

SmartAgenda :

Outil de prise de rendez-vous automatique

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

Omar Lo

Conseiller au travail de Bachelor :

Philippe Dugerdil, Professeur HES

Genève, le 01 septembre 2017

Haute École de Gestion de Genève (HEG-GE)

Filière Informatique de Gestion

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du Bachelor of Science en Informatique de gestion

L'étudiant atteste que son travail a été vérifié par un logiciel de détection de plagiat.

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Genève, le 01 septembre 2017

Omar Lo

Remerciements

Je tiens tout d'abord à adresser mes sincères remerciements à mon conseiller au travail de Bachelor, Monsieur Philippe Dugerdil pour son engagement et sa disponibilité tout au long de ce travail. Ses encouragements, ses conseils et remarques m'ont permis de surmonter les multiples obstacles rencontrés durant la réalisation de ce projet.

Je tiens à remercier aussi Monsieur Pierre Kauffmann pour ses explications qui m'ont permis de mieux comprendre certains APIs Watson.

Mes remerciements vont également à l'endroit du personnel du centre informatique de la HES-SO Genève de m'avoir autorisé et aidé à accéder au serveur Exchange de la HEG avec mon application.

Enfin je remercie tous mes amis et proches qui m'ont soutenu tout au long de ce projet.

Résumé

Ce travail a pour but de créer un outil permettant à un groupe de collègues d'alléger leurs processus collaboratifs pour la création des réunions.

L'outil composé de deux parties, fournit une interface capable d'interagir avec un utilisateur à travers un enregistrement vocal ou textuel.

La première partie de l'application est dotée d'une « intelligence » grâce à l'utilisation des APIs cognitifs de Watson lui permettant :

- de transcrire l'enregistrement audio en texte
- d'extraire des noms de personne dans le texte
- d'extraire des contraintes temporelles (date et heure) dans le texte
- d'extraire des noms de lieu dans le texte

La deuxième partie de l'outil utilise les APIs Microsoft Exchange pour accéder aux ressources du serveur Exchange afin de réaliser les fonctionnalités proprement dites de l'application.

En utilisant ces deux technologies, l'application pourra interpréter un texte pour identifier les participants et les contraintes temporelles et interroger le serveur Exchange pour proposer et créer une réunion à la meilleure date.

Avant d'aborder la réalisation proprement dite de l'application, nous avons fait une étude sur les APIs Watson et Exchange. Cette étude a permis d'explorer les différents APIs mis en place par IBM à travers sa plateforme Watson. Ces APIs de type cognitif fournissent des services capables de traiter et de comprendre le langage naturel.

Cette étude a permis également d'explorer l'API Exchange développé par Microsoft pour externaliser les ressources de Microsoft Exchange Server. Cet API nommé EWS-JAVA-API permet de développer des applications clientes pouvant manipuler les boîtes aux lettres de Microsoft Exchange Server.

Le reste du document traite les différentes phases de réalisation de l'application en allant à la spécification, à la conception et au développement.

Table des matières

Déclaration	i
Remerciements.....	ii
Résumé	iii
Liste des tableaux	vi
Liste des figures	vi
1. Introduction	9
2. Présentation du problème à résoudre	10
3. Introduction aux APIs Watson.....	10
3.1 Watson IBM c'est quoi ?.....	10
3.2 Présentation générale des APIs Watson	11
3.2.1 API Watson Document Conversion	11
3.2.2 API Watson Tone Analyzer.....	11
3.2.3 Speech to Text.....	12
3.2.4 Text to Speech.....	12
3.2.5 Language Translator.....	12
3.2.6 Natural Language Understanding	12
3.2.7 API Watson Conversation.....	12
3.3 Exemple d'architecture des applications Watson	13
4. Présentation de l'API Microsoft Exchange utilisé	14
4.1 EWS c'est quoi ?	15
5. Architecture de la solution.....	15
5.1 Périmètre de l'outil.....	15
5.2 Spécification.....	16
5.2.1 Traitement des uses case.....	16
6. Conception	19
6.1 Présentation de l'architecture de la solution	19
6.2 Explication de l'architecture de la solution	19
6.2.1 Composant domaine.....	19
6.2.2 Composant servicesWatson	20
6.2.3 Composant servicesExchange	20
6.2.4 Composant images	21
6.2.5 Composant presentation.....	21
6.3 Collaboration entre les différentes classes de l'application	22
6.4 Diagramme d'activité.....	26
6.5 Diagramme de séquence.....	27
7. Réalisation.....	28

7.1	Utilisation des APIs Watson.....	28
7.1.1	Speech To Text.....	28
7.1.1.1	Explication du code la classe MySpeechToText.....	29
7.1.1.2	Les Limites de l'API « Speech To text »	30
7.1.2	Natural Language Understanding (NLU)	30
7.1.2.1	Explication du code la classe MyNaturalLanguageUnderstanding	32
7.1.2.2	Les Limites de l'API « Natural Language Understanding »	33
7.1.3	Watson conversation	35
7.1.3.1	Explication du code.....	38
7.1.3.2	Les Limites de l'API « Conversation service »	39
7.2	Utilisation de EWS JAVA API.....	39
7.2.1	Etablir la connexion entre l'application et le serveur Exchange.....	40
7.2.1.1	Explication du code de la classe ConnexionServeur	40
7.2.2	Chercher l'email des participants sur le serveur	41
7.2.2.1	Explication du code de la classe MyGetEmailParticipant	42
7.2.3	Chercher les disponibilités des participants sur le serveur	43
7.2.3.1	Explication du code.....	44
7.2.4	Créer un rendez-vous sur les agendas des participants.....	45
7.2.4.1	Explication du code.....	46
7.2.5	Les limites du SDK EWS-JAVA-API	47
8.	Démonstration avec SmartAgenda pour créer un rendez-vous.....	47
8.1	Type de texte valide sur SmartAgenda	48
8.2	Type de texte invalide sur SmartAgenda.....	51
9.	Les difficultés rencontrées durant ce travail	54
9.1	Les difficultés liées à l'utilisation des APIs Watson.....	54
9.1.1	La mauvaise qualité de l'API « Speech To text »	54
9.2	Les difficultés liées à l'utilisation de EWS-JAVA-API.....	56
10.	Conclusion	56
	Bibliographie.....	57
	Annexe 1 : Accès et utilisation des APIs Watson	58
	Annexe 2 : Utilisation de EWS-JAVA-API.....	89

Liste des tableaux

Tableau 1 : Besoins clés des utilisateurs	10
Tableau 2 : Carte CRC du composant « Interface »	13
Tableau 3 : Carte CRC du composant « Back-end system »	13
Tableau 4 : Carte CRC du composant « Application »	14
Tableau 5 : Carte CRC du composant « domaine »	20
Tableau 6 : Carte CRC du composant « servicesWatson »	20
Tableau 7 : Carte CRC du composant « servicesExchange »	21
Tableau 8 : Carte CRC du composant « images »	21
Tableau 9 : Carte CRC du composant « presentation »	22
Tableau 10 : Carte CRC de la classe SmartAgendaMain	23
Tableau 11 : Carte CRC de la classe SmartAgendaViewGetInfoRv	23
Tableau 12 : Carte CRC de la classe SmartAgendaViewLogin	24
Tableau 13 : Carte CRC de la classe SmartAgendaViewDisplayInfoRv	24
Tableau 14 : Carte CRC de la classe SmartAgendaViewSuggestionPlageHoraire	24
Tableau 15 : Carte CRC de la classe Participant	25
Tableau 16 : Carte CRC de la classe RecherchePlageHoraire	25
Tableau 17 : Carte CRC de la classe PlageHoraireSuggere	25

Liste des figures

Figure 1 : Liste des APIs Watson	11
Figure 2 : Architecture d'application Watson	13
Figure 3 : Architecture du use case « Identifier l'utilisateur »	16
Figure 4 : Architecture du use case « Créer un rendez-vous »	17
Figure 5 : Architecture de la solution	19
Figure 6 : Diagramme d'activité	26
Figure 7 : Diagramme de séquence	27
Figure 8 : Démo en ligne de l'API « NLU »	31
Figure 9 : Démo en ligne de l'API « NLU » avec des noms compliqués	34
Figure 10 : Démo 1 en ligne de l'API « NLU » avec des prénoms des personnes	34
Figure 11 : Démo 2 en ligne de l'API « NLU » avec des prénoms des personnes	35
Figure 12 : Démo 1 en ligne du service « conversation » créé sur Bluemix	36
Figure 13 : Démo 2 en ligne du service « conversation » créé sur Bluemix	36
Figure 14 : Démo 3 en ligne du service « conversation » créé sur Bluemix	37
Figure 15 : Démo 4 en ligne du service « conversation » créé sur Bluemix	39
Figure 16 : Ecran principal de SmartAgenda	47
Figure 17 : Démo Ecran principal de SmartAgenda	48
Figure 18 : Ecran de visualisation des infos du rendez-vous	49
Figure 19 : Ecran pop-up	49
Figure 20 : Ecran de création de rendez-vous	49
Figure 21 : Ecran de création de rendez-vous avec sélection	50
Figure 22 : Ecran pop-up de visualisation du rendez-vous à créer	50
Figure 23 : Ecran pop-up pour un rendez-vous créé	50
Figure 24 : Capture d'écran de mon agenda	51
Figure 25 : Démo avec date et heure invalide	51
Figure 26 : Ecran pop-up avec date et heure invalide	52
Figure 27 : Démo avec date invalide	52
Figure 28 : Ecran pop-up avec date invalide	52
Figure 29 : Ecran pop-up avec des formats invalides	53
Figure 30 : Ecran pop-up pour omission de date	53

