
## Equipe

BEN AISSA Nadim
BONIFAY Tobias
GAZZEH Sourour
SCHALAKWIJK MATHIEU

# *PLAN DU RAPPORT*

# Introduction

Dans le cadre de notre projet, qui s'inscrit dans une approche globale basée sur une architecture en microservices pour un système de restaurant, notre démarche vise à atteindre les objectifs suivants :

- Expérimenter une seconde architecture orientée client (les IHM utilisent les microservices)
- Expérimenter une architecture BFF.
- Effectuer une comparaison entre ces deux approches afin d'évaluer leurs avantages respectifs.
- Examiner l'évolution des besoins et les choix stratégiques qui ont été faits.
- Prendre du recul pour une vision globale de l'avancement du projet.

# Besoins initiaux

Ce projet vise à mettre en place un système de prise de commande sur une borne située à l'entrée du restaurant, permettant ainsi aux clients de passer leur commande de manière efficace et intuitive.

## I. Analyse des besoins

Au cours avons identifié trois personas représentant des utilisateurs potentiels, chacun caractérisé par un profil unique et des besoins spécifiques par rapport à notre application :

### Persona 1 : Sophie

- Âge : 35 ans
- Style de vie : Sophie mène une vie active entre son travail et ses engagements associatifs. Elle apprécie les moments de détente à la maison.
- Motivation : Sophie cherche des options de repas à emporter pour savourer des moments de tranquillité à la maison sans avoir à cuisiner elle-même. Elle attache également de l'importance à pouvoir suivre avec précision l'état d'avancement de sa commande.

### Persona 2 : Pierre

- Âge : 45 ans
- Style de vie : Pierre a un style de vie occupé et travaille dans un bureau. Il apprécie les moments de calme et de détente, loin du travail pendant sa pause déjeuner.
- Motivation : Pierre aime prendre une pause déjeuner agréable et tranquille dans le restaurant. Il recherche une interface claire pour choisir l'option "manger sur place".

### Persona 3 : Amina

- Âge :  30 ans
- Style de vie : Amina mène un style de vie conscient de sa santé, axé sur le bien-être et la pratique du yoga. Elle privilégie les repas végétariens ou à base de plantes.
- Motivation : Amina est consciente de sa santé et préfère des plats végétariens ou à base de plantes. Elle aime avoir des informations détaillées sur les ingrédients pour faire des choix alimentaires éclairés. Elle choisit souvent l'option à emporter pour pouvoir méditer chez elle tout en dégustant son repas.

### Persona 4 : Anna

- Âge :  20 ans
- Style de vie : Anna est une étudiante en arts, créative et ouverte d'esprit. Elle aime explorer de nouvelles expériences et apprécie la diversité des plats proposés.
- Motivation : Anna aime essayer de nouveaux plats à chaque visite au restaurant. Elle a du mal à se décider et change souvent d'avis. Elle a besoin d'une option d'annulation et pourrait opter pour emporter si elle n'est pas sûre.

Suite à notre analyse des besoins du client, les fonctionnalitées identifiées sont les suivantes :

- **Consultation du menu** : Le système doit permettre aux clients de consulter le menu proposé par le restaurant. Ils devront avoir la possibilité de parcourir les différentes catégories et sous-catégories, ainsi que les plats disponibles, les tarifs associés, et les détails des ingrédients
- **Option de commande à emporter ou sur place** : Les clients doivent avoir le choix entre apporter leur commande ou manger sur place. Cette option doit être clairement indiquée et facilement sélectionnable.
- **Annulation de la commande** : Un mécanisme d'annulation de la commande doit être mis en place. Les clients doivent pouvoir revenir en arrière et annuler leur commande si nécessaire.
- **Suivi de la commande (Pour emporter ou sur place)** : Pour les commandes, les clients doivent être en mesure de suivre l'état d'avancement de leur commande. Cela inclut le temps de préparation de la commande, ainsi que l'identification unique de leur commande pour faciliter son retrait.

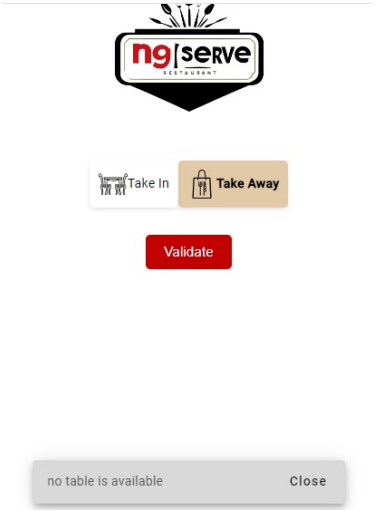
## II. Décisions stratégiques

### IHM :

Nous avons conçu une maquette répondant précisément aux besoins identifiés pour chaque persona. De plus, nous avons pris en considération les spécificités liées au

support d'affichage, à savoir une borne de commande. Cette approche de conception vise à garantir une interface adaptée et intuitive pour nos utilisateurs potentiels, tout en assurant une expérience fluide et efficace, qu'il s'agisse de Sophie, Pierre, Amina ou Anna.

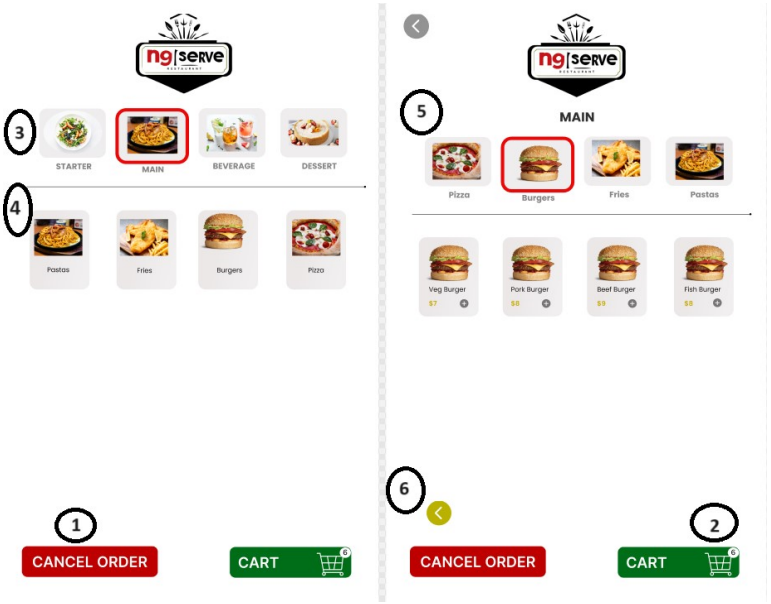
Le client a la possibilité de sélectionner entre les options "Take in" et "Take away", puis il doit appuyer sur le bouton "Validate" pour confirmer son choix. Si toutes les tables sont réservées, une page s'affiche pour informer le client de cette situation. Dans cette situation, le client aura seulement le choix à manger sur place :



