



M1 Informatique – Rapport PSTL

Noms, prénoms et spécialité :

BRAHIMI Lounes	M1 STL
ANSARI TABRIZI Edwin	M1 STL

Sujet :

Localisation de features dans les applications Web

1. Introduction :

Tout logiciel nécessite une maintenance continue pour continuer à fonctionner et à servir le client, au cours de son cycle de vie, les développeurs exécutent des activités en constante évolution, principalement en raison du besoin continu de corriger des bogues, de mettre à jour, d'adapter le logiciel et d'améliorer sa productivité en ajoutant de nouvelles fonctionnalités, la compréhension de la structure du système logiciel est donc une nécessité avant que des modifications puissent être apportées, dans ce cas, le développeur doit identifier l'emplacement dans le code source qui correspond à une fonctionnalité spécifique, c'est ce qu'on appelle la localisation de fonctionnalités.

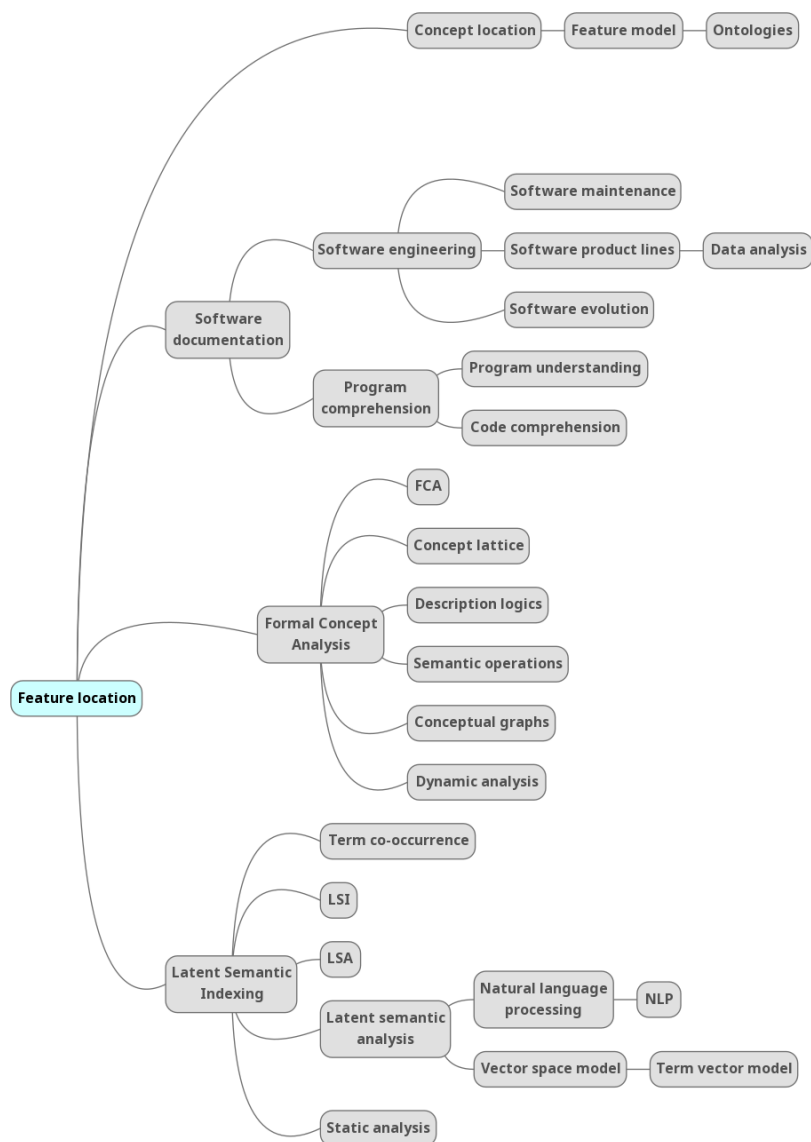


Figure 1 Carte heuristique

2. Contexte :

La localisation de features consiste à détecter des portions de codes représentant des fonctionnalités recherchées au sein du code source d'une application, la technique utilisée est l'analyse sémantique latente. Cette dernière consiste à analyser les relations entre un ensemble de documents et les mots qu'ils contiennent pour produire un ensemble de concepts liés aux documents et aux mots.