Figure 31 : Ecran pop-up pour omission d'heure	53
Figure 32 : Ecran pop-up si aucun participant dans le texte	54
Figure 33 : Ecran pop-up si omission du lieu de rendez-vous	54
Figure 34 : Résultat obtenu avec l'API « Speech To text » de Watson en ligne	55
Figure 35 : Résultat obtenu avec SmartAgenda	55
Figure 36 : Page d'accueil pour créer un compte sur Bluemix	59
Figure 37 : Information de mon compte sur Bluemix	59
Figure 38 : Page d'accueil de mon compte sur Bluemix	60
Figure 39 : Sélection de l'API Watson « Conversation »	60
Figure 40 : Page de création du service « Conversation »	61
Figure 41 : Page de paramétrage de l'API Watson « Conversation »	61
Figure 42 : Clic sur « Launch tool » pour paramétrer « Conversation »	62
Figure 43 : Création d'un Workspace	62
Figure 44 : Configuration de notre Workspace	62
Figure 45 : Création d'un intent	63
Figure 46 : Configuration de l'intent	63
Figure 47 : Ajout d'un début de phrases dans l'intent	63
Figure 48 : Liste de nos intents	64
Figure 49 : Enregistrement de l'intent « meeting »	64
Figure 50 : Entraînement de la machine avec les nouveaux intents	65
Figure 51 : Ajout des entitie pour le dialogue	65
Figure 52 : Création d' entitie « lieu » pour le dialogue	66
Figure 53 : Configuration de l'entitie « lieu » pour le dialogue	66
Figure 54 : Activation du matching	67
Figure 55 : Enregistrement de l'entitie « lieu »	67
Figure 56 : Entitie « lieu » enregistré dans le système	67
Figure 57 : System entities	68
Figure 58 : Liste des entities du système	68
Figure 59 : Sélection de l'entitie time et date du système	69
Figure 60 : Activation de l'entitie time et date du système	69
Figure 61 : Construction d'un dialogue dans le système	69
Figure 62 : Créer un dialogue dans le système	70
Figure 63 : Créer un nouveau nœud	70
Figure 64 : Configuration du nœud	71
Figure 65 : Enregistrement de notre nœud	71
Figure 66 : Configuration du dialogue	72
Figure 67 : Configuration du dialogue	72
Figure 68 : Configuration du dialogue	73
Figure 69 : Liste des nœuds du dialogue	73
Figure 70 : Personnalisation du nœud « Tout le reste »	74
Figure 71 : Lancement du dialogue en ligne	74
Figure 72 : Page d'accueil du dialogue en ligne	75
Figure 73 : Question1	75
Figure 74 : Réponse du système à la question1	76
Figure 75 : Réponse du système à la question 2	76
Figure 76 : Réponse du système à la question 3	77
Figure 77 : Réponse du système à la question 4	77
Figure 78 : Réponse du système à la question 5	78
Figure 79 : Page de configuration du dialogue	79
Figure 80 : Liste de nos services Watson créés sur Bluemix	79
Figure 81 : Workspace de service « Conversation »	80
Figure 82 : Clic sur l'icône sélectionnée	80
Figure 83 : Clic sur « View details »	80
Figure 84 : Détails du Workspace	81
Figure 85 : Informations pour le service « Conversation »	82

Figure 86 : Données d'identification du service « Conversation »	82
Figure 87 : Les Credentials pour le service « Conversation »	82
Figure 88 : Sélection du service « Speech To text »	83
Figure 89 : Page de création du service « Speech To text »	83
Figure 90 : Page d'accueil du service « Speech To text »	84
Figure 91 : Informations du service « Speech To text »	84
Figure 92 : Données d'identification du service « Speech To text »	84
Figure 93 : Les Credentials pour le service « Speech To text »	85
Figure 94 : Sélection du service « NLU »	85
Figure 95 : Page de création du service « NLU »	86
Figure 96 : Page d'accueil du service « NLU »	86
Figure 97 : Informations du service « NLU »	86
Figure 98 : Données d'identification du service « NLU »	87
Figure 99 : Les Credentials pour le service « NLU »	87
Figure 100 : Configuration du SDK des APIs Watson	88
Figure 101 : Configuration du SDK de ews-java-api	89

1. Introduction

Le groupe de travail sur l'industrie 4.0, basé à L'OPI avait exprimé à travers Monsieur Philippe Dugerdil professeur HES son désir de disposer d'un outil numérique permettant d'améliorer ses processus collaboratifs qui pourrait intéresser les étudiants de la HEG dans le cadre de leur travail de Bachelor.

C'est ainsi que j'ai répondu à cet appel afin de réaliser cette application durant mon travail de diplôme à la HEG sous la supervision de Monsieur Philippe Dugerdil.

L'objectif de ce travail est de trouver une solution efficace pour faciliter la collaboration entre les membres du groupe de travail sur l'industrie 4.0. Une réunion est alors planifiée avec le(s) mandant(s) pour définir le périmètre du projet dans le but de faire les spécifications de l'application. C'est au cours de ces échanges qu'est né SmartAgenda. Cet outil est une application desktop permettant la prise de rendez-vous automatique de manière vocale entre collègues de travail.

Pour réaliser cette application, nous avons utilisé deux technologies très présentes aujourd'hui sur le marché à savoir Watson et Exchange.

Nous allons dans la première partie de ce travail exposer le problème à résoudre par cette application.

Nous présenterons dans la deuxième partie du document une introduction aux APIs Watson et l'API Exchange Server utilisé dans le développement de cet outil.

La troisième partie du document sera consacrée à l'architecture de la solution proposée pour réaliser cette application.

2. Présentation du problème à résoudre

Le défi de l'informatique a été toujours d'améliorer les processus de travail entre les humains.

L'application que nous devons développer s'inscrit dans cette dynamique.

Le tableau suivant regroupe les besoins des utilisateurs auxquels les fonctionnalités de l'application doivent répondre

Tableau 1 : Besoins clés des utilisateurs

N°	Besoin(métier)	Priorité	Concerne	Solutions actuelles	Solutions proposées
1	Se connecter sur SmartAgenda	Elevé	Les utilisateurs		
2	Effectuer une demande vocale d'organisation de rendez-vous avec un groupe de personnes dont les agendas sont gérés par le même serveur	Elevé	Les utilisateurs		

Pour répondre à ces besoins, nous allons utiliser les APIs Watson et Exchange Server.

3. Introduction aux APIs Watson

Aujourd'hui le terme AI (Artificielle Intelligence) est omniprésent dans le milieu informatique.

On peut parmi les entreprises les plus influentes dans ce domaine, citer IBM à travers son produit Watson, Amazon, Google, Microsoft.

Tout d'abord voici une définition du terme « traitement cognitif » :

« Les processus cognitifs sont les différents modes à travers lesquels un système traite l'information en y répondant par une action. Deux types de système capables de réaliser des processus cognitifs peuvent se distinguer :

les systèmes naturels : un neurone, un réseau de neurone, un cerveau (humain ou animal), un groupe d'individus (poissons, fourmis), etc.

les systèmes artificiels : réseau de neurones artificiels, système expert, etc. »
(WIKIPEDIA)

3.1 Watson IBM c'est quoi ?

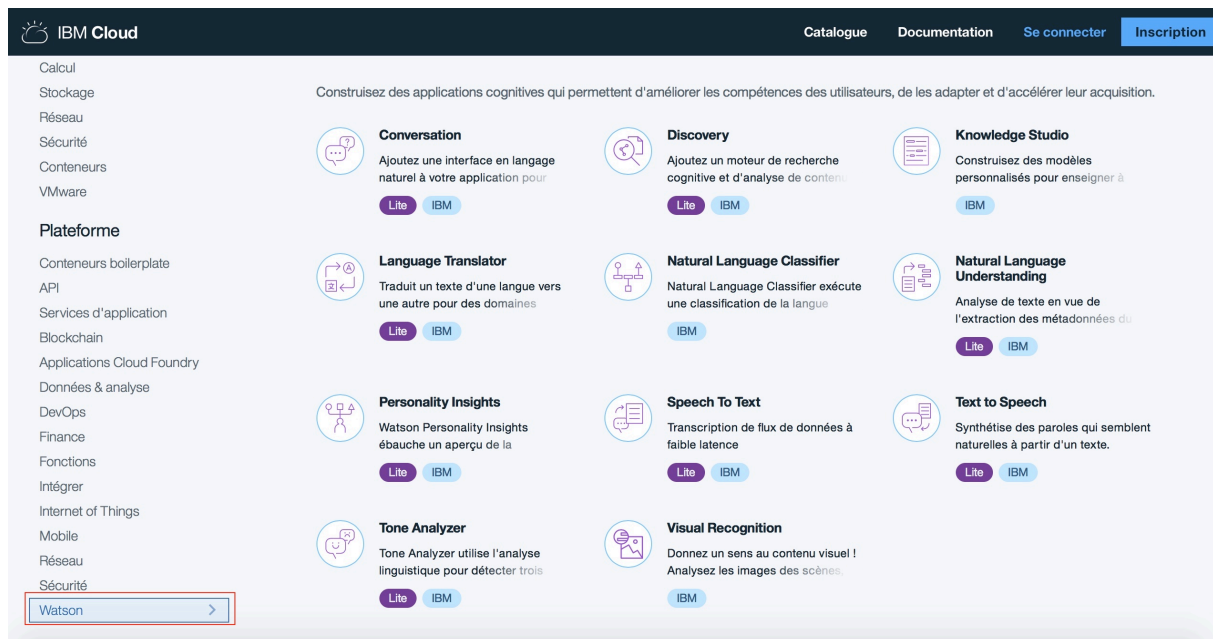
Pour répondre à cette interrogation, je vous propose cette définition proposée par IBM :

« Watson est un moteur analytique efficace qui rassemble de nombreuses sources d'informations en temps réel, en tire des données exploitables et leur attribue un niveau de fiabilité et de confiance. »
(IBM Watson)

3.2 Présentation générale des APIs Watson

IBM à travers sa plateforme Watson, met à la disposition des développeurs, une palette d'APIs cognitifs pour concevoir des applications pouvant interagir avec l'humain.