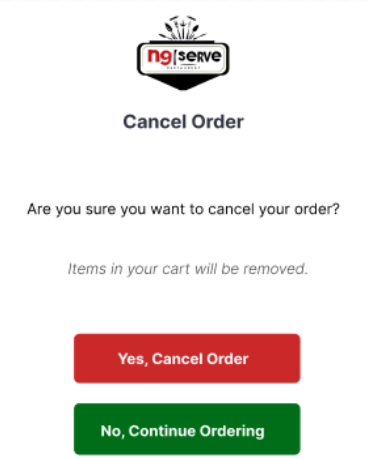




Pour la gestion du menu la figure ci-dessous montre la maquette réalisée.
Les catégories sont visibles en haut de la page (3). En cliquant sur l'une d'entre elles par exemple sur "Main", les sous-catégories associées à "Main" (pâtes, pizzas, burgers) sont révélées.
Lorsqu'on clique sur l'une de ces sous-catégories, la liste des plats correspondants s'affiche sur une autre page (5). De plus, si ces plats sont répartis sur plusieurs pages, un bouton flèche (6) permet de faire défiler horizontalement pour accéder à l'ensemble de la sélection.
Cette implémentation permet d'offrir une navigation intuitive et organisée pour les utilisateurs. En disposant les catégories en haut de la page et en affichant les sous-catégories au clic, on favorise une exploration progressive et évite la surcharge d'informations. En redirigeant les utilisateurs vers des pages dédiées lorsqu'ils sélectionnent une sous-catégorie, on guide efficacement leur recherche et facilite l'accès au contenu spécifique. Enfin, la fonctionnalité de pagination horizontale offre une solution efficace pour parcourir une liste étendue de plats sans avoir à revenir en arrière. Cette approche vise ainsi à améliorer l'expérience utilisateur, à favoriser une recherche efficace et à garantir une organisation optimale des éléments proposés.

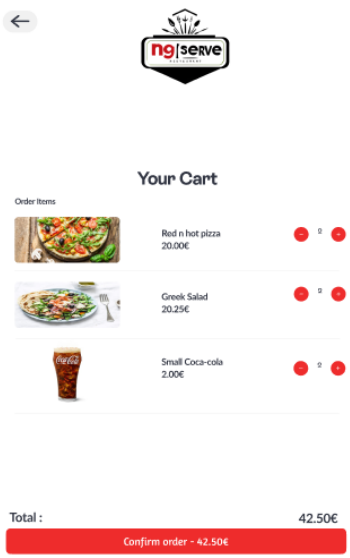



Le bouton " Cancel Order" (1) offre la possibilité d'annuler la commande parce que l'utilisateur peut avoir changé d'avis ou constaté une erreur dans sa commande. Lorsque l'on clique sur ce bouton, une page de confirmation :

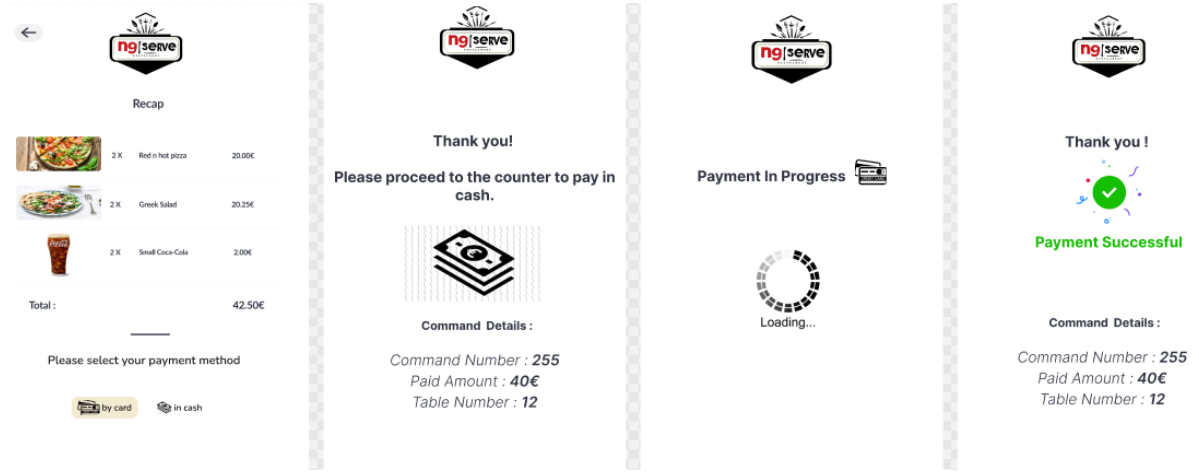



Si on choisit "Yes, cancel order", la commande sera effectivement annulée, et la page d'accueil sera affichée. En revanche, en sélectionnant "No, Continue ordering", aucune action ne sera entreprise, et on demeure sur la page actuelle avec toutes les modifications déjà apportées, y compris les plats ajoutés dans le panier.

