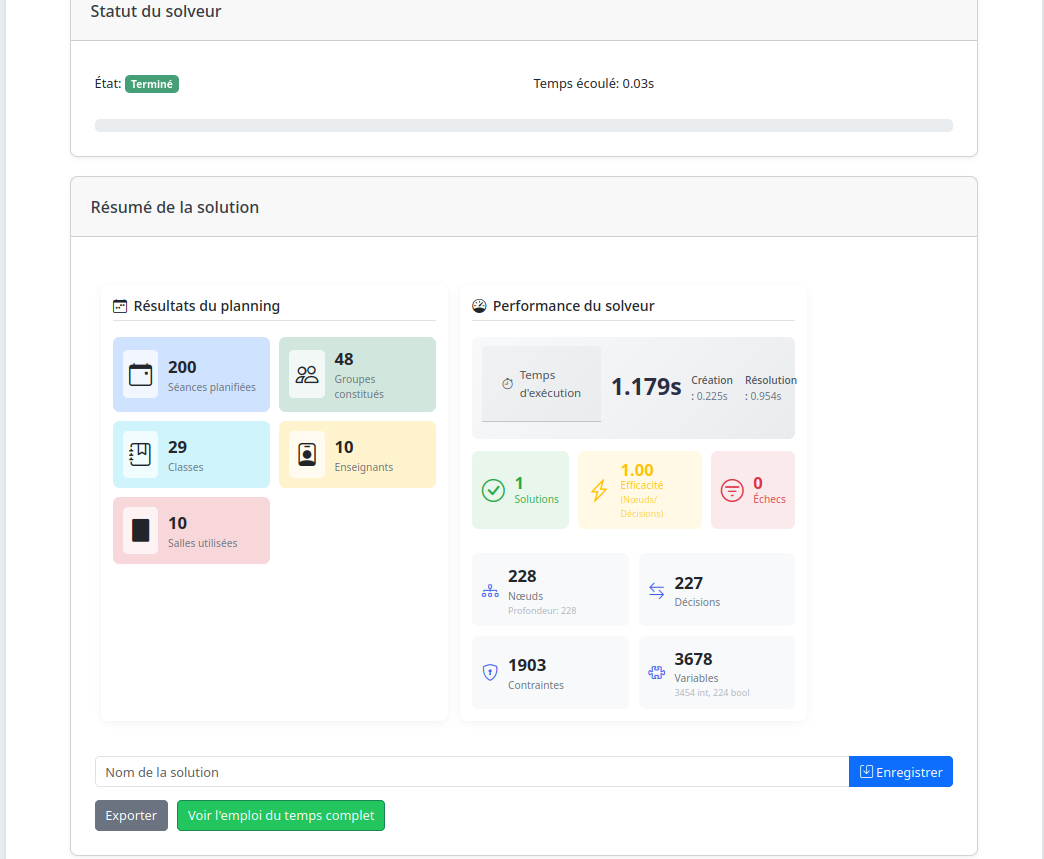
* Un champ de saisie pour nommer la solution avant de l'enregistrer
* Un bouton "Enregistrer" qui sauvegarde la solution sous le nom spécifié dans le système
* Un bouton "Exporter" qui génère un fichier JSON contenant les détails complets de la solution
* Un lien "Voir l'emploi du temps complet" qui redirige vers une application dédiée de visualisation détaillée

Un bouton "Nouvelle résolution" permet de revenir à la première étape pour réaliser une nouvelle génération d'emploi du temps.

Le fichier resultat.js contient les fonctions principales associées à cette section :

* attendreEtat(etatVise) pour surveiller l'état du solveur via des requêtes API
* chargerResultatSolveur() pour récupérer et afficher la solution
* afficherStatsSolution(solution, statistics) pour générer les statistiques détaillées
* exporterSolutionJSON(data) pour l'export des données

Cette présentation des résultats, à la fois visuelle et informative, permet aux utilisateurs d'évaluer rapidement la qualité de la solution générée et d'accéder aux détails techniques s'ils le souhaitent.