Figure 1 : Liste des APIs Watson



(IBM Cloud)

3.2.1 API Watson Document Conversion

Cet API Watson est utilisé pour faire la conversion d'un document dans un autre format (par exemple conversion d'un document WORD en HTML). Les inputs de l'API peuvent être un document HTML, WORD ou PDF et les outputs du JSON, du HTML normalisé ou du plain text. Pour l'instant ce service est disponible sur 8 langues qui sont : l'anglais, le français, l'allemand, le japonais, l'italien, le brésilien, le portugais et l'espagnol.

Ce service peut être utilisé par d'autres APIs Watson tel que le service « Watson Retrieve and Rank Service » dont le but est de retrouver une information pertinente à partir d'une collection de documents.

3.2.2 API Watson Tone Analyzer

Il permet de faire une étude linguistique d'un texte afin de trouver les sons émotionnels, sociaux et linguistiques. Ce service est destiné surtout aux entreprises qui veulent contrôler leurs échanges écrits entre leurs clients. Le service prend en paramètre soit un fichier JSON ou un texte brut ou un document HTML d'une taille pouvant aller jusqu'à 1000 phrases (128ko de taille).

3.2.3 Speech to Text

Speech to Text est utilisé pour transcrire un fichier audio en texte ou de faire une transcription sous forme de texte en continu, au fur et à mesure que l'on parle à travers un microphone. On peut utiliser ce service à travers soit un interface WebSocket, soit un interface http REST, soit un interface http asynchrone. Il existe également une interface pour créer des modèles personnalisés de langue mais ce service est disponible uniquement en anglais, en japonais et en espagnol.

Dans ce travail, nous avons utilisé ce service à travers l'interface http REST pour la transcription des informations vocales en texte.

3.2.4 Text to Speech

Ce service est l'inverse du service « Speech to Text ». Il permet la conversion du texte en audio selon plusieurs formats dont MP3, FLAC, WAV, et WebM. Il supporte plusieurs langues dont le français et peut être utilisé via deux interfaces différents : interface http REST et interface WebSocket. Ce service peut être aussi customisé avec des prononciations spécifiques mais cela n'est disponible pour l'instant qu'en anglais et japonais.

3.2.5 Language Translator

C'est un service qui est conçu pour traduire du texte vers un langage cible. Il est disponible sur plusieurs langues dont le français et peut être personnalisé avec des termes spécifiques.

3.2.6 Natural Language Understanding

Le service Natural Language Understanding (compréhension de la langue naturelle) regroupe plusieurs APIs afin de fournir une analyse du texte sur la base de la langue naturelle. Cet API est capable de détecter automatiquement la langue du texte passé en paramètre.

L'API « Natural Language Understanding » est utilisé pour retrouver dans un texte des termes suivants : catégories, concepts, émotions, entités, mots-clés, métadonnées, relations, rôles sémantiques et sentiment.

3.2.7 API Watson Conversation

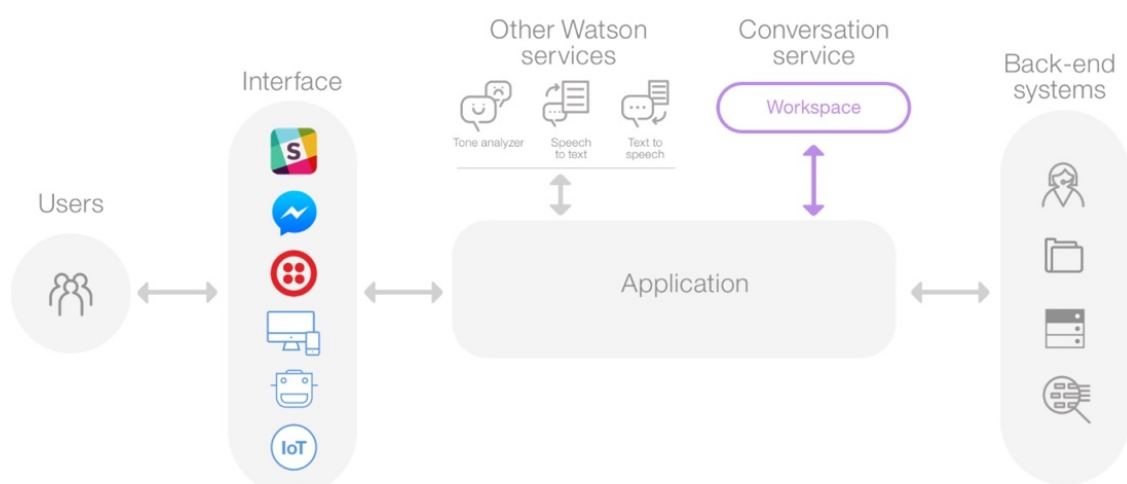
Ce service permet de créer des applications qui peuvent tenir une conversation avec l'humain en utilisant l'API Natural Language Understanding et les outils intégrés pour permettre à la machine d'apprendre toute seule. Ce service, une fois entraîné, pourra prendre des décisions tout seul sur la base des informations reçues et fournir des réponses cohérentes aux utilisateurs.

Watson fournit également d'autres APIs cognitifs tels que « Visual Recognition » pour reconnaître des images, « Personality Insights » pour fournir des informations sur la personnalité d'une personne à travers un texte écrit, et « Natural Language Classifier ».

3.3 Exemple d'architecture des applications Watson

Dans cette partie, nous allons présenter et expliquer l'architecture suivante utilisant certains APIs Watson.

Figure 2 : Architecture d'application Watson



(IBM Cloud Docs)

Pour expliquer cette architecture, nous allons utiliser les cartes CRC de R. Wirfs-brock.

Tableau 2 : Carte CRC du composant « Interface »

« composant » Interface	
Responsabilité	Collaboration
Présenter les boîtes de dialogue avec les Users	
Recueillir les informations du dialogue : Soit à travers un champ de saisie Soit à travers un micro ou un moyen d'interaction avec un humain	
Présenter les réponses du système aux Users	

Tableau 3 : Carte CRC du composant « Back-end system »

« composant » Back-end systems	
Responsabilité	Collaboration
Recevoir les demandes à traiter	
Implanter les besoins métiers pour traiter les demandes des Users et retourner les réponses	

Tableau 4 : Carte CRC du composant « Application »

« composant » Application	
Responsabilité	Collaboration
Obtenir les informations du dialogue à travers l'interface de dialogue avec les Users	« composant » Interface
Interroger les services de Watson pour formater les informations obtenues des Users	Soit avec API « Toner Analyzer » de Watson pour traduire du texte vers un langage cible Soit avec API « Speech To text » pour transcrire la voix en texte Soit avec API « Text To speech » pour convertir un texte en audio
Transmettre les informations formatées aux services de Watson conversation pour obtenir la décision	API « Watson conversion »
Interroger les services du Back-end systems en lui fournissant la décision de Watson conversation pour obtenir la réponse définitive du système	« composant » Back-end systems

Le « Workspace » est un conteneur créé sur Bluemix d'IBM qui permet de définir les flux de conversation avec le composant Application. Il permet également d'identifier le service sur Bluemix. Autrement dit il a un id qui est unique sur Bluemix qui nous permet d'appeler le service à distant. Quand on va utiliser le service « Conversation » créé sur Bluemix, on va référencer l'id du Workspace

Le Back-end system regroupe les services que l'application fait appel pour répondre aux besoins des utilisateurs.

4. Présentation de l'API Microsoft Exchange utilisé

Microsoft à travers sa plateforme de développement met à la disposition des développeurs deux types d'APIs pour accéder aux ressources de Microsoft Exchange Server à travers leurs applications.

Il s'agit de l'API REST Office 365 nouvellement utilisés à l'arrivée de Office 2013 et EWS¹ Managed API crée depuis 2010.

Pourquoi n'avons-nous pas utilisé l'API REST Office 365 pour développer cette application ?

¹ Echange Web Services

Comme nous l'avons expliqué précédemment, SmartAgenda est conçu pour créer des rendez-vous selon les disponibilités des personnes concernées. Pour que cela soit possible avec les APIs REST Office 365, il faut que les boîtes aux lettres Office 365 soient gérées dans Microsoft Azure AD qui est le cloud de Microsoft.

Donc on ne pouvait pas utiliser cette solution pour développer la version test de cette application car nous avons décidé que notre cible soit un serveur Exchange propriétaire.

C'est pourquoi dans ce travail, nous avons utilisé EWS Managed API pour réaliser la version test de SmartAgenda.

4.1 EWS c'est quoi ?

Pour répondre à cette question, je vous propose cette définition proposée par Microsoft :

« EWS est un service complet que vos applications peuvent utiliser pour accéder à presque toutes les informations stockées dans une boîte aux lettres Exchange Online, Exchange Online dans le cadre d'Office 365 ou Exchange en local. EWS utilise les protocoles web standard pour fournir un accès à un serveur Exchange. » (Microsoft 2017)

Ce service est développé sur plusieurs langages de programmation mais on s'est intéressé à celui développé en Java. Il s'agit de l'API ews-java-api.