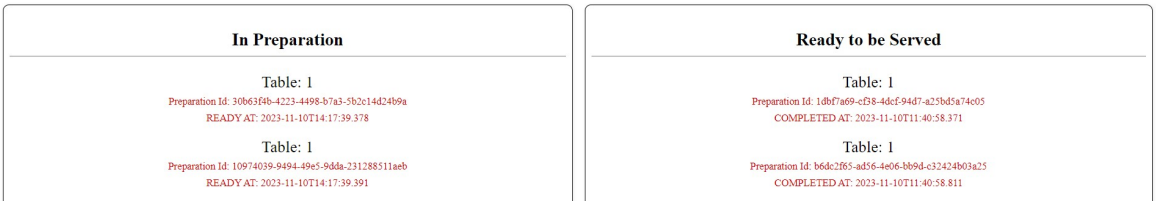
Pour le bouton "Cart" (2); il permet d'afficher les plats qui ont été ajoutés à la commande. Un indicateur numérique est présent sur ce bouton offre une visibilité instantanée sur le nombre d'articles ajoutés.
La page du panier offre une vue détaillée des plats ajoutés, avec la possibilité d'ajuster les quantités à l'aide des boutons "+/-". Le total de la commande ainsi que le prix de chaque plat sont clairement affichés, offrant ainsi une expérience conviviale et transparente pour finaliser la commande :




Une fois le paiement effectué, nous affichons le numéro de commande, permettant ainsi de suivre la préparation de la commande. Nous fournissons également une estimation du temps de préparation, le numéro de table et le montant payé:




Pour le suivi de la commande, nous avons supposé que le client consultera un tableau spécifique répertoriant les différentes commandes, indiquant leur état de préparation ou de prêt à être servi, comme suit :




## Implémentation :

- **Gestion du menu :**

Pour concevoir notre interface graphique, nous avons défini ce qui était le plus important pour l'utilisateur : être face à une interface simple et compréhensible tout en permettant d'accéder le plus rapidement possible aux produits souhaités. C'est

pour cela que nous avons imaginé un système de sous-catégories et ingrédients pour que l'utilisateur n'ait pas à chercher un plat dans une liste trop longue. Nous avons ainsi, pris la décision de restructurer notre système de microservices pour gérer les sous-catégories et ajouter les ingrédients de chaque repas, ce qui a entraîné la création d'un code générique capable d'efficacement gérer des sous-catégories spécifiques pour chaque type de restaurant. Cette avancée offre l'avantage significatif d'accroître l'adaptabilité et l'évolutivité de notre système, simplifiant ainsi l'intégration de nouveaux types de restaurants à l'avenir. En somme, notre solution peut maintenant s'ajuster de manière transparente à une variété de catégories de restaurants, renforçant sa flexibilité et sa capacité d'évolution.

- **Gestion des commandes :**

Pour le service de prendre à emporter nous avons pris la décision de garder l'implémentation du système restaurant déjà existante, nous avons envisagé deux possibilités :

- Créer une seule table virtuelle (la n°101) réservée aux commandes à emporter. Toutes les bornes utiliseraient cette table pour leur commande à emporter. Le problème de cette approche est qu'elle nous oblige à faire toutes nos requêtes en même temps pour éviter la concurrence entre les bornes. De plus, même si les requêtes sont faites en une seule fois, il existe toujours un risque de concurrence.
- Créer une table virtuelle par borne (n° 101-103). Cela nous permet d'éviter la concurrence des bornes. Cela nous permet aussi de réserver la table virtuelle au début de la commande (ce qui est plus simple pour garder la même logique que les tables non virtuelles). C'est cette possibilité qui a été retenue.

Pour le service de manger sur place, nous réservons une table dès que le client fait le choix de prendre son repas sur place. Si le client décide d'annuler la commande, la table est immédiatement libérée, assurant ainsi une gestion fluide et efficace des tables. De plus, nous avons créé un script pour libérer toutes les tables (liberate-all.sh), en supposant que cette tâche soit exécutée par le serveur.
Pour envoyer les plats sélectionnés au système de microservices, le choix qui a été fait est d'envoyer tous les plats lors de la validation de la commande. Ainsi, le client peut enlever des plats de sa commande sans faire d'appel réseau. Cela permet aussi de ne pas modifier le service restaurant car il ne prend pas en compte la suppression d'un plat dans une commande.

## III. Approche Micro-services

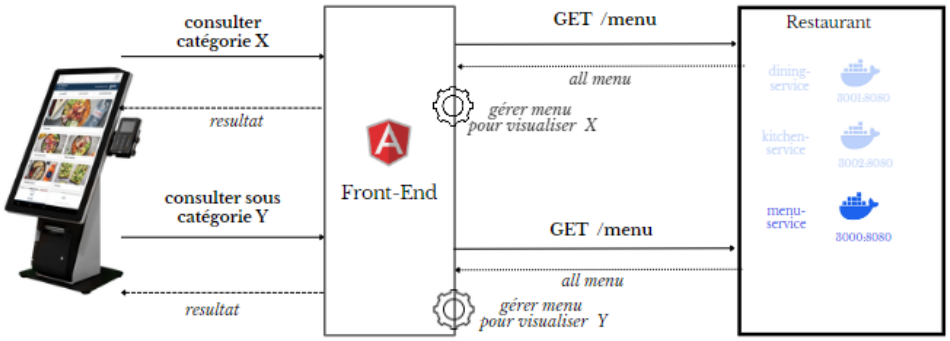
Cette approche utilise le frontend seul pour faire des requêtes aux microservices déjà existants. Nous avons développé le frontend en utilisant le framework Angular. La logique de l'application est placée dans des services qui communiquent directement avec les microservices. Ces services séparent la logique d'affichage des

données, centralisant ainsi la logique métier, favorisant la réutilisation du code et garantissant la maintenabilité du projet, tout en permettant des mises à jour en temps réel.