Selon une requête textuelle de longueur variable l'algorithme propose en réponse une liste ordonnée des documents selon leurs pertinences, ainsi qu'une liste de fonctionnalités au sein de chaque document résultant triées elles aussi selon leurs pertinences vis-à-vis de la requête.

L'algorithme "LSA" est conçu sous l'hypothèse que les mots dont le sens est proche ont une grande chance de figurer dans les mêmes documents. Ce dernier utilise plusieurs méthodes de différentes natures et procède selon plusieurs étapes.

L'algorithme traite les documents et génère un dictionnaire de termes (pertinents) qui sera utilisé pour la construction d'une matrice où les lignes représentent les termes élus et les colonnes représentent les documents. Ensuite, une méthode de pondération nommée "TF-IDF" est utilisée pour donner plus de sémantique aux poids présents dans les cellules de la matrice, un procédé d'algèbre linéaire nommé "décomposition en valeurs singulières" est appliqué sur la matrice pour réduire la taille de données tout en préservant la similarité, la requête est traduite en un vecteur. Enfin, un dernier calcul "la similarité cosinus" est appliquée sur le vecteur de la requête et le vecteur d'une matrice résultant du SVD pour produire les scores des documents, ces derniers sont classés par ordres décroissants.

3. Objectifs :

L'objectif principal du projet est d'implémenter un algorithme permettant la localisation de l'intégralité des fonctionnalités recherchées dans le code source d'une application, le résultat de cette détection doit être visualisé de façon expressive et manipulé de façon simple par l'utilisateur, ainsi, l'application est une extension de VSCode dont le client peut importer une application, exécuté

l'algorithme de localisation en saisissant une requête de façon textuelle ou bien en exécutant l'algorithme sur un fichier script contenant un ensemble de requêtes, dans ce dernier cas l'utilisateur devra avoir plusieurs résultats où chacun correspond à une fonctionnalité, le client doit visualiser les résultats dans une tree-view (structure arborescente) où les documents affichent des ranges qui correspondent aux portions de codes ou figure la fonctionnalité recherchée. L'extension doit être validée sur une application web et présenter les résultats.

4. Travail Réaliser :

Phase 1 : Ignorer les termes superflus :

L'algorithme prend en entrée une liste de mots à ignorer. Cette dernière correspond aux termes qui figurent dans la grammaire d'un langage informatique (if, while) ou des mots de liaisons et autres utilisés dans la sémantique d'un langage naturel (il, elle, à). De plus, il ignore un ensemble de caractères spéciaux.

Ce premier traitement est effectué car cette collection d'éléments du langage n'a pas de valeur sémantique et le fait qu'ils sont souvent présents à répétition dans tous les documents, ils perturbent le contexte de recherche.

Phase 2 : Génération du dictionnaire :

Les documents étant représentés tel une très large collection de mots d'en l'ordre n'a aucune importance. Mis à part les termes ignorés à l'étape précédente, tous les mots n'apportent pas une réelle valeur ajoutée au contexte du document. Les mots d'index définissent des termes qui peuvent apporter un indice sur la sémantique du document. Ces derniers sont les mots hors ceux ignorés à l'étape précédente et qui sont présents plus d'une fois dans le même ou dans plusieurs documents.

L'algorithme parcourt donc tous les documents issus de la phase antécédente et génère une liste de jetons des mots. Enfin, il construit le dictionnaire qui est constitué de l'ensemble de mots index et les identifiants des documents auxquels ils appartiennent.

Phase 3 : Construction de la matrice de termes par documents :

Dans cette phase l'algorithme définit une matrice où chaque mot d'index est une ligne et chaque document est une colonne, toute cellule représente le nombre de fois que le terme apparaît dans le document.