***Figure 14 :*** *Interface de visualisation des résultats*

## **Interaction avec les API :**

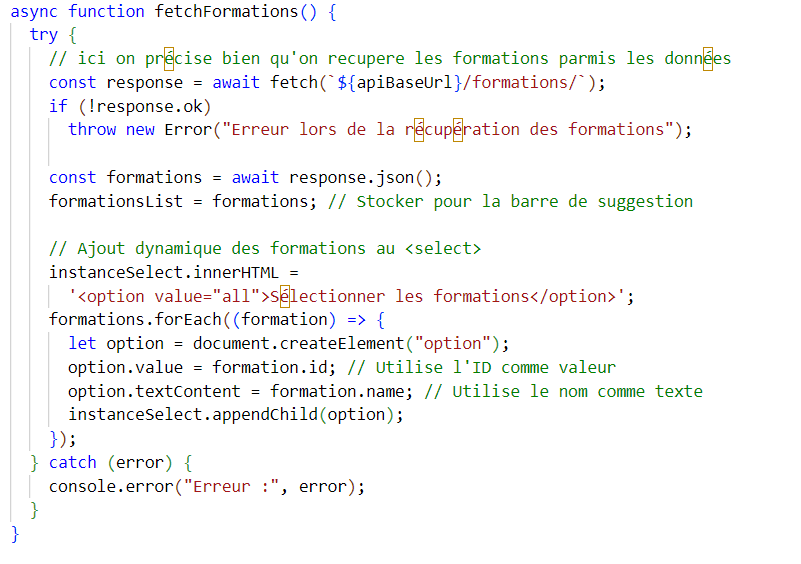
Notre application frontend interagit avec deux API REST distinctes pour fonctionner : l'API de données et l'API du solveur. Cette architecture permet une séparation claire des responsabilités et facilite la maintenance future.

### **1. Communication avec l'API de données**

L'API de données fournit toutes les informations nécessaires pour constituer le périmètre de travail. Notre application y accède via plusieurs endpoints :

* /api/formations/ : Récupère la liste complète des formations disponibles
* /api/steps/formation/{formationId} : Obtient les périodes (ou semestres) associées à une formation spécifique
* /api/perimeters/?steps={stepIds} : Récupère toutes les données (règles, cours, enseignants, salles) pour un ensemble de périodes sélectionnées

La communication avec cette API s'effectue de manière asynchrone à l'aide de la fonction fetch() et des promesses JavaScript. Par exemple, pour récupérer les formations :



*Figure 15 : Code de la fonction fetchFormations ()*

Les données récupérées sont stockées dans la variable globale universityData qui est ensuite utilisée par les différentes sections de l'application.

### **2. Communication avec l'API du solveur**

L'API du solveur gère la création d'instances, l'exécution des calculs et la récupération des résultats. Notre application interagit avec elle via plusieurs endpoints clés :

* /solver/new (POST) : Crée une nouvelle instance du solveur avec les données et la stratégie fournies
* /solver/{id}/status (GET) : Vérifie l'état actuel du solveur (READY, RUNNING, FINISHED, FAILED)
* /solver/{id}/start (POST) : Démarre le processus de résolution
* /solver/{id}/result (GET) : Récupère les résultats de la solution générée
* /solver/{id}/rename (POST) : Renomme le champ « name » d’une solution
* /solver/all (GET) : Récupère la liste des configurations sauvegardées
* /solver/{id}/instance et /solver/{id}/strategy (GET) : Récupère les détails d'une configuration existante

La communication avec cette API nécessite une gestion plus complexe, notamment pour le suivi de l'état du solveur. Nous avons implémenté un système de polling qui vérifie régulièrement l'état jusqu'à ce que le processus soit terminé :



***Figure 16 :*** *Code de la* ***fonction attendreEtat***

### **3. Gestion des données et transformation**

Un aspect important de notre application est la transformation des données entre les différents formats :

* **JSON → Interface utilisateur** : Les données JSON récupérées des API sont rendues dans l'interface sous forme de tableaux, listes et formulaires.
* **Interface utilisateur → JSON** : Les sélections et configurations de l'utilisateur sont converties en JSON structuré pour les requêtes API.
* **JSON → XML** : Pour certaines opérations, nous générons dynamiquement du XML à partir des données JSON, notamment pour la visualisation du périmètre sélectionné.

Ces transformations sont gérées par des fonctions dédiées comme generateConfigJSON() pour la configuration du solveur et generateCompleteTimetabling() pour la génération du XML.

### **4. Gestion des erreurs et feedback utilisateur**

Nous avons implémenté une gestion robuste des erreurs pour toutes les communications avec les API :

* Vérification systématique des codes de statut HTTP
* Messages d'erreur explicites pour l'utilisateur
* Logging détaillé des erreurs dans la console pour le débogage
* Affichage de feedback visuel (badges, alertes) pour informer l'utilisateur de l'état des opérations

Cette approche d'interaction avec les API permet à notre application frontend de fonctionner de manière fiable tout en offrant une expérience utilisateur fluide, sans exposer la complexité technique sous-jacente.