## Travail réalisé :

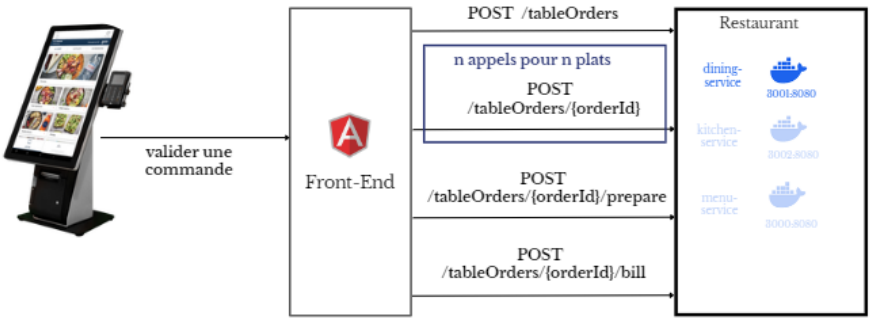
- **Gestion du menu :**

Ayant implémenté les sous catégories dans les microservices du restaurant, il a été nécessaire d'implémenter une logique conséquente dans le frontend pour traiter les données brut du menu. Dans cette approche, le frontend est responsable du stockage complet du menu et de la gestion de toute la logique métier, y compris la catégorisation et la sous-catégorisation des plats, l'association des ingrédients, ainsi que le tri des repas par ordre alphabétique.
Cependant, cette approche peut entraîner une surcharge de travail côté frontend, notamment avec des menus volumineux ou complexes, ce qui peut affecter les performances de l'application et rendre la maintenance plus complexe. La figure ci-dessous montre la communication avec les microservices pour la gestion du menu




- **Gestion des commandes :**

Pour la création et la validation d'une commande à emporter ou sur place, notre frontend effectue plusieurs requêtes vers les microservices. Initialement, il débute en créant la commande à l'aide d'une requête dédiée, distinguant entre "sur place" et "à emporter". Dans le cas d'une commande "sur place", cette requête permet de réserver la table et d'annuler toute concurrence. En revanche, pour une commande à emporter, la requête est émise lors de la validation.
Ensuite, à chaque étape de validation de plat, une nouvelle requête est générée pour ajouter le plat à la commande en cours. Enfin, une requête supplémentaire est envoyée pour indiquer que la commande est prête à être préparée. Si la commande est à emporter, une requête distincte est également émise afin de libérer la table virtuelle liée à la borne.
Le processus de la validation d'une commande à emporter est représenté dans le diagramme de séquence ci-dessous :




En ce qui concerne la gestion des commandes à emporter, une particularité de cette approche est que notre frontend est autorisé à créer des tables virtuelles associées à chaque borne, ce qui simplifie la gestion de ce type de commande, mais présente des inquiétudes en matière de sécurité, car cette gestion est effectuée côté frontend.

## IV. Approche BFF (Backend For Frontend)

Cette approche consistait à exploiter de manière efficiente les micro-services fournis afin de répondre de manière efficace aux besoins des clients. Les principaux objectifs étaient les suivants :

- Faciliter l'interaction entre le front-end et les microservices.
- Fournir une couche d'abstraction pour le front-end, simplifiant ainsi les requêtes et réponses aux microservices backend.
- Optimiser les performances en réduisant le nombre de requêtes entre le front-end et le service restaurant.

### Méthodologie :

Pour atteindre ces objectifs, les étapes suivantes ont été entreprises :

1) **Analyse des besoins** : Nous avons analysé minutieusement les besoins spécifiques du front-end ainsi que les fonctionnalités des microservices. Cette phase a permis de définir les points d'interaction cruciaux.
2) **Conception du BFF** : Sur la base de l'analyse, une conception détaillée du BFF a été élaborée, identifiant les points d'entrée, les méthodes de communication avec les microservices, et les formats de données échangées.
3) **Développement du BFF** : Le BFF a ensuite été développé en utilisant les technologies et les outils appropriés, en s'assurant de suivre les meilleures pratiques de développement.
4) I**ntégration avec le Front-End** : L'intégration du BFF avec le front-end a été réalisée avec une attention particulière à la gestion transparente des requêtes et réponses.

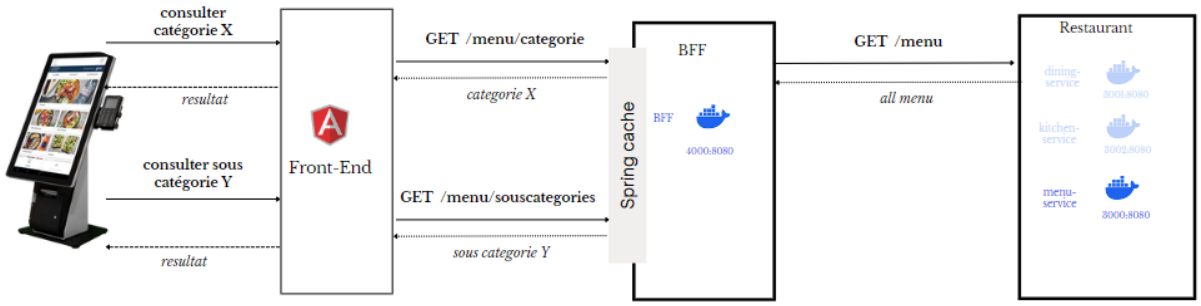
## Travail réalisé

- **Gestion du menu :**

Pour le BFF, Nous avons mis en place un mécanisme de mise en cache pour améliorer les performances. Lors du démarrage du BFF, une requête est envoyée aux microservices pour récupérer le menu, puis les données sont mises en cache pour une utilisation ultérieure. Ce cache, utilisant des clés et des valeurs, englobe également les sous-catégories et les catégories, permettant ainsi au BFF d'extraire directement les informations nécessaires, assurant des réponses plus rapides aux demandes des clients. Cette stratégie de mise en cache améliore l'efficacité de la communication entre le frontend et le BFF.

