Pro’Gramme

▼ Sommaire

▼ Contexte du projet

Pro'Gramme est une application web dédiée aux boulangeries indépendantes, conçue pour optimiser leur organisation et faciliter le développement de leur activité.

Elle permet de :

- Visualiser l'activité de la boulangerie via un dashboard intuitif
- Gérer un catalogue de recettes personnalisé
- Calculer automatiquement les quantités d'ingrédients nécessaires
- Suivre et gérer les stocks en temps réel
- Planifier efficacement les préparations quotidiennes

L'objectif principal est d'améliorer la productivité des artisans boulangers en leur offrant un outil professionnel simple et performant pour gérer leur production au quotidien.

▼ Notre équipe

Notre équipe est composée de 5 développeurs, chacun étant responsable de fonctionnalités spécifiques :

Nom	Rôle	Missions
MATHYS Hemmy-Lola	***Tech Lead*** /Dev Front-End	Coordination technique, gestion Git, intégration globale, Améliorations UX/UI, contrastes, audit *Lighthouse*, corrections liées à l'accessibilité
KALLOGA Sira	Dev Front-End – ***Stock***	Gestion du stock, interaction avec le localStorage, affichage dynamique des ingrédients
LAWSON Abraham	Dev Front-End – ***Recettes***	Affichage de recettes, ajout/suppression, intégration formulaire, structure HTML & JS
MASSON Léo	Dev Front-End – ***Calculateur***	Développement de la logique de calcul des quantités en JS, adaptation des recettes selon le nombre de personnes, vérification du stock
STANKOVIC Luka	Dev Front-End – ***Planning***	Interface de planification des repas, CRUD de planification, affichage des recettes du jour

→ Chaque membre de l'équipe a veillé à respecter les bonnes pratiques d'accessibilité (contrastes, navigation clavier, rôles ARIA, structure HTML sémantique) dans le développement de ses fonctionnalités, afin de garantir une expérience inclusive pour tous les utilisateurs.

▼ Structure du projet

Notre projet est structuré en dossiers thématiques pour maintenir un code clair, maintenable et évolutif. Voici l'organisation détaillée des fichiers et dossiers principaux :

```
├── index.html          # Page d'accueil / Dashboard principal
├── README.md           # Documentation du projet (installation, usage)
│
├── css/
│   ├── style.css       # Style principale du site
```

```
│   └── planning.css        # Dédiée à la page Planning
│
├── data/
│   ├── recipes.json        # Recettes utilisées dans le calculateur et la liste
│   ├── stock.json          # État du stock pour la gestion des recettes
│   ├── users.json          # Informations sur les utilisateurs (simulées)
│   └── planning.json       # (Non utilisé - prévu pour la persistance du pla
│
├── html/
│   ├── add-recipe.html     # Formulaire d'ajout de recette
│   ├── calculator.html     # Outil de calcul de quantités
│   ├── my-recipes.html     # Liste des recettes existantes
│   ├── planning.html       # Visualisation du planning des recettes
│   ├── stock.html          # Page de gestion du stock
│   └── users.html          # Affichage des utilisateurs
│
├── js/
│   ├── add-recipe.js       # Logique JS pour l'ajout de recettes
│   ├── calculator.js       # Adaptation des quantités selon le nb de person
│   ├── dashboard.js        # Comportement du tableau de bord (index.htm
│   ├── my-recipes.js       # Affichage dynamique des recettes
│   ├── nav.js              # Gestion du menu responsive (burger)
│   ├── planning.js         # Gestion du planning quotidien
│   └── stock.js            # Stock centralisé utilisé dans plusieurs vues
```

▼ *Notes*

→ `planning.css` *a été séparé de style.css en raison de contraintes de fusion de dernière minute liées au travail en équipe.*

→ `planning.json` *a été laissé vide et inutilisé, car la logique de sauvegarde du planning n'a pas été implémentée dans cette version du projet.*

→ *Tous les fichiers non utilisés ou secondaires sont mentionnés ici pour* ***assurer la transparence et la traçabilité*** *du développement.*

▼ Organisation du développement

Pour garantir une bonne coordination et un suivi fluide du projet, nous avons adopté une organisation inspirée des méthodes agiles, avec les éléments suivants :

▼ Gestion de projet

- **Tableau Kanban sur Trello** :

  Nous avons utilisé un tableau pour :

  - Lister les tâches à faire (*Backlog*)
  - Suivre les tâches en cours (*In Progress*)
  - Valider les tâches terminées (*Done*)

- **Réunions régulières informelles** : Points fréquents entre les membres pour échanger sur l'avancement, les blocages et la répartition des tâches.