## **Fonctionnalités implémentées :**

Nous avons développé plusieurs fonctionnalités clés pour offrir une expérience utilisateur complète :

1. **Recherche et sélection avancées** :

* Autocomplétion dans la barre de recherche des formations
* Sélection multiple de périodes avec "Tout sélectionner"
* Résumé visuel des sélections avec possibilité d'édition et suppression

1. **Gestion des contraintes et règles** :

* Activation/désactivation des contraintes fondamentales et métier
* Interface de désactivation de règles avec trois modes d'accès (liste, filtrage, saisie manuelle)
* Recherche et filtrage avancé des règles avec plusieurs critères
* Compteur de règles actives/inactives

1. **Gestion des ressources** :

* Filtrage et tri des cours, enseignants et salles
* Activation/désactivation individuelle ou globale
* Recherche textuelle dans chaque catégorie
* Indication visuelle de l'état actif/inactif

1. **Configuration avancée du solveur** :

* Interface intuitive pour les stratégies de variables
* Validation des entrées utilisateur
* Export/import de configurations
* Mode avancé avec édition directe du JSON

1. **Suivi et visualisation des résultats** :

* Barre de progression
* Statistiques sur la solution générée
* Aperçu de l'emploi du temps
* Export de la solution en JSON
* Possibilité d'arrêter le solveur pendant son exécution
* Options de visualisation des données en différents formats (XML, JSON)

1. **Navigation et expérience utilisateur** :

* Interface à onglets avec navigation séquentielle (boutons Précédent/Suivant)
* Sauvegarde de l'état des onglets dans l'URL
* Animations et transitions fluides
* Responsive design adapté aux différents formats d'écran

1. **Intégration avec les API** :

* Génération dynamique du XML à partir des sélections
* Communication asynchrone avec le serveur
* Gestion des erreurs avec messages utilisateur
* Polling pour suivre l'état du solveur

## **Difficultés rencontrées et solutions apportées :**

Durant le développement de l'application, nous avons rencontré plusieurs défis techniques que nous avons surmontés :

1. **Gestion des dépendances entre sections** :

La configuration du solveur dépendait fortement des données sélectionnées dans les sections précédentes. Nous avons implémenté un système d'événements personnalisés pour notifier les différentes parties de l'application des changements d'état.

1. **Traitement de grandes quantités de données** :

Les règles et ressources pouvaient être très nombreuses, risquant de ralentir l'interface. Nous avons optimisé les performances par :

* L'utilisation de tableaux paginés
* Des mécanismes de recherche et filtrage efficaces
* Le chargement à la demande des données détaillées

1. **Complexité de la configuration du solveur** : Pour faciliter la gestion des nombreux paramètres, nous avons :

* Créé une interface graphique intuitive pour les stratégies de variables
* Ajouté des validations en temps réel des entrées utilisateur
* Intégré une aide contextuelle pour guider les utilisateurs

## **Résultats obtenus :**

Notre projet a abouti à une interface de pilotage complète et fonctionnelle pour le solveur d'emplois du temps du LERIA. Les principaux résultats obtenus sont les suivants :

### **Interface utilisateur intuitive**

Nous avons développé une interface web responsive, organisée en sections distinctes suivant un flux de travail logique. Cette interface permet aux utilisateurs non experts d'exploiter efficacement les capacités du solveur sans connaissances techniques préalables.

La page d'accueil offre un point d'entrée clair avec deux options simples : créer une nouvelle configuration ou charger une configuration existante. Cette approche facilite le démarrage rapide et la réutilisation des paramétrages précédents.

### **Fonctionnalités complètes de sélection et filtrage**

Le système permet une sélection précise des données d'entrée avec une recherche intuitive des formations et une gestion claire des périodes associées. Les utilisateurs peuvent visualiser facilement leurs sélections et les modifier si nécessaire.

Les sections de filtrage des règles et des ressources offrent différentes approches adaptées aux besoins variés des utilisateurs, de la simple activation/désactivation jusqu'au filtrage avancé utilisant de multiples critères.

### **Configuration flexible du solveur**

Notre interface propose plusieurs niveaux de configuration, des paramètres généraux simples (comme le temps maximum d'exécution) aux stratégies avancées de variables, en passant par un mode expert permettant l'édition directe du JSON de configuration.

Les utilisateurs peuvent sauvegarder et charger leurs configurations, facilitant ainsi le partage et la réutilisation des paramétrages efficaces.

### **Visualisation des résultats**

La section des solutions offre un suivi en temps réel de l'exécution du solveur, avec la possibilité d'interrompre le processus si nécessaire via un bouton "Arrêter". Les résultats sont présentés de manière structurée avec des statistiques claires, et peuvent être visualisés dans différents formats (JSON, XML). Les utilisateurs peuvent nommer, sauvegarder et exporter leurs solutions, ou visualiser l'emploi du temps complet via l'application dédiée.

### **Tests et validation**

L'application a été testée avec différents scénarios d'utilisation, validant ainsi sa robustesse et son adaptabilité. Les principales fonctionnalités fonctionnent comme prévu, permettant un pilotage complet du solveur de la sélection des données jusqu'à l'exploitation des résultats.