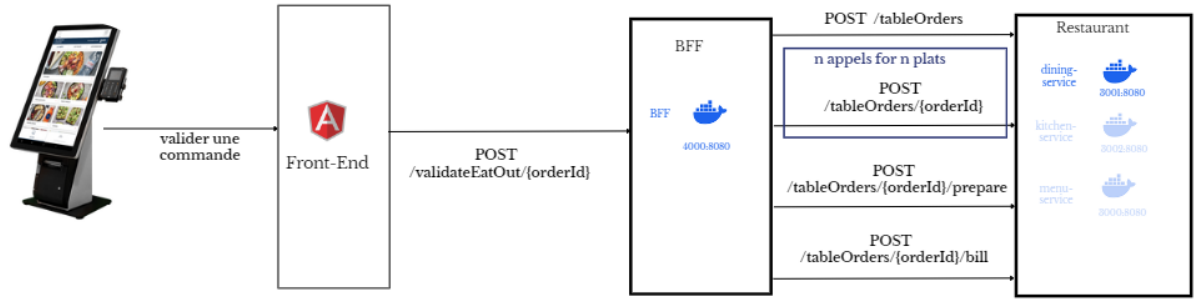
De plus, le BFF n'est pas seulement chargé de la récupération du menu, mais il gère également l'ensemble de la logique métier concernant les catégories, sous-catégories et ingrédients. Il fournit au frontend uniquement les données demandées, déjà triées et prêtes à être affichées, éliminant ainsi le besoin de gestion supplémentaire du côté du client .

La figure ci dessous montre un diagramme de séquence pour la gestion des catégories :




- **Gestion des commandes :**

Pour la gestion et la validation de la commande, notre frontend appel une seule fois le BFF pour créer la commande, ajouter tous les plats puis la valider comme montré dans la figure ci-dessous :



## V. Comparaison des 2 approches

Dans l'approche où le front-end interagit directement avec les microservices pour obtenir les données nécessaires, plusieurs points critiques sont à considérer :

- **Complexité des données :** Personnaliser les données des menus peut s'avérer complexe, car le front-end dépend étroitement des microservices. Cela peut rendre difficile l'ajustement précis des données en fonction des besoins spécifiques, nécessitant ainsi une coordination étroite avec les microservices pour les adaptations requises.
- **Stockage des données :** Le front-end conserve l'intégralité du menu, ce qui peut entraîner une surcharge de données et, à terme, provoquer des problèmes de performances, voire un crash du front-end lorsque la quantité de données sauvegardées devient excessive.
- **Multiples Requêtes :** Le processus de validation des commandes implique l'envoi de plusieurs requêtes depuis le front-end vers les microservices, ce qui peut potentiellement augmenter le temps de latence de l'application, affectant ainsi la réactivité de l'interface utilisateur.
- **Sécurité et Gestion :** Bien que le front-end ait un contrôle total sur les requêtes et les réponses, les endpoints des microservices peuvent, dans certaines situations, renvoyer plus de données que nécessaire. Cela expose des informations superflues, potentiellement sensibles, aux utilisateurs, posant ainsi un risque en matière de confidentialité. De plus, le fait que le front-end puisse créer des tables directement peut causer des problèmes de sécurité en exposant ces APIs aux clients.

D'un autre côté, dans l'approche où le BFF est introduit comme une couche intermédiaire entre le front-end et les microservices, plusieurs avantages significatifs sont observés :

- **Simplicité de communication :** Chaque interface utilisateur n'accède qu'aux données et fonctionnalités qui lui sont appropriées. Le BFF agit comme un filtre et agrège les requêtes, minimisant ainsi le nombre d'appels au service restaurant. En particulier, lors de la validation de la commande, au lieu d'effectuer n + 2 appels depuis le front-end en cas de n plats, grâce au BFF, un seul appel global est suffisant. Cette optimisation réduit considérablement le trafic inutile et améliore la performance globale de l'application.
- **Indépendance des changements des microservices :** Les modifications apportées aux microservices peuvent être gérées au niveau du BFF, sans affecter directement le front-end. Un exemple concret de cette indépendance se manifeste dans la gestion du menu, où des modifications ont été apportées aux microservices et le BFF a pris en charge d'intégrer les sous-catégories. Cette approche offre une flexibilité précieuse lors de l'ajout, de la modification ou de la suppression de fonctionnalités du système restaurant, tout en maintenant l'intégrité du front-end.

- **Sécurité :** Le BFF joue un rôle essentiel dans l'implémentation de mécanismes de sécurité. Le front-end n'a accès qu'aux ressources nécessaires, et dans notre cas, le BFF gère la création des tables virtuelles, renforçant ainsi la protection des interactions utilisateur. Cette approche renforce la sécurité en contrôlant l'accès aux fonctionnalités sensibles et en minimisant les risques potentiels

En conclusion, l'introduction du BFF présente des avantages significatifs en matière de personnalisation, de sécurité, de performance et de flexibilité face aux évolutions du service restaurant. De plus, le BFF offre la possibilité d'exploiter de manière efficiente des solutions existantes sans nécessiter de modifications ou de développements supplémentaires, ce qui peut représenter un avantage économique considérable.

# Evolution des besoins

En réponse à l'évolution des besoins, nous avons repensé l'expérience de commande en introduisant des tables interactives. Ces tables, conçues pour accueillir de 2 à 4 clients, offrent une approche innovante pour passer commande. Chaque client a désormais la possibilité de faire sa propre sélection tout en contribuant à une commande groupée. Un client peut choisir de prendre en charge le paiement total ou de régler uniquement sa part. De plus, pendant le temps qui s'écoule entre la commande et la livraison des plats, les clients peuvent profiter d'activités interactives sur la table. De plus, ils sont régulièrement informés de l'état d'avancement de la préparation de leur commande grâce à une tablette visible pour tous.

## I. Analyse des besoins :

À la suite de ces évolutions, nous avons identifié de nouveaux personas en complément de ceux initialement définis dans la première extension :

### Persona 1 : Myriam

- Âge : 29 ans
- Style de vie : Elle aime passer du temps avec ses collègues en dehors du travail et organise régulièrement des sorties en groupe.
- Motivation : Myriam préfère faire une commande groupée car cela lui permet de faciliter le processus et de créer un moment convivial avec ses collègues. Elle aime coordonner les préférences de chacun pour s'assurer que tout le monde se régale et passe un bon moment ensemble. Cela renforce également l'esprit d'équipe au sein de l'entreprise.