Il fournit toutes les fonctionnalités nécessaires pour manipuler les ressources disponibles dans Exchange Server.

Pour accéder et manipuler les ressources du serveur, l'API a besoin des identifiants valides de l'utilisateur pour établir l'authentification.

5. Architecture de la solution

Cette partie du document est consacrée à la définition du périmètre de l'application, à la spécification, à la conception et à la réalisation.

5.1 Périmètre de l'outil

L'outil permettra à l'utilisateur d'effectuer une demande vocale d'organisation de rendez-vous avec un groupe de personnes dont les agendas sont gérés par le même serveur. Le système devra interpréter la demande vocale, identifier les participants et contraintes temporelles, consulter les agendas des personnes concernées sur le serveur et proposer un rendez-vous à la meilleure date.

L'architecture de la solution devra permettre d'adapter l'outil à différents serveurs mail/agenda. La solution sera testée avec Microsoft Exchange.

Ce projet est issu d'un groupe de travail sur l'industrie 4.0, organisé par l'OPI (contacts : MM Hilfiker et Mesnier). La solution, une fois opérationnelle pourrait être testée aux SIG (contact : M. Junet).

5.2 Spécification

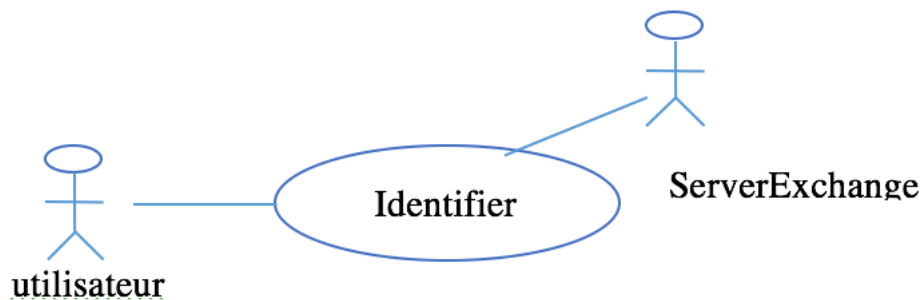
L'outil offre les fonctions suivantes :

- Identifier l'utilisateur autorisé.
- Créer un rendez-vous entre collègues de travail de manière vocale ou textuelle

5.2.1 Traitement des uses case

A. **Nom** : Identifier l'utilisateur

Figure 3 : Architecture du use case « Identifier l'utilisateur »



Acteurs : secondaire : Server des données des utilisateurs

Déclencheur : l'utilisateur ouvre l'application

Flot de base

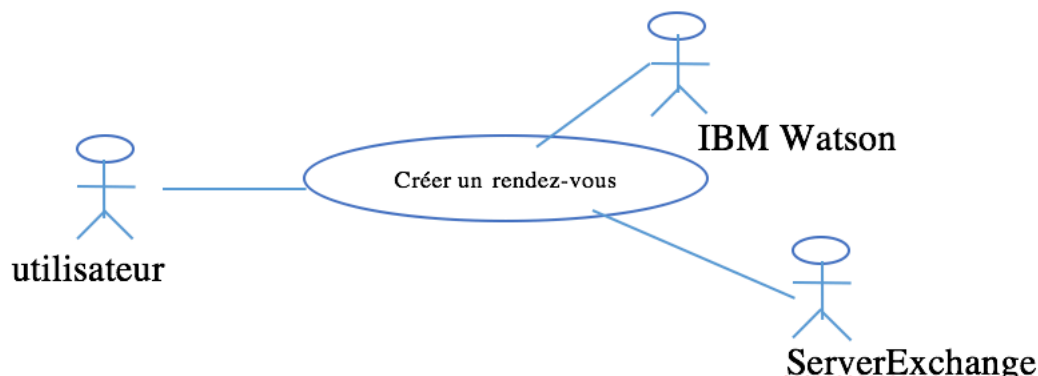
1. Le système affiche la page de connexion
2. L'utilisateur saisit son nom d'utilisateur et son mot de passe
3. L'utilisateur clique sur le bouton « Connexion »
4. Le système vérifie que l'utilisateur existe dans la base de donnée et que son nom d'utilisateur et son mot de passe sont corrects
5. Le système affiche la page principale
6. Fin du use case

Flot alternatif

- 6a. Le nom d'utilisateur ou le mot de passe est incorrect
 - 6a1. Le système affiche une fenêtre pop-up avec un message d'erreur
 - 6a2. L'utilisateur clique le bouton « Ok » de la fenêtre de pop-up
 - 6a3. Retour au point 1.

B. Nom : Créer un rendez-vous entre collègues de travail par saisie vocale ou texte

Figure 4 : Architecture du use case « Créer un rendez-vous »



Acteurs : principal : utilisateur, secondaires : ServerExchange, IBM Watson

Prérequis : l'utilisateur s'est déjà connecté sur le système

Flot de base

1. L'utilisateur clique sur le micro
2. Le système vérifie que l'ordinateur est équipé d'un microphone et démarre l'enregistrement
3. L'utilisateur fournit de manière vocale les informations² du rendez-vous.
4. Le système envoie les informations vocales aux services Watson « Speech To Text » pour transcrire l'enregistrement vocal en texte.
5. L'utilisateur reclique sur le micro.
6. Le système arrête l'enregistrement et récupère le texte correspondant à l'enregistrement vocal
7. Le système affiche le texte sur l'écran
8. L'utilisateur clique sur le bouton « Extraire les données du rendez-vous »
9. Le système envoie le texte aux services Watson « NLU » pour extraire les noms et prénoms des participants et « Conversation service » pour extraire les contraintes temporelles (date et heure) et lieu du rendez-vous.
10. Le système vérifie si chaque participant est valide en envoyant aux services Exchange le nom et le prénom du participant et récupère son email sur le serveur
11. Le système affiche la date, l'heure et la liste des participants au rendez-vous
12. L'utilisateur clique sur le bouton « Chercher Disponibilités »
13. Le système envoie aux services Exchanges la date, l'heure et la liste des participants du rendez-vous pour trouver les plages horaires disponibles

² Contraintes temporelles (date et heure) + la liste des participants et le lieu du rendez-vous

14. Le système affiche la liste des disponibilités communes et un niveau de qualité (Excellent ou Good)
15. L'utilisateur sélectionne une proposition de rendez-vous
16. L'utilisateur clique sur le bouton « Créer un rendez-vous »
17. Le système affiche une fenêtre pop-up avec la date, l'heure, le lieu et la liste des participants et demande à l'utilisateur s'il veut créer ce rendez-vous.
18. L'utilisateur clique le bouton « Oui » de la fenêtre de pop-up
19. Le système envoie les données aux services Exchange pour créer le rendez-vous et notifier tous les participants
20. Le système affiche une fenêtre pop-up en disant qu'un nouveau rendez-vous est créé

Flot alternatif

- 1a. L'utilisateur sélectionne le radio bouton « Saisie Texte »
 - 1a1. L'utilisateur saisit les informations³ du rendez-vous.
 - 1a2. Retour au point 8
- 2a. L'ordinateur n'est pas équipé d'un microphone
 - 2a1. Le système sélectionne l'option « Saisie Texte » et affiche un message disant à l'utilisateur de choisir l'option « Saisie Texte »
 - 2a2. Retour au point 1a1
- 9a. L'utilisateur a oublié de fournir la date ou l'heure ou lieu du rendez-vous
 - 9a1. Le système affiche une fenêtre pop-up avec un message demandant à l'utilisateur de fournir l'information manquante.
 - 9a2. L'utilisateur clique le bouton « Ok » de la fenêtre de pop-up
 - 9a3. Retour au point 7
- 9b. Le texte du rendez-vous est ambiguë
 - 9b1. Le système affiche une fenêtre pop-up avec un message demandant à l'utilisateur de corriger le texte.
 - 9b2. L'utilisateur clique le bouton « Ok » de la fenêtre de pop-up
 - 9b3. Retour au point 7
- 10a. Le participant n'est pas enregistré sur le serveur Exchange
 - 10a1. Le système affiche une fenêtre pop-up avec un message disant que ce participant (avec son nom et prénom) est inconnu du système.
 - 10a2. L'utilisateur clique le bouton « Ok » de la fenêtre de pop-up
 - 10a3. Retour au point 7
- 15a. L'utilisateur sélectionne une proposition de rendez-vous et saisit une information optionnelle pour le rendez-vous

³ Contraintes temporelles (date et heure) + Liste des participants + lieu du rendez-vous

15a1. Retour au point 16

18a. L'utilisateur clique sur le bouton « Non » de la fenêtre de pop-up

18a1. Retour au point 14

18b. L'utilisateur clique sur le bouton « Annuler » de la fenêtre de pop-up

18b1. Fin du use case

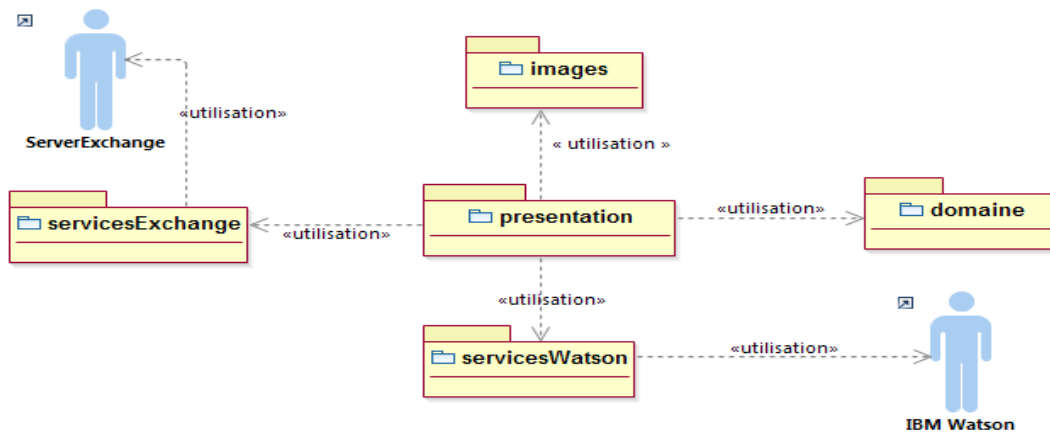
6. Conception

Nous allons finalement dans cette partie, présenter et expliquer l'architecture finale de la solution et les différentes collaborations entre les classes.