1	1	1
0	1	1
1	0	0
0	1	0

Figure 2 Matrice M

Phase 4 : Term frequency-inverse document frequency :

Le TF-IDF est une méthode de pondération utilisée sur la matrice obtenue à la section précédente, les poids des termes présents dans les cellules sont modifiés de sorte que les mots rares sont plus pondérés que les mots courants. Ainsi, un terme apparaissant dans seulement un petit nombre de documents devrait probablement être pondéré plus lourd qu'un autre qui apparaît dans presque tout documents. Enfin, le concept de cette méthode est le calcul suivant qui est effectué sur chaque cellule :

1. Calcul de TF :

Le nombre d'apparition du terme dans le document divisé par le nombre total de termes dans le document.

2. Calcul de IDF :

Le nombre de documents divisé par le nombre de documents dans lesquels le mot apparaît.

3. Calcul de TF-IDF :

$$\text{TF-IDF} = \text{TF} * \log(\text{IDF}).$$

1.89	1.26	1.89
0	1.49	2.23
2.23	0	0
0	1.49	0

Figure 3 Matrice M après TF-IDF

Phase 5 : Décomposition en valeurs singulières :

Le « svd » est un procédé d'algèbre linéaire qui transforme la matrice obtenue en une représentation dimensionnelle avec le moins d'informations possible qui met l'accent sur les relations les plus fortes. Plus concrètement, Il convertit la matrice en une « relation » entre les termes et des « concepts », et une relation entre ces concepts et les documents.

L'algorithme factorise la matrice en 3 autres ($M = USV^*$). La matrice « U » donne les coordonnées de chaque terme sur l'espace « concept », la matrice « V^* » donne les coordonnées de chaque document dans l'espace « concept », et la matrice « S » de valeurs singulières nous donne un indice sur le nombre de dimensions ou « concepts » que nous devons inclure.

4	0	0
0	2.5	0
0	0	1.2

Figure 3 Matrice S

-0.51	0.85	-0.06
-0.52	-0.37	-0.76
-0.67	-0.35	0.64

Figure 4 Matrice V^*

-0.73	0.19	0.10
-0.57	-0.55	0.25
-0.28	0.77	-0.12
-0.19	-0.22	-0.95

Figure 5 Matrice U

Enfin, une approximation de rang 2 est obtenue en tranchant les matrices, les deux premières colonnes de « U » et « V » sont conservées, ainsi que les deux premières colonnes et lignes de « S ».

4	0
0	2.5

Figure 6 Matrice S

-0.51	0.85
-0.52	-0.37
-0.67	-0.35

Figure 7 Matrice V^*

-0.73	0.19
-0.57	-0.55
-0.28	0.77
-0.19	-0.22

Figure 8 Matrice U

Phase 6 : Génération du vecteur de la requête :

Le texte de la requête tout comme les documents subie le même traitement, les mots à ignorer et les caractères spéciaux sont supprimés, une liste de jetons de termes est construite, l'algorithme génère un vecteur « Q » de longueur égale au nombre de mots d'index, ses valeurs sont à 0, pour chaque mot d'index dans la requête il incrémente son indice dans le vecteur.

0	1	0	1
---	---	---	---

Figure 9 Vecteur Q

Ensuite, une nouvelle méthode de pondération similaire à TF-IDF est utilisée, cette dernière augmente le score des mots-clés pertinents dans la requête et pénalise les moins importants, ainsi, si un terme apparaît dans plusieurs requêtes il est défavorisé.

0	1.3	0	0.5
---	-----	---	-----

Figure 10 Vecteur Q

Ensuite, de nouvelles coordonnées sont calculées pour le vecteur de la requête dans l'espace bidimensionnel à dimension réduite obtenue avec la décomposition en valeurs singulières selon le calcul suivant :

$$Q = Q \cdot S$$

$$S = S^{-1}$$

$$Q = Q \cdot S$$

-0.19	-0.31
-------	-------

Figure 11 Vecteur Q

Phase 7 : La similitude cosinus :

La similitude cosinus est calculée en faisant le produit scalaire entre le vecteur de requête « Q » et le vecteur de document « V* », le résultat est divisé par le produit du module du vecteur de requête « Q » et du vecteur de document « V* ».