### Persona 2 : Marc

- Âge : 50 ans
- Style de vie : Il est dévoué envers sa famille et aime passer du temps de qualité avec ses enfants et petits-enfants.
- Motivation : Marc préfère prendre en charge le paiement total de la commande pour permettre à sa famille de profiter pleinement de l'expérience sans se soucier des détails financiers.

### Persona 3 :Thomas

- Âge : 9 ans
- Style de vie : Thomas adore dessiner, colorier et exprimer sa créativité à travers l'art.
- Motivation : Thomas est content de pouvoir colorier sur la table interactive pendant l'attente. Cela lui permet d'exprimer sa créativité tout en s'amusant.

### Persona 4 : Laura

- Âge : 35 ans
- Style de vie : Laura est une mère de famille soucieuse de son budget et aime gérer ses dépenses de manière équilibrée.
- Motivation : Elle préfère payer uniquement sa part de la commande pour maintenir un équilibre financier tout en participant à une expérience de repas en groupe.

Suite à notre analyse des besoins du client, les fonctionnalitées identifiées sont les suivantes :

- **Sélection individuelle et commande groupée :** Chaque client a désormais la possibilité de faire sa propre sélection tout en contribuant à une commande groupée.
- **Options de paiement flexibles :** Un client peut choisir de prendre en charge le paiement total ou de régler uniquement sa part.
- **Activités interactives :** Les clients peuvent profiter d'une activité de coloriage sur la table pendant le temps qui s'écoule entre la commande et la livraison des plats.
- **Suivi de l'état d'avancement de la commande :** Les clients sont régulièrement informés de l'état d'avancement de la préparation de leur commande grâce à une tablette visible pour tous.

## II. Décisions prises

Lors de notre processus de sélection d'architecture, nous nous sommes basées sur la comparaison entre les deux approches différentes : celle avec l'utilisation de l'architecture BFF (Backend For Frontend) et celle sans. Après une analyse

approfondie des avantages et des inconvénients de chaque méthode, il est devenu évident que l'adoption de l'architecture BFF était la solution optimale pour répondre aux exigences de notre système.

Avec l'architecture BFF (Backend For Frontend), nous avons pu modeler notre système pour répondre spécifiquement à nos besoins en matière de fonctionnalités liées à l'interface utilisateur. Cette approche nous a offert la souplesse nécessaire pour mettre en place une gestion des commandes groupées de manière optimale. En comparaison, l'autre approche envisagée aurait nécessité de prendre des décisions difficiles. D'un côté, il aurait été envisageable de stocker toutes les commandes dans la partie front-end de l'application et de les envoyer au système de microservices une fois que tous les clients ont fait leurs choix. Cependant, cela aurait pu entraîner des problèmes de performance et de gestion des données. D'un autre côté, il aurait été possible de modifier de manière significative le service restaurant pour gérer les commandes groupées, mais cela aurait pu nécessiter des efforts de développement importants et potentiellement compliquer la maintenance du système. De plus, l'architecture BFF facilite la gestion des requêtes et des réponses entre le frontend et les microservices. Cela nous permet d'optimiser la communication entre les deux parties, garantissant ainsi des performances et une réactivité accrues.

Parallèlement, pour satisfaire les besoins spécifiques des nouvelles fonctionnalités introduites, nous avons intégré de nouveaux composants essentiels. Ces composants comprennent un module de jeu pour divertir les clients, un mécanisme permettant d'identifier les clients par leur nom pour une personnalisation accrue, ainsi qu'une fonction qui interroge les clients sur leur préférence de paiement, qu'il soit groupé ou individuel.

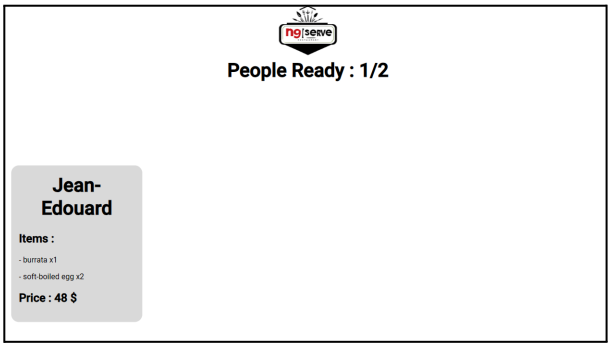
## III. Impact sur l’IHM

En ce qui concerne l'impact sur l'interface utilisateur (IHM), nous avons conservé la structure globale pour la tablette client, puisque l’interface est responsive donc elle s'adapte aussi bien à la borne verticale qu’à l’interface horizontale d’un côté de la table tactile. A l'exception de la page permettant de choisir entre manger sur place ou à emporter, car cette fonctionnalité est désormais exclusivement dédiée à la consommation sur place.

Pour l’écran central, nous avons considéré qu’il s’agit d’un écran orienté classique.

L'écran d'accueil invite le groupe d'utilisateurs à indiquer leur nombre, permettant ainsi, dans une implémentation sur table tactile, d'afficher le nombre approprié d'écrans personnels. Une fois la taille du groupe indiquée, l'écran central affiche le nombre et les noms des personnes ayant validé leur commande, ainsi que les

produits commandés. Cet écran est actualisé en temps réel pour suivre l'avancée des commandes.