▼ Gestion du code

Une branche Git distincte a été attribuée à chaque développeur pour le développement de sa fonctionnalité spécifique :

 → `main` : Branche principale contenant la version finale de notre application

→ `dev` : Branche destinée à la fusion du code

→ `feature/ux` : Amélioration de l'expérience utilisateur (UX) de l'application

→ `feature/calculator` : Calculateur des quantités pour les recettes

→ `feature/dashboard` : Visualisation et gestion du tableau de bord

→ `feature/planning` : Planification des activités quotidiennes

→ `feature/recipes` : Gestion du catalogue de recettes

→ `feature/stocks` : Gestion du catalogue de stock

→ `debug-version` : Version opérationnelle du projet

▼ Technologies et outils utilisés

▼ Langages

→ `HTML5`,

→ `CSS3`,

→ `JavaScript (ES6)`

▼ Environnement de développement

→ Visual Studio Code,

→ Git & GitHub (GitHub Pages),

→ Git flow

▼ Outils Design et UI

→ Figma : *Design System*

> https://www.figma.com/design/no1UXOI5bNiTIwzdakEn5s/Design-System---Pro-Gramme?node-id=1-108&t=5hr6zOFCQOP14QRc-1

→ Coolors : *Choix des couleurs de l'application*

→ Accessible Colors : *Vérification de l'accessibilité*

▼ Optimisation & Accessibilité

→ **Lighthouse** (Google Chrome) : *audit de performance, accessibilité et bonnes pratiques*

→ **Minify CSS & JS** : *réduction du poids des fichiers pour un chargement plus rapide*

→ **Balises ARIA**, **tabindex**, **structure HTML sémantique** : *pour une meilleure prise en charge par les lecteurs d'écran et une navigation clavier fluide*

▼ Fonctionnalités développées

▼ Dashboard de suivi

Branche associée → `feature/dashboard`

Fichiers concernés → `index.html`, `dashboard.js`

Description :

Cette fonctionnalité centralise les entrées clés du site : elle sert de point de départ pour suivre les recettes planifiées, les favoris et l'état des stocks. Elle permet d'avoir une vue d'ensemble en un coup d'œil, facilitant ainsi l'organisation quotidienne.

Fonctions clés :

- Accès direct au planning des recettes à venir.
- Accès aux recettes préférées enregistrées.
- Affichage synthétique de l'état des stocks (alerte en cas de seuil critique).
- Interface épurée et accessible, avec navigation clavier et hiérarchie visuelle claire.
- Lien avec les autres pages du site pour fluidifier l'expérience utilisateur.

▼ Gestion des stocks

Branche associée → `feature/stocks`

Fichiers concernés → `stock.html`, `stock.js`, `stock.json`

Description :

Cette fonctionnalité permet de gérer dynamiquement les ingrédients disponibles dans le stock, avec un affichage en temps réel, l'ajout manuel de nouveaux ingrédients, et une déduction automatique en lien avec le calculateur de recettes.

Fonctions clés :

- Affichage automatique des stocks dans un tableau accessible.
- Ajout manuel d'un ingrédient (nom, quantité, unité).
- Déduction automatique des quantités depuis le calculateur.
- Alerte "*stock à recharger*" si la quantité passe sous un seuil défini.
- Accessibilité : navigation clavier, `aria-live`, `role="alert"`, contraste renforcé.
- Lien dynamique entre les modules `calculator.html` et `stock.html`.

▼ Recettes

Branche associée → `feature/recipes`

Fichiers concernés → `my-recipes.html`, `my-recipes.js`, `add-recipe.html`, `add-recipe.js`, `recipes.json`

Description :

Cette fonctionnalité permet d'afficher l'ensemble des recettes disponibles et d'en créer de nouvelles via un formulaire. Elle constitue la base de données utilisée par les autres modules (calculateur, planning).

Fonctions clés :

- Liste dynamique des recettes à partir du fichier JSON.
- Formulaire accessible pour ajouter une recette (nom, ingrédients, quantités, unités, portions).
- Structuration logique et sémantique du contenu (balises claires, champs labellisés).
- Intégration avec les modules de calcul et de planning.
- Respect des normes d'accessibilité : contraste, navigation au clavier, retour utilisateur.

▼ Calculateur de recettes

Branche associée → `feature/calculator`

Fichiers concernés → `calculator.html`, `calculator.js`, `recipes.json`

Description :

Le calculateur ajuste automatiquement les quantités d'ingrédients en fonction du nombre de personnes saisies pour une recette. Il permet une adaptation rapide et pédagogique des recettes.

Fonctions clés :

- Chargement dynamique des données de recettes via `recipes.json`.
- Affichage des quantités standard puis recalcule selon l'entrée utilisateur.
- Gestion intelligente des quantités fractionnaires (œufs, unités non divisibles).
- Affichage d'un message pédagogique en cas d'arrondis.
- Accessibilité renforcée : navigation clavier, feedback vocal via `aria-live`.