6.1 Présentation de l'architecture de la solution

L'application SmartAgenda est structurée de la manière suivante :

Figure 5 : Architecture de la solution



6.2 Explication de l'architecture de la solution

Nous allons dans partie, utiliser les cartes CRC de R. Wirfs-brock pour expliquer cette architecture, avant de présenter le diagrammes d'activité et séquence qui découlent de cette analyse.

6.2.1 Composant domaine

Il regroupe tous nos objets du domaine manipulés de cette application. Ce composant contient les classes suivantes :

- Participant
- PlageHoraireSuggeree
- RecherchePlageHoraire

Tableau 5 : Carte CRC du composant « domaine »

Composant « domaine »	
Responsabilité	Collaboration
Gérer les objets du domaine et retourner leurs informations en cas de besoin	

6.2.2 Composant servicesWatson

Il permet de recourir aux APIs Watson pour la transcription de la voix, la récupération de la liste des participants et les contraintes temporelles et le lieu du rendez-vous. Ce composant contient les classes suivantes :

- MySpeechToText
- MyNaturalLanguageUnderstanding
- MyWatsonConversation

Tableau 6 : Carte CRC du composant « servicesWatson »

Composant « servicesWatson »	
Responsabilité	Collaboration
Transcrire la voix en texte	IBM Watson (API Speech To Text)
Retourner le texte brute de la transcription	
Récupérer une liste de personnes contenus dans le texte brut	IBM Watson (API Natural Language Understanding)
Retourner cette liste de participants	
Récupérer les contraintes temporelles et le lieu du rendez-vous contenus dans le texte brut	IBM Watson (API Conversation service)
Retourner les données temporelles et le lieu du rendez-vous	

6.2.3 Composant servicesExchange

Ce composant permet de recourir aux APIs EWS afin de vérifier l'authentification de l'utilisateur et de retourner une instance de ExchangeService. Il permet aussi de centraliser des données liées à la recherche de disponibilités et de création de rendez-vous ainsi que l'obtention des emails des participants et les disponibilités des participants, la création de rendez-vous et l'envoi de notification aux participants. Ce package contient les classes suivantes :

- ConnexionServeur
- MyGetEmailParticipant

- MyGetPlagesHoraires
- MyAppointment
- Data

Tableau 7 : Carte CRC du composant « servicesExchange »

Composant « servicesExchange »	
Responsabilité	Collaboration
Etablir la connexion avec le serveur Exchange	ServerExchange(ExchangeService)
Instancier et retourner le service Exchange au contrôleur	
Obtenir l'email d'un participant	ServerExchange(ExchangeService)
Retourner l'email du participant	
Chercher les plages horaires disponibles	ServerExchange(ExchangeService)
Retourner les plages horaires disponibles	
Créer un rendez-vous et envoyer une notification aux participants	ServerExchange(ExchangeService)

6.2.4 Composant images

Son rôle dans cette application est de stocker uniquement les deux fichiers jpeg représentant l'image du micro.

Tableau 8 : Carte CRC du composant « images »

Composant « images »	
Responsabilité	Collaboration
Stocker les images du micro	

6.2.5 Composant presentation

Il regroupe les différentes vues de l'application et assure le rôle de contrôleur. Ce package contient les classes suivantes :

- MyFocusTraversalPolicy : Cette classe permet gérer la tabulation sur les différents écrans de l'application
- SmartAgendaMain
- SmartAgendaViewLogin
- SmartAgendaViewGetInfoRv
- SmartAgendaViewDisplayInfoRv
- SmartAgendaViewSuggestionPlageHoraire

Tableau 9 : Carte CRC du composant « presentation »

Composant « presentation »	
Responsabilité	Collaboration
Lancer l'afficher de la page de connexion	
Recueillir les identifications de l'utilisateur	
Vérifier l'authentification l'utilisateur	« composant » servicesExchange
Etablir la connexion avec le serveur, Obtenir l'instance de la classe ExchangeService et la centraliser	« composant » servicesExchange
Obtenir l'image du micro	« composant » images
Lancer l'affichage de la page de création de rendez-vous	
Recueillir les informations du rendez-vous <ul style="list-style-type: none"> - Liste des participants - Contraintes temporelles 	« composant » servicesExchange « composant » servicesWatson « composant » domaine
Lancer la page d'affichage de la liste des participants au rendez-vous	
Lancer l'affichage de la page de proposition de plages horaires pour un rendez-vous	
Obtenir une liste de proposition de plages horaires	« composant » servicesExchange
Recueillir la sélection de l'utilisateur	
Demander la confirmation de créer un rendez-vous	
Créer un rendez-vous et envoyer les notifications aux participants	« composant » servicesExchange

6.3 Collaboration entre les différentes classes de l'application

Cette partie est consacrée à l'explication des collaborations entre les différentes classes qui implémentent les fonctionnalités de l'application SmartAgenda. Pour cela, nous utilisons les cartes CRC de R. Wirfs-brock.

Tableau 10 : Carte CRC de la classe SmartAgendaMain

SmartAgendaMain	
Responsabilité	Collaboration
Démarrer l'application en instanciant la classe SmartAgendaViewLogin	SmartAgendaViewLogin

Tableau 11 : Carte CRC de la classe SmartAgendaViewGetInfoRv

SmartAgendaViewGetInfoRv	
Responsabilité	Collaboration
Afficher la fenêtre de création de rendez-vous	
Vérifier que l'ordinateur est équipé d'un microphone	
Recueillir l'enregistrement audio du rendez-vous	
Instancier MySpeechToText pour obtenir la transcription de l'audio	MySpeechToText
Afficher le texte de correspondant à l'audio	
Ou bien permettre à l'utilisateur de saisir directement le texte du rendez-vous	
Instancier MyNaturalLanguageUnderstanding pour obtenir la liste des participants	MyNaturalLanguageUnderstanding
Instancier MyGetEmailParticipant pour vérifier si les participants existent sur le serveur et obtenir leur adresse email	MyGetEmailParticipant
Instancier MyWatsonConversation pour obtenir les contraintes temporelles (date et heure) et le lieu du rendez-vous	MyWatsonConversation
Instancier SmartAgendaViewDisplayInfoRv pour afficher les informations du rendez-vous (date, heure et liste des participants)	SmartAgendaViewDisplayInfoRv

Tableau 12 : Carte CRC de la classe SmartAgendaViewLogin

SmartAgendaViewLogin	
Responsabilité	Collaboration
Afficher l'écran de login	
Recueillir les identifiants de l'utilisateur	
Instancier ConnexionServeur pour valider la connexion	ConnexionServeur
Instancier SmartAgendaViewGetInfoRv pour afficher la fenêtre de création de rendez-vous	SmartAgendaViewGetInfoRv

Tableau 13 : Carte CRC de la classe SmartAgendaViewDisplayInfoRv

SmartAgendaViewDisplayInfoRv	
Responsabilité	Collaboration
Afficher la fenêtre des informations du rendez-vous	
Instancier SmartAgendaViewSuggestionPlageHoraire pour afficher les disponibilités	SmartAgendaViewSuggestionPlageHoraire

Tableau 14 : Carte CRC de la classe SmartAgendaViewSuggestionPlageHoraire

SmartAgendaViewSuggestionPlageHoraire	
Responsabilité	Collaboration
Afficher la fenêtre des suggestions de rendez-vous	
Instancier RecherchePlageHoraire	RecherchePlageHoraire
Instancier MyGetPlagesHoraires pour obtenir les disponibilités communes des participants	MyGetPlagesHoraires
Instancier RecherchePlageHoraire pour formater le résultat de la recherche et l'afficher sur l'écran	RecherchePlageHoraire
Recueillir la sélection d'une proposition de l'utilisateur	
Instancier MyAppointement pour créer un rendez-vous selon la proposition sélectionnée	MyAppointement

Tableau 15 : Carte CRC de la classe Participant

Participant	
Responsabilité	Collaboration
Connaître les données du participant et les retourner en cas de besoin	

Tableau 16 : Carte CRC de la classe RecherchePlageHoraire

RecherchePlageHoraire	
Responsabilité	Collaboration
Connaître les données d'une plage horaire à rechercher sur le serveur et les retourner en cas de besoin	