### **Documentation**

Une documentation complète du code a été réalisée, incluant des commentaires explicites et une organisation modulaire facilitant les futures évolutions. Le présent rapport constitue également une ressource précieuse pour comprendre l'architecture et le fonctionnement de l'application.

Ces résultats correspondent aux objectifs initiaux du projet et offrent une solution complète pour faciliter l'utilisation du solveur d'emplois du temps par les responsables de formation et le personnel administratif de l'université.

**VI. Conclusion :**

Ce projet de développement d'une interface de pilotage pour le solveur d'emplois du temps du LERIA a été une expérience enrichissante qui nous a permis de mettre en application nos connaissances théoriques dans un contexte professionnel concret.

### **Bilan du projet**

Nous avons réussi à développer une application web complète qui répond pleinement aux objectifs initiaux :

* **Accessibilité accrue** : L'interface rend le solveur accessible aux utilisateurs non-techniques comme les responsables de formation et le personnel administratif.
* **Workflow intuitif** : L'organisation en sections séquentielles guide naturellement l'utilisateur à travers le processus de génération d'emplois du temps.
* **Flexibilité** : L'application offre différents niveaux de configuration adaptés à tous les profils d'utilisateurs, des plus novices aux plus expérimentés.
* **Exploitation des résultats** : Les solutions générées sont présentées de manière claire et peuvent être facilement sauvegardées, exportées ou visualisées.

Sur le plan technique, nous avons relevé plusieurs défis importants :

* La manipulation de structures de données complexes et imbriquées
* La communication asynchrone avec plusieurs API
* La création d'interfaces utilisateur interactives et réactives
* L'optimisation des performances pour gérer de grandes quantités de données

### **Compétences développées**

Ce projet nous a permis de développer diverses compétences techniques et transversales :

* **Développement frontend** : Maîtrise avancée de HTML5, CSS3, JavaScript et Bootstrap
* **Conception d'interface** : Principes d'UX/UI pour créer une expérience utilisateur intuitive
* **Communication API** : Techniques d'interaction avec des API REST
* **Gestion de projet** : Organisation du travail en équipe, partage des tâches, suivi de l'avancement
* **Documentation** : Rédaction de documentation technique claire et complète

### **Perspectives d'évolution**

L'application développée offre plusieurs axes d'amélioration pour des développements futurs :

* **Visualisation avancée** : Intégration d'une interface de visualisation graphique des emplois du temps directement dans l'application
* **Comparaison de solutions** : Fonctionnalités permettant de comparer plusieurs solutions générées selon différents critères
* **Personnalisation accrue** : Système de profils utilisateurs pour sauvegarder les préférences et configurations favorites
* **Analytics** : Tableaux de bord avec statistiques sur les performances des différentes configurations du solveur
* **Intégration système** : Connexion avec les systèmes d'information existants de l'université

### **Mot de la fin**

Ce stage nous a offert une opportunité unique de contribuer à l'amélioration d'un outil essentiel pour l'organisation pédagogique de l'université. La collaboration étroite avec l'équipe du LERIA a été particulièrement formatrice, nous permettant de bénéficier de leur expertise dans le domaine de la recherche opérationnelle et de la programmation par contraintes.

Nous sommes convaincus que l'interface développée facilitera grandement le travail des responsables de formation et du personnel administratif, leur permettant de générer des emplois du temps optimisés tout en réduisant significativement la complexité technique de cette tâche.

Cette expérience constitue une étape importante dans notre parcours académique et professionnel, nous ayant permis de mettre en pratique nos connaissances théoriques dans un contexte réel et de développer des compétences recherchées sur le marché du travail.

**ANNEXES**

**Ressources techniques**

|  |  |
| --- | --- |
| **Référence** | **Lieu ressource** |
| Documentation officielle Bootstrap | <https://getbootstrap.com/docs/5.3/> |
| Bootstrap 5 CSS | <https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css> |
| Bootstrap 5 JS | <https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js> |
| Bootstrap Icons | <https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.0/font/bootstrap-icons.css> |

**Ressources pédagogiques**

|  |  |  |
| --- | --- | --- |
| **Titre** | **Auteur** | **Lien** |
| Livret CM JavaScript/PHP (L3) | David Lesaint | <https://moodle.univ-angers.fr/pluginfile.php/2923190/mod_resource/content/48/l3-info-dw-livret-cm.pdf> |
| Sources des exemples JavaScript | David Lesaint | <https://moodle.univ-angers.fr/mod/resource/view.php?id=580916&redirect=1> |
| Programme de L2 : CM JS/PHP | David Lesaint | <https://moodle.univ-angers.fr/pluginfile.php/2028117/mod_resource/content/32/l2mi-dw-livret-cm.pdf> |