- Intégration avec le gestionnaire de stock pour mise à jour des ingrédients.

▼ Planning

Branche associée → `feature/planning`

Fichiers concernés → `planning.html`, `planning.js`, `planning.json`

Description :

Le module Planning permet d'organiser les repas sur un calendrier, en assignant des recettes aux jours de la semaine. Il facilite la préparation et la gestion des stocks.

Fonctions clés :

- Interface calendrier pour planifier les recettes.
- Sélection de recettes existantes et affectation à une date.
- Sauvegarde locale des recettes prévues (simulation via fichier JSON).
- Interface accessible : navigation fluide au clavier, mise en valeur des focus.
- Lien logique avec les modules "recettes" et "calculateur".

▼ Accessibilité & Performances

L'interface respecte les meilleures pratiques pour garantir une expérience accessible et performante:

▼ Typographie et couleurs

- **Police** : **Inter**, une police claire et optimisée pour l'écran.
- **Contrastes** : Couleurs validées conformes au niveau **AAA** des WCAG 2.1, assurant une bonne lisibilité même en cas de daltonisme ou faible luminosité.
-

▼ Accessibilité fonctionnelle

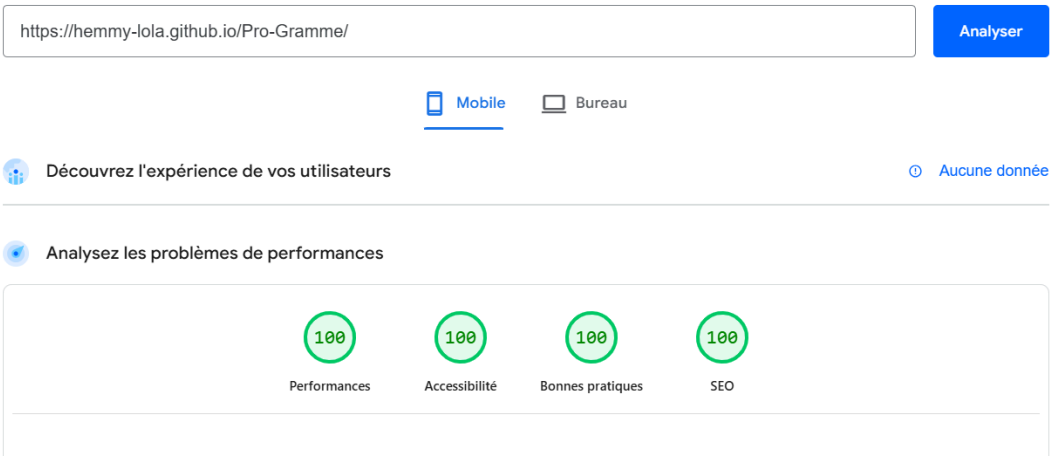
- **Navigation clavier complète** : Tous les éléments interactifs (liens, boutons, formulaires, menus) sont accessibles au clavier via Tab / Shift+Tab, activables avec Entrée ou Espace.

- **Rôles ARIA appropriés** :
  - `aria-label` pour éléments sans texte visible,
  - `aria-required` et `aria-invalid` sur les champs de formulaire,
  - `role="alert"` et `aria-live="polite"` pour messages dynamiques et résultats de recherche.
- **Labels explicites** associés aux champs via `for` / `id`.
- **Zones dynamiques** avec `aria-live="polite"` pour les mises à jour sans rechargement.

▼ Tests et audits

- Tests de navigation clavier (tabulation complète, focus visible)
- Tests lecteurs d'écran (NVDA sous Windows et VoiceOver sur Mac) : bonne annonce des titres, champs, erreurs, et retours dynamiques.

▼ Performances



Performance de notre page index via PageSpeed Insights

▼ Déploiement

Le projet est déployé depuis la branche principale (`main`), après fusion des modifications validées sur la branche de développement (`dev`). Le fichier `index.html` à la racine sert de point d'entrée.

Principale difficulté rencontrée : Configurer correctement les chemins d'accès aux ressources, qui différaient entre l'environnement local et le serveur de production.
→ Après ajustement, nous avons pu garantir un déploiement stable et fonctionnel.

▼ Améliorations futures

→ Étendre l’accessibilité pour toucher un public plus large, en testant davantage de dispositifs et technologies d’assistance.

→ Minimiser et optimiser le code pour améliorer la performance, réduire les temps de chargement et rendre l’app plus fluide sur tous les appareils.

→ Intégrer un backend pour assurer la gestion sécurisée et permanente des données, permettre la sauvegarde des recettes/plannings et offrir une expérience utilisateur personnalisée.