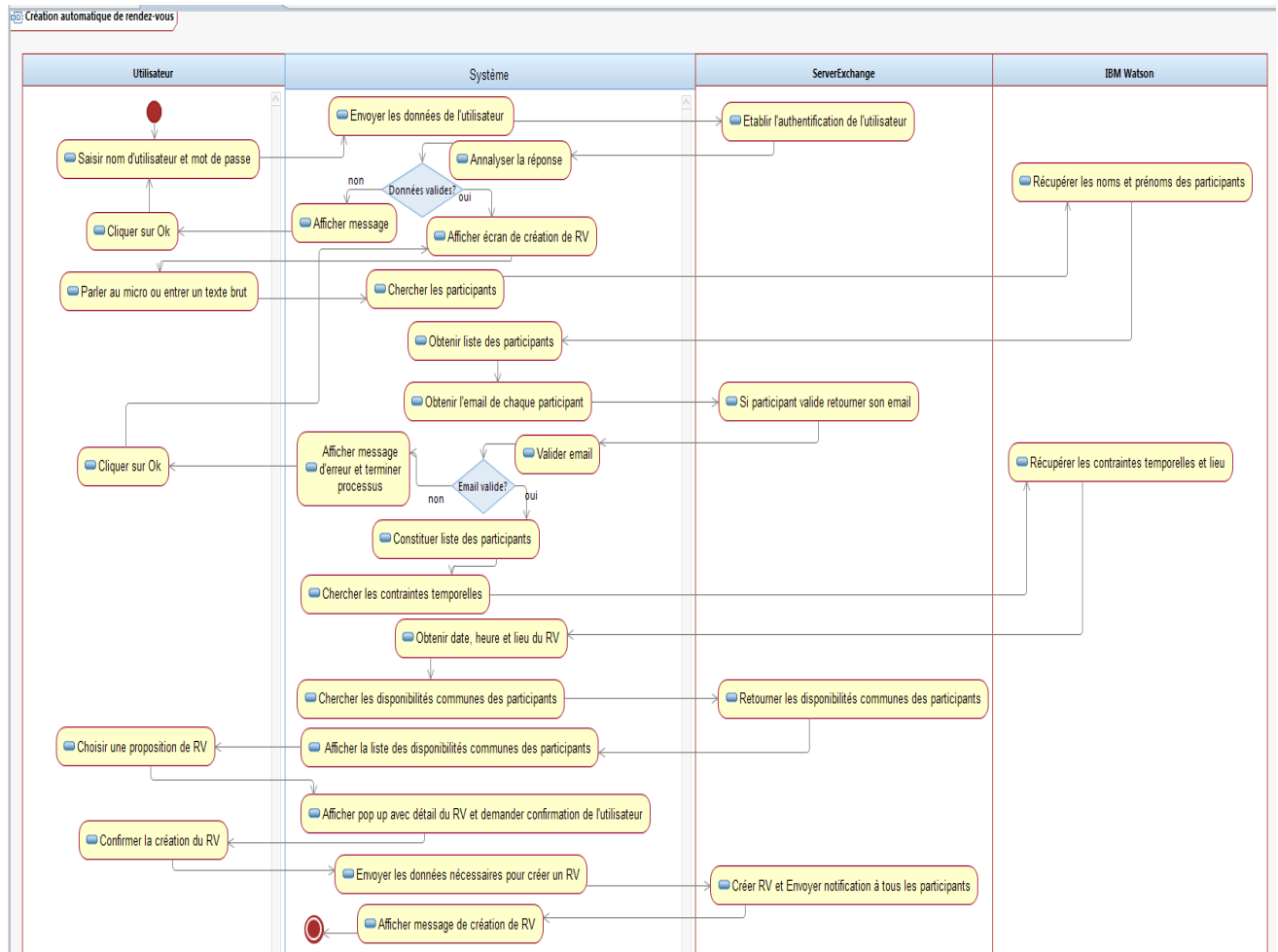
Tableau 17 : Carte CRC de la classe PlageHoraireSuggere

PlageHoraireSuggere	
Responsabilité	Collaboration
Connaître les données d'une plage horaire suggérée et les retourner en cas de besoin	

6.4 Diagramme d'activité

Ce diagramme décrit le processus complet de création de rendez-vous sur SmartAgenda.

Figure 6 : Diagramme d'activité



6.5 Diagramme de séquence

Ce diagramme décrit la séquence des flux de message entre nos différents objets pour aboutir à une création de rendez-vous sur SmartAgenda.

Figure 7 : Diagramme de séquence



7. Réalisation

La réalisation de cette application a nécessité l'utilisation des APIs cognitifs fournis par IBM Watson pour concevoir la partie « intelligente » de l'outil et EWS fourni par Microsoft pour la recherche de disponibilité et la création de rendez-vous.

7.1 Utilisation des APIs Watson

Dans ce travail, nous avons fait appel à trois services Watson pour réaliser l'interaction entre l'utilisateur et l'application de manière « intelligente ». Ces trois services sont :

- Speech To Text
- Natural Language Understanding
- Conversation service

7.1.1 Speech To Text

Nous avons utilisé cet API pour mettre à l'utilisateur de d'interagir avec l'outil de manière vocale à travers un micro.

Le rôle de ce service est de récupérer l'enregistrement audio de l'utilisateur et de le transformer en texte.

MySpeechTotext est la classe qui implémente l'API « Speech To Text » de Watson. Elle est codée de la façon suivante :

```
1 package servicesWatson;
2
3 import java.util.ArrayList;
4
5 /**
6  * Réalisé dans le cadre de mon Travail de Bachelor 2017
7  *
8  * Classe permettant de transcrire l'enregistrement vocal en texte et de retourner ce texte
9  *
10  * @author Lo Omar - HEG-Genève
11  * @version 1.0
12  */
13
14 public class MySpeechToText {
15
16     private static MySpeechToText uniqueInstance = new MySpeechToText();
17
18     /**
19      * Identifiants de mon service sur Bluemix pour avoir accès à l'API Speech
20      * to text
21      */
22     private static final String MY_USERNAME = "5c557b35-ce35-46e8-9197-82cec8ce2ed3";
23     private static final String MY_PASSWORD = "c4BNV7vSqxDI";
24     private TargetDataLine line;
25     private int sampleRate = 48000;
26     private ArrayList<String> aList;
27
28     /** Constructeur */
29     private MySpeechToText() {}
30
31     public static MySpeechToText getUniqueInstance() {
32         return uniqueInstance;
33     }
34 }
```

```

50 public void startTranscription() {
51     try {
52         SpeechToText service = new SpeechToText();
53         service.setUsernameAndPassword(MY_USERNAME, MY_PASSWORD);
54
55         AudioFormat format = new AudioFormat(sampleRate, 16, 1, true, false);
56         DataLine.Info info = new DataLine.Info(TargetDataLine.class, format);
57
58         line = (TargetDataLine) AudioSystem.getLine(info);
59         line.open(format);
60         line.start();
61
62         AudioInputStream audio = new AudioInputStream(line);
63         RecognizeOptions options = new RecognizeOptions.Builder().continuous(true).interimResults(true)
64             .timestamps(true).wordConfidence(true).model("fr-FR_BroadbandModel")
65             .inactivityTimeout(5) // Arrêt l'enregistrement après 5 secondes de silence
66             .contentType(HttpMediaType.AUDIO_RAW + "; rate=" + sampleRate).smartFormatting(true)
67             .keywords(new String[] { "créer", "rendez-vous", "lundi", "mardi", "mercredi", "matin",
68                 "après-midi" })
69             .keywordsThreshold(0.5).build();
70         alist = new ArrayList<String>();
71         service.recognizeUsingWebSocket(audio, options, new BaseRecognizeCallback() {
72             @Override
73             public void onTranscription(SpeechResults speechResults) {
74                 List<Transcript> rslts = speechResults.getResults();
75                 if (rslts != null && !rslts.isEmpty()) {
76                     String text = rslts.get(0).getAlternatives().get(0).getTranscript();
77                     alist.add(text + "\n");
78                 }
79             }
80         });
81     } catch (Exception e) {
82         System.out.println("MySpeechToText.getString(): " + e.getMessage());
83         e.printStackTrace();
84     }
85 }
86
87 // startTranscription
88
89 public String getString() {
90     try {
91         Thread.sleep(1 * 1000);
92         // closing the WebSockets underlying InputStream will close the
93         // WebSocket itself.
94         line.stop();
95         line.close();
96     } catch (Exception e) {
97         // TODO: handle exception
98         System.out.println("MySpeechToText.close() " + e.getMessage());
99         e.printStackTrace();
100     }
101     int index = alist.size()-1;
102     return alist.get(index);
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }

```

7.1.1.1 Explication du code la classe MySpeechToText

MySpeechToText est la classe qui utilise le service « Speech To Text » de Watson. C'est un singleton car nous voulons éviter d'instancier cette classe à chaque fois qu'on veut appeler l'API « Speech To Text » de Watson.

La méthode startTranscription permet de démarrer le service « Speech To Text » de Watson avec l'instanciation suivante :

```

SpeechToText service = new SpeechToText();
service.setUsernameAndPassword(MY_USERNAME, MY_PASSWORD);

```

Après l'instanciation, on a besoin de setter le username et le password obtenus au moment de la création du service sur notre compte Bluemix afin de permettre à l'instance de se connecter au bon service.

La classe RecognizeOptions permet de paramétrer le service par exemple en choisissant dans quelle langue on peut utiliser le service à travers la méthode model. On peut aussi injecter des mots clés dans le service pour l'entraîner à reconnaître certains mots dans l'audio à transcrire.

La méthode recognizeUsingWebSocket du service avec en paramètre notre audio, une instance de RecognizeOptions et une instance de BaseRecognizeCallBack nous retourne le texte correspondant à l'audio. La transcription se fait au fur et à mesure que l'utilisateur parle.

La méthode getString de notre classe permet de stopper l'enregistrement audio et le service « Speech To Text » et retourner la transcription finale de l'audio qui sera affichée dans l'écran de l'utilisateur.

La documentation complète de l'API « Speech To Text » de Watson disponible à l'adresse : <https://www.ibm.com/watson/developercloud/speech-to-text/api/v1/>

7.1.1.2 Les Limites de l'API « Speech To text »

L'objectif en utilisant ce service était d'avoir une transcription fidèle de l'enregistrement vocal sans perte de données. Mais malheureusement cet API n'est pas très performant pour répondre à notre objectif.