-0.45	0.92	0.86
-------	------	------

Figure 12 Score des documents par indice correspondant

Enfin, les documents sont triés par ordre décroissant de scores, plus le document est classé parmi les premiers plus il est pertinent.

0.92	0.86	-0.45
------	------	-------

Figure 13 documents par ordre de pertinence selon le score

Phase 8 : Extension VSCode avec tree-view :

Au début quand l'algorithme lit le répertoire contenant les documents et génère le dictionnaire, ce dernier stock les ranges de chaque mot d'index, il donne en sortie une structure représentant les documents triés par ordre de pertinence selon leurs scores, ainsi que les ranges de chaque fonctionnalité, tel que toutes les fonctionnalités résultantes sont triées et accompagnées de leurs scores spécifiques.

La structure résultante de l'algorithme est reçue par l'extension VSCode qui la transforme en tree-view pour une meilleure visualisation et manipulation de l'application.

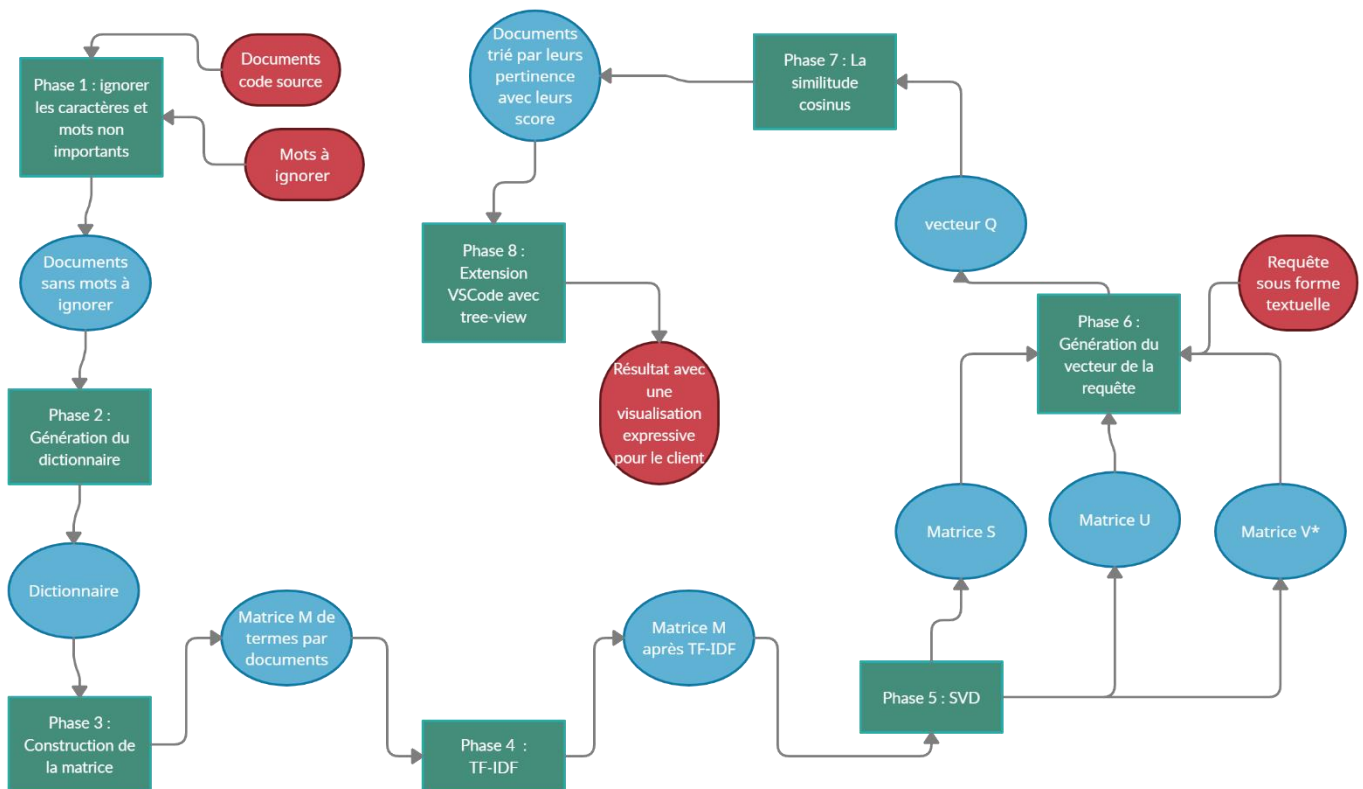


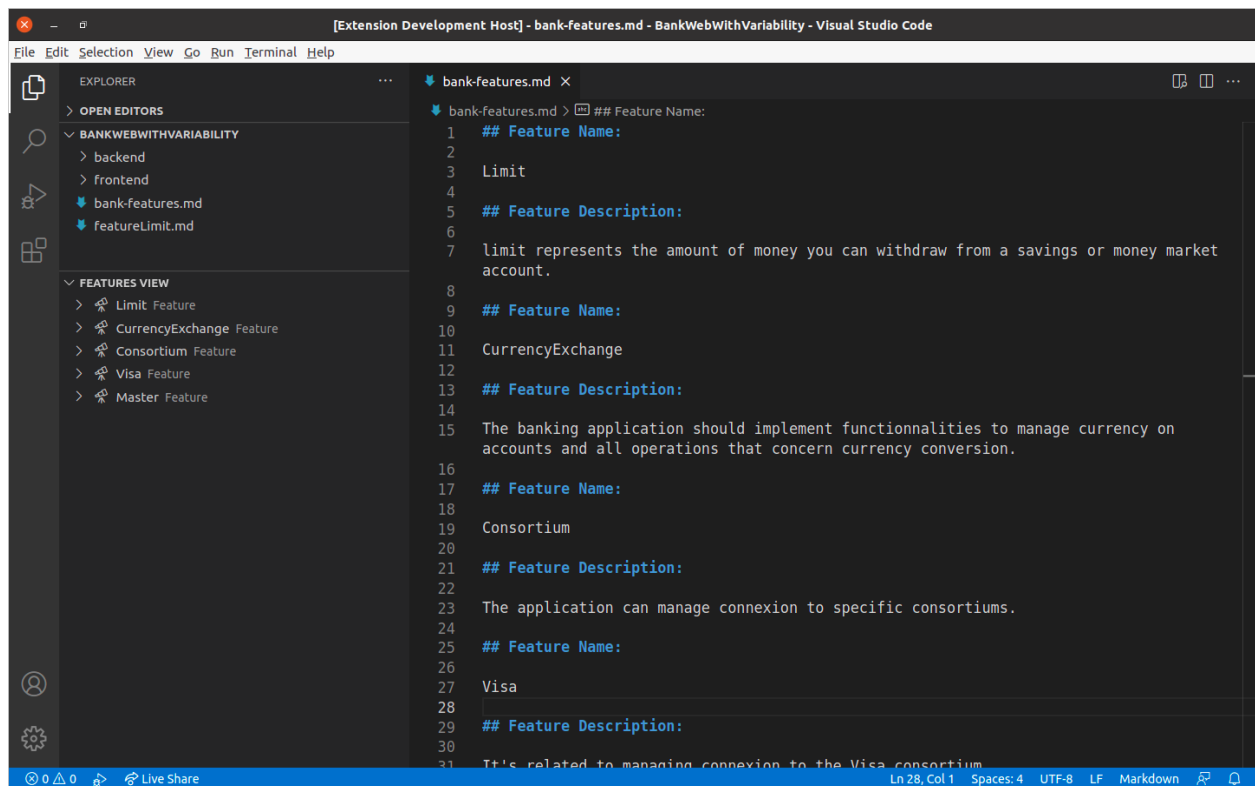
Figure 14 Workflow

4. Test de validation et expérimentation :

Après avoir terminé les étapes d'analyse, de conception, d'implémentation et de tests unitaires. L'application a été testée sur des vrais artefacts logiciels afin de faire une comparaison entre les résultats obtenus (actual value) et les résultats attendus (expected value) mais aussi pour expérimenter la performance et voir si l'application passe bien à l'échelle.