Une fois que tout le monde a commandé, le groupe est en attente des plats, et peut jouer à un jeu. Chacun peut faire une action sur son interface personnelle et le retour de ces actions est affiché en temps réel sur l'écran central. En même temps, les clients peuvent voir l'avancée de la préparation de la commande sur l'écran central.
Une fois que le repas est fini, l'un des clients peut choisir s'ils veulent séparer la note ou pas. Suite à cela, deux possibilités :

- La note est commune et une seule personne paye sur le terminal de paiement
- La note est séparée et chacun paye à tour de rôle sur le terminal de paiement



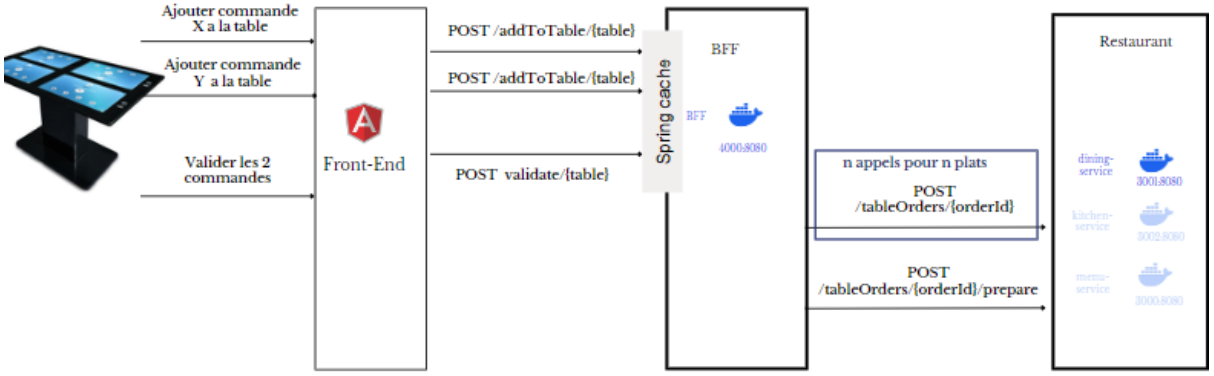
## IV. Impact sur l'architecture BFF et microservices

Aucune modification donc n'a été apportée au système de microservices du restaurant pour intégrer ces nouveaux besoins. L'architecture de base du BFF est restée essentiellement inchangée, les ajustements se concentrant principalement sur l'intégration de nouvelles API pour incorporer les fonctionnalités supplémentaires. Cette approche s'est avérée efficace, nous permettant de répondre aux nouveaux besoins tout en préservant la stabilité de notre architecture existante.

- **Gestion des commandes groupées :**

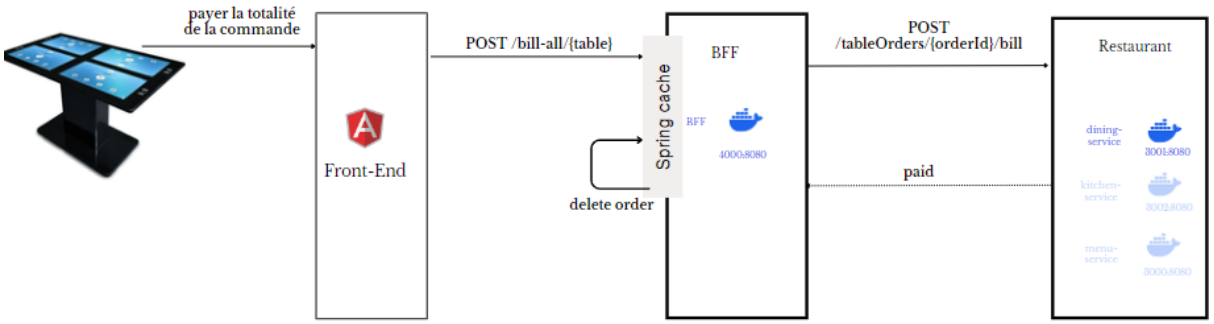
Les commandes groupées sont gérées comme une seule commande de la même manière dans la version 1 du projet. La seule différence réside dans le fait que lorsque le client termine sa commande, celle-ci est enregistrée dans le cache du

BFF. Le diagramme de séquence ci-dessous montre le processus de passer une commande groupée :

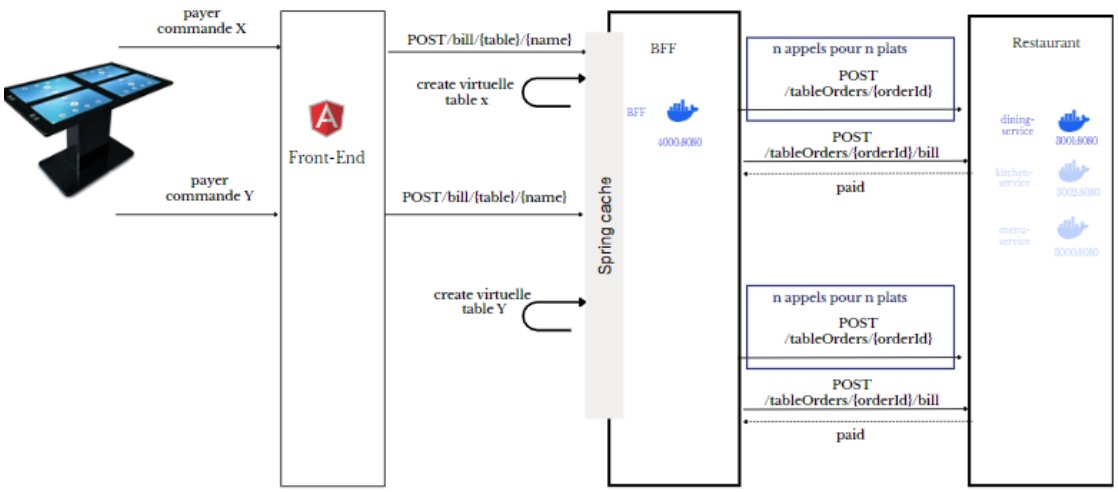



- **Gestion du paiement :**

Le processus de paiement total des commandes groupées est géré de la même manière que dans la version 1, avec l'envoi d'une seule requête au système de microservices du restaurant.