La version de ce service est incapable de reconnaître les dates et de fournir le bon format. Seule la version disponible en anglais et japonais le permet. C'est pourquoi il nous était impossible d'extraire les contraintes temporelles (date et heure) dans le texte transcrit car ces dernières étaient mal formatées par l'API.

7.1.2 Natural Language Understanding (NLU)

Pour extraire la liste des participants à la réunion dans le texte brut fourni par l'utilisateur, nous avons utilisé l'API « Natural Language Understanding » de Watson.

En effet ce service est capable de comprendre un texte et de retrouver certaines données comme les mots, les entités (personne, lieu, Compagnie etc.).

Cette classe retourne soit un arrayList contenant la liste des participants trouvés dans le texte, soit arrayList vide si aucun nom de personne n'est détecté dans le texte.

On peut tester ce service en ligne sur l'adresse suivante : https://natural-language-understanding-demo.ng.bluemix.net/?cm_mc_uid=18482519226615143151166&cm_mc_sid_50200000=1514576969&cm_mc_sid_52640000=1514576969

Voici un exemple d'utilisation de ce service en ligne :

Figure 8 : Démo en ligne de l'API « NLU »

Text

URL

Je m'appelle Omar Lo, étudiant à la Haute école de gestion de Genève

French

For results unique to your business needs consider building a custom model.

Analyze

Sentiment

Emotion

Keywords

Entities

Categories

Concept

Semantic Roles

Extract people, companies, organizations, cities, geographic features, and other information from the content.

Name	Type	Score
Omar Lo	Person	<div></div> 0.96
Genève	Location	<div></div> 0.04

Avec ce service on arrive donc à extraire les participants à partir du texte fourni par l'utilisateur.

Les scores indiquent le niveau de confiance de la machine pour définir une entité.

Exemple :

Omar Lo → score = 0.96 signifie que la machine est sûre à 96% que c'est une personne.

Genève → score = 0.04 (très faible) signifie que la machine est sûre à 4% que c'est un lieu.

Ce faible score est peut-être dû à une déduction sémantique.

MyNaturalLanguageUnderstanding est la classe qui implémente l'API « Natural Language Understanding » de Watson. Elle est codée de la façon suivante :

```
package servicesWatson;

import java.util.ArrayList;

/**
 * Réalisé dans le cadre de mon Travail de Bachelor 2017
 *
 * Classe permettant d'extraire la liste des participants avec leur nom et prénom
 * à partir d'un texte brut et de retourner cette liste
 *
 * @author Lo Omar - HEG-Genève
 * @version 1.0
 */
public class MyNaturalLanguageUnderstanding {

    private static MyNaturalLanguageUnderstanding uniqueInstance = new MyNaturalLanguageUnderstanding();

    /**
     * Identifiants de mon service sur Bluemix pour avoir accès à l'API
     * NaturalLanguageUnderstanding
     */
    private static final String MY_USERNAME = "15b80016-2b53-45e4-bea1-4a4c5d031cbb";
    private static final String MY_PASSWORD = "MI4ZNdaNlv01";

    /** Constructeur */
    private MyNaturalLanguageUnderstanding() {}

    public static MyNaturalLanguageUnderstanding getUniqueInstance() {
        return uniqueInstance;
    }

    public ArrayList<Participant> getListeParticipants(String texteBrut) {
        try {
            NaturalLanguageUnderstanding service = new NaturalLanguageUnderstanding(
                NaturalLanguageUnderstanding.VERSION_DATE_2017_02_27, MY_USERNAME, MY_PASSWORD);

            EntitiesOptions entitiesOptions = new EntitiesOptions.Builder().emotion(true).sentiment(true).build();
            KeywordsOptions keywordsOptions = new KeywordsOptions.Builder().emotion(true).sentiment(true).build();
            Features features = new Features.Builder().entities(entitiesOptions).keywords(keywordsOptions).build();
            AnalyzeOptions parameters = new AnalyzeOptions.Builder().text(texteBrut).features(features).build();

            AnalysisResults response = service.analyze(parameters).execute();

            List<EntitiesResult> entitiesResults = response.getEntities();
            ArrayList<Participant> participants = new ArrayList<Participant>();

            if (entitiesResults.size() > 0) {
                for (EntitiesResult entitiesResult : entitiesResults) {
                    if (entitiesResult.getType().equals("Person")) {
                        String[] strPersonne = entitiesResult.getText().split(" ");
                        try {
                            String prenom = strPersonne[0];
                            String nom = strPersonne[1];
                            participants.add(new Participant(nom, prenom));
                        } catch (Exception e) {
                            // TODO: handle exception
                            JOptionPane.showMessageDialog(null, "Il Faut le prénom et le nom de chaque participant!");
                            System.out.println("MyNaturalLanguageUnderstanding.getListeParticipants() " + e.getMessage());
                            e.printStackTrace();
                            return null;
                        }
                    }
                }
            }

            return participants;
        } catch (BadRequestException e) {
            // TODO: handle exception
            JOptionPane.showMessageDialog(null, "Veuillez vérifier le texte saisi!");
            System.out.println("MyNaturalLanguageUnderstanding.getListeParticipants() " + e.getMessage());
            e.printStackTrace();
            return null;
        }
    }

    // getEntities
}

// MyNaturalLanguage
```

7.1.2.1 Explication du code la classe MyNaturalLanguageUnderstanding

MyNaturalLanguageUnderstanding est la classe qui utilise l'API « Natural Understanding » de Watson. C'est un singleton pour les mêmes raisons citées en haut.

La méthode getListeParticipants prend en paramètre le texte brut et démarre le service avec une instance de NaturalLanguageUnderstanding et retourne la liste des participants.

```
NaturalLanguageUnderstanding service = new NaturalLanguageUnderstanding(  
    NaturalLanguageUnderstanding.VERSION_DATE_2017_02_27, MY_USERNAME, MY_PASSWORD);
```

Le constructeur de cette classe prend en paramètre la version du service et le username et le password obtenus au moment de la création du service sur notre compte Bluemix afin de permettre à l'instance de se connecter au bon service.

Il nous faut :

- une instance de EntitiesOptions pour l'analyse des entités.
- une instance de KeywordsOptions pour l'analyse des mots clés
- une instance de Features qui fixe les termes à prendre en compte dans l'analyse et les options qui sont activées.

La classe AnalyseOptions permet de construire la requête avec comme paramètre le texte à analyser et les features.

Et enfin l'exécution de la méthode analyse avec comme paramètre une instance de AnalyseOptions nous retourne une réponse de type AnalysisResults sur laquelle on applique la méthodes getEntities pour récupérer toutes les entités retrouvées dans le texte.

L'objectif dans ce travail était tout simplement de retrouver les personnes, pour cela on applique la méthode getType sur chaque entité trouvée si c'est une personne on la prend. C'est ainsi qu'on a pu extraire les noms et les prénoms des participants.

La documentation complète de cet API est disponible sur l'adresse suivante : <https://www.ibm.com/watson/developercloud/natural-language-understanding/api/v1/#introduction>

7.1.2.2 Les Limites de l'API « Natural Language Understanding »

Ce service a du mal souvent avec les noms de personne très longs ou trop compliqués. Il coupe souvent les noms trop longs. Ceci est illustré par la figure suivante :

Figure 9 : Démo en ligne de l'API « NLU » avec des noms compliqués

The screenshot shows the NLU API demo interface. At the top, there are tabs for 'Text' and 'URL'. The 'Text' tab is selected. Below the tabs is a text input area containing the sentence: 'créer un meeting le jeudi le matin entre Omar Lo et Patrick Ribeiro Amaral'. Below the input area, it says 'French' and 'For results unique to your business needs consider building a custom model.' There is an 'Analyze' button. Below the button are several filter buttons: 'Sentiment', 'Emotion', 'Keywords', 'Entities' (which is highlighted), 'Categories', 'Concept', and 'Semantic Roles'. Below the filters is a table with three columns: 'Name', 'Type', and 'Score'. The table contains two rows of results:

Name	Type	Score
Omar Lo	Person	0.96
Patrick Ribeiro	Person	0.54

On voit sur cet exemple que l'API « NLU » n'a pas pu extraire le nom complet de Patrick. Ceci est donc un handicap pour notre application.

On voulait dans le texte du rendez-vous fournir que les prénoms des participants mais il se trouve que dans certains cas, le service a du mal à identifier une personne à travers son prénom seulement, comme le montre la figure qui suit :

Figure 10 : Démo 1 en ligne de l'API « NLU » avec des prénoms des personnes

The screenshot shows the NLU API demo interface. At the top, there are tabs for 'Text' and 'URL'. The 'Text' tab is selected. Below the tabs is a text input area containing the sentence: 'créer un meeting le jeudi le matin entre Omar et Philippe'. Below the input area, it says 'French' and 'For results unique to your business needs consider building a custom model.' There is an 'Analyze' button. Below the button are several filter buttons: 'Sentiment', 'Emotion', 'Keywords', 'Entities' (which is highlighted), 'Categories', 'Concept', and 'Semantic Roles'.

Name	Type	Score
Omar	Person	<div><div></div></div> 0.98

Ici bien vrai que Philippe est une personne mais l'API n'arrive pas à l'identifier. Pour corriger cela, on exige dans notre application, le nom et le prénom des participants.

Figure 11 : Démo 2 en ligne de l'API « NLU » avec des prénoms des personnes

Text

URL

créer un meeting le jeudi le matin entre Omar et Philippe Dugerdil

French

For results unique to your business needs consider building a [custom model](#).