Les codes sources testées sont des applications web open-source disponibles sur internet en public. Tout de même l'outil de localisation de feature est facilement configurable pour trouver des résultats les plus pertinents possible dans n'importe quel type de base de code logiciel.

Voici donc quelques exemples d'expérimentation :



```
[Extension Development Host] - bank-features.md - BankWebWithVariability - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
> OPEN EDITORS
  > BANKWEBWITHVARIABILITY
    > backend
    > frontend
    > bank-features.md
    > featureLimit.md
  > FEATURES VIEW
    > Limit Feature
    > CurrencyExchange Feature
    > Consortium Feature
    > Visa Feature
    > Master Feature

bank-features.md
1  ## Feature Name:
2
3  Limit
4
5  ## Feature Description:
6
7  limit represents the amount of money you can withdraw from a savings or money market
  account.
8
9  ## Feature Name:
10
11  CurrencyExchange
12
13  ## Feature Description:
14
15  The banking application should implement functionalities to manage currency on
  accounts and all operations that concern currency conversion.
16
17  ## Feature Name:
18
19  Consortium
20
21  ## Feature Description:
22
23  The application can manage connexion to specific consortiums.
24
25  ## Feature Name:
26
27  Visa
28
29  ## Feature Description:
30
31  It's related to managing connexion to the Visa consortium.

Ln 28, Col 1 Spaces: 4 UTF-8 LF Markdown Live Share
```

Application BankWebWithVariability :

Logiciel	Requête	Nombre de documents souhaités	Nombre de documents trouvés	Nombre de Range trouvé	Taux de réussite	Temps d'exécution
BankWebWith Variability	Limit	2	4	21	100%	5 seconde indexation + 10ms requête
BankWebWith Variability	CurrencyExchange	5	4	13	100%	5 seconde indexation + 10ms requête
BankWebWith Variability	Consortium	3	3	27	100%	5 seconde indexation + 10ms requête
BankWebWith Variability	Visa	2	2	9	100%	5 seconde indexation + 10ms requête
BankWebWith Variability	Master	2	4	17	66%	5 seconde indexation + 10ms requête

Application EShopOnContainers :

Ce logiciel est un “container” de web, il sert à créer d'autres applications web, donc la taille du code source est très important comparée à d'autres artefacts.

Logiciel	Requête	Nombre de documents souhaités	Nombre de documents trouvés	Nombre de Range trouvé	Taux de réussite	Temps d'exécution
eShopOnContainers	Catalog	3	6	14	80%	11 minutes indexation + 40s requête
eShopOnContainers	Identity	4	4	17	60%	11 minutes indexation + 40s requête
eShopOnContainers	Basket	5	7	19	60%	11 minutes indexation + 40s requête

5. Conclusion :

Dans le cadre de ce projet nous avons travaillé sur deux majeure partie en binôme :

1. L'Implémentation de l'algorithme :

Dans cette partie nous avons appris plusieurs techniques et algorithmes de traitement de langage naturel tel que “Analyse sémantique latente”, “Analyse formelle de concepts”, “term frequency-inverse document frequency”, “décomposition en valeurs singulières” et autres. Nous avons implémenté l'algorithme LSA dans le langage TypeScript que nous ne

connaissions pas auparavant et avons appris en autonomie à l'aide de la documentation et qui s'est avéré très intéressant vu qu'il nous permet de passer outre les problèmes de sécurité de JavaScript.

2. Extension VSCode :

Afin de traiter les requêtes de l'utilisateur et de visualiser le résultat sous un format pratique et concis nous avons appris à créer des extensions VSCode à l'aide de la documentation de l'API. On a appris à afficher les résultats en tree-view, faire des alias à des portions de code avec les ranges (intervalle de code). Ainsi qu'intégrer la recherche par requête dans l'extension et l'importation de répertoire.

3. Evolutions :

L'algorithme "analyse formelle de concepts" est considéré comme le concurrent de LSA, une évolution intéressante qu'on aurait aimé implémenter est la fusion des deux algorithmes pour obtenir les résultats les plus exhaustifs possibles.