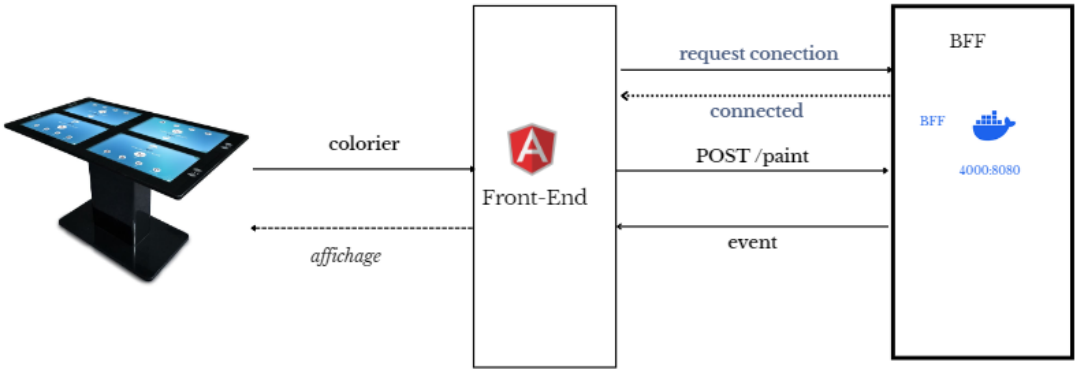



Pour gérer les paiements individuels, nous avons réutilisé le code déjà développé dans le BFF pour les commandes à emporter. Pour une commande à emporter, nous créons une table virtuelle par commande. Nous avons étendu cette idée pour gérer les paiements individuels en créant une table virtuelle contenant ses plats par personne, ce qui nous permet ainsi d'appeler le système de paiement associé aux microservices pour chaque personne comme montré dans le diagramme ci-dessous:




- **Gestion du jeu :**

La partie du jeu que nous implémentons est très simple : chaque client choisit une couleur et l'utilise pour colorier sur la tablette. Le coloriage effectué par chaque client est affiché. Le frontend envoie une requête au BFF pour enregistrer le coloriage de chaque client. De plus, pour garantir que les joueurs restent constamment à jour, toute action effectuée est immédiatement reflétée à l'écran, offrant ainsi une expérience de jeu immersive et dynamique. Cette communication en temps réel avec le frontend est réalisée grâce à l'utilisation des événements Server-sent (Server-sent events).




- **Suivi des commande :**

L'utilisation des événements Server-sent était également utilisée pour les commandes. Dès qu'un client finalise sa commande, elle s'affiche instantanément sur la tablette centrale. De même, pour le suivi de la commande, lors du changement du statut de la commande, que ce soit la fin de la préparation ou pour la livraison, ce changement est instantanément reflété sur la tablette centrale, permettant ainsi d'être constamment à jour concernant le statut de la commande. Sans l'utilisation de ces événements, nous aurions dû recourir au polling en envoyant des requêtes chaque seconde pour récupérer les données du BFF, ce qui aurait surchargé le BFF.

# V. Prise de recul

L’approche Backend for Frontend (BFF) a considérablement simplifié la réutilisation du code, représentant ainsi une avancée significative dans le développement et la maintenance du système. Le BFF nous a permis d'optimiser l'adaptation du code pour répondre efficacement aux nouvelles fonctionnalités requises. De même, l'utilisation du système de cache comme une base de données temporaire s'est avérée être une stratégie astucieuse. Elle nous a facilité l’intégration des nouvelles fonctionnalités comme commande groupée, gestion du jeu, tout en préservant l'intégrité du code existant et de le réutiliser de manière efficiente.

Le BFF a apporté une flexibilité considérable au projet, facilitant ainsi son adaptation aux impératifs du changement dans un environnement dynamique. Un exemple concret d'une fonctionnalité qui aurait pu être intégrée réside dans la possibilité d'annuler une commande tant qu'elle n'a pas été transmise en cuisine. Cette option aurait apporté un bénéfice notable, d'autant plus que sa faisabilité est significativement renforcée grâce à la mise en place du cache dans le BFF.

Néanmoins, cette approche, bien qu'encourageant la réutilisation du code, présente une limitation. Elle restreint la possibilité de fournir un suivi personnalisé de commande pour chaque utilisateur, car les commandes sont traitées comme une entité unique. Bien qu'il aurait été envisageable d'adapter la commande en fonction du nom de l'utilisateur, nous avons choisi de privilégier l'efficacité du code existant.
Cela met en lumière que le BFF nous a permis d'ajouter facilement de nouvelles fonctionnalités, bien que pour une implémentation optimale, il aurait été pertinent dès la première version d'avoir un code très générique dans le BFF.

# Répartition des tâches

***Maquettes***

Pour les différentes maquettes de nos interfaces, nous avons tous participé à la définition des besoins et au design afin de comprendre les points clé du projet et de tous se familiariser avec le fonctionnement de l'interface.

***Partie 1 du projet (commande sur borne)***

**Implémentation du bff :**

En collaboration, *Sourour* et *Nadim*, nous avons travaillé ensemble sur la définition des users stories, définissant ainsi les fonctionnalités essentielles et les scénarios d'utilisation du système.

*Sourour* a pris en charge la gestion complète du menu, en mettant l'accent sur la création de sous-catégories, de catégories et la gestion des ingrédients. Pendant ce temps, *Nadim* s'est concentré sur la gestion des tables, ce qui inclut la création des tables virtuelles (pour les commandes à emporter) ainsi que la gestion des paiements.

**Implémentation du frontend :**

Pour le frontend communiquant directement avec les micro services ainsi que le frontend communiquant avec le bff, *Tobias* s'est occupé principalement d'implémenter l'interface graphique (utilisabilité, user expérience, adaptabilité à l'écran). *Mathieu* a majoritairement implémenté la logique pour les appels aux micro services et au bff.

***Partie 2 du projet (tables interactives)***

**Evolution du bff :**

*Nadim* s'est dédié à l'intégration de nouvelles fonctionnalités dans le code et à implémenter un système de cache pour les commandes groupées, tandis que Sourour s'est concentrée sur la liaison du code existant avec les nouveaux composants ajoutés et la mise en œuvre des Server-sent events pour la diffusion automatique de mises à jour en temps réel concernant le jeu et les commandes sur la tablette centrale.

**Evolution du frontend-bff :**

Concernant les interfaces clients de la table, nous n'avons quasiment pas eu besoin de modifier l'interface (le design était déjà responsive pour passer en mode horizontal). Pour l'implémentation de l'écran central, notre répartition du travail a été la même que pour la première partie du projet (*Tobias* pour l'interface graphique et *Mathieu* pour la logique).