Analyze

Sentiment

Emotion

Keywords

Entities

Categories

Concept

Semantic Roles

Omar	Person	<div><div></div></div> 0.96
Philippe Dugerdil	Person	<div><div></div></div> 0.73

C'est pour cette raison que nous obligeons l'utilisateur à fournir le prénom et le nom de chaque participant pour que l'API puisse récupérer tous les participants contenus dans le texte.

7.1.3 Watson conversation

L'utilisation de cet API nous a permis finalement d'extraire les contraintes temporelles (date et heure) et le lieu du rendez-vous.

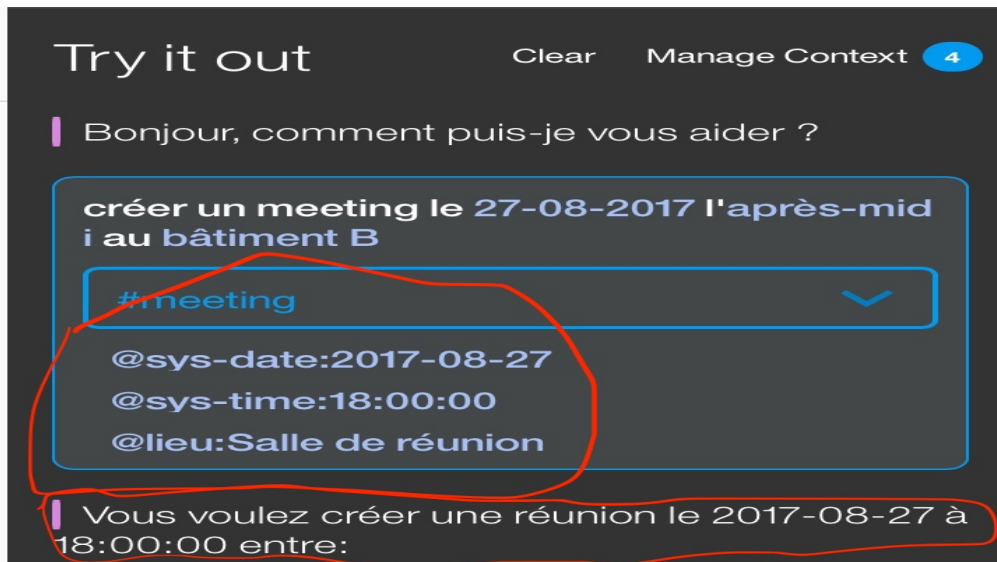
En effet ce service est capable de dialoguer avec un utilisateur à travers une interface textuelle. Selon le texte reçu, l'API est en mesure de fournir une réponse cohérente pour le besoin de l'utilisateur.

Pour cela il faut au préalable entrainer le service en lui injectant un certain nombre de mots qui vont constituer les intentions du dialogue, les entités manipulées dans le dialogue et construire finalement les conditions du dialogue proprement dit.

Une fois entraînée la machine pourra repérer les intentions du dialogue dans le texte reçu, vérifier que les entités manipulées sont toutes présentes dans le texte et fournir la réponse appropriée.

Exemple : « créer un meeting le 27-08-2017 l'après-midi au bâtiment B »

Figure 12 : Démo 1 en ligne du service « conversation » créé sur Bluemix



Ici l'intention du dialogue est le mot « meeting », les entités sont la date, l'heure et le lieu.

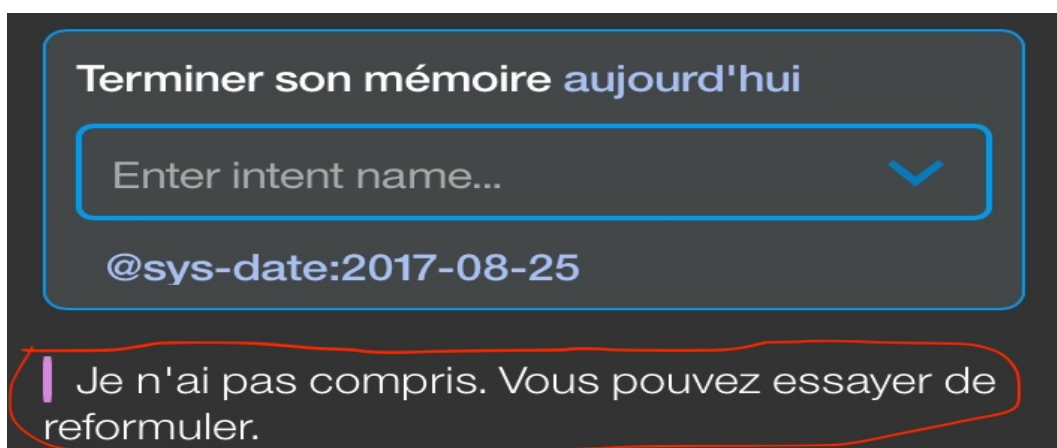
Et la réponse de la machine est : « Vous voulez créer une réunion le 2017-08-27 à 18 :00 :00 entre : ».

Si la machine ne trouve pas l'intention dans le texte reçu, elle doit pouvoir quand même fournir une réponse pour dire à l'utilisateur qu'elle n'a pas compris son intention.

Exemple : Question posée à la machine : « Terminer son mémoire aujourd'hui »

Réponse de la machine :

Figure 13 : Démo 2 en ligne du service « conversation » créé sur Bluemix

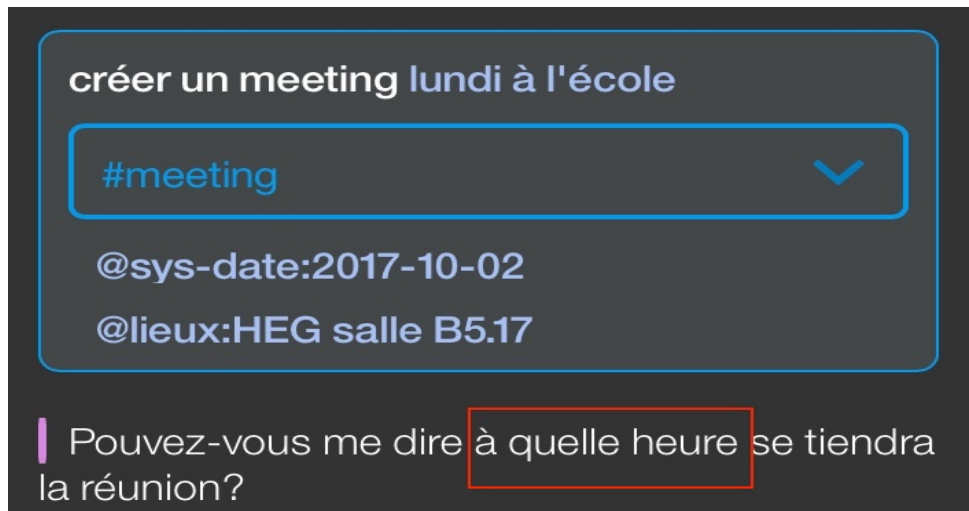


S'il y a des entités qui manquent également dans le texte du dialogue, la machine doit pouvoir donner une réponse à l'utilisateur pour l'inviter fournir la valeur manquante.

Exemple : Question posée à la machine : « créer un meeting lundi à l'école »

Réponse :

Figure 14 : Démo 3 en ligne du service « conversation » créé sur Bluemix



Les autres tests de ce service sont mis en annexe de ce document.

MyWatsonConversation est la classe qui implémente l'API « Conversation » de Watson. Elle est codée de la façon suivante :

```
package servicesWatson;

import java.util.HashMap;

/**
 * Réalisé dans le cadre de mon Travail de Bachelor 2017
 *
 * Classe permettant d'extraire la date, l'heure et le lieu du rendez-vous à partir d'un texte brut
 * et de les retourner
 *
 * @author Lo Omar - HEG-Genève
 * @version 1.0
 */

public class MyWatsonConversation {

    private static MyWatsonConversation uniqueInstance = new MyWatsonConversation();

    /**
     * Identifiants de mon service sur Bluemix pour avoir accès à l'API
     * Conversation service
     */
    private static final String MY_USERNAME = "6fe0ef83-2e86-4d6a-8d2b-c0a4e8aa7507";
    private static final String MY_PASSWORD = "ngPoYJYAqoP";
    private static final String MY_WORKSPACEID = "a9923861-0bc2-423d-b711-ff97cd23870b";

    /** Constructeur */
    private MyWatsonConversation() {}
}
```

```

public static MyWatsonConversation getUniqueInstance() {
    return uniqueInstance;
} // getUniqueInstance

public HashMap<String, String> getMapDateLieuHeure(String texteBrut) {
    try {
        HashMap<String, String> map = new HashMap<String, String>();

        ConversationService service = new ConversationService("2017-05-26");
        service.setUsernameAndPassword(MY_USERNAME, MY_PASSWORD);

        MessageRequest newMessage = new MessageRequest.Builder().inputText(texteBrut)
            // Replace with the context obtained from the initial
            // request
            // .context(...)
            .build();

        MessageResponse response = service.message(MY_WORKSPACEID, newMessage).execute();

        String outputText = response.getOutput().get("text").toString().substring(1,
            response.getOutput().get("text").toString().length() - 1);

        if (response.getEntities().size() == 3) {
            map.put(LIEU, response.getEntities().get(0).getValue());
            map.put(DATE, response.getEntities().get(1).getValue());
            map.put(TIME, response.getEntities().get(2).getValue());
            map.put(OUTPUT_TEXT, outputText);
        } else {
            map.put("outputText", outputText);
        }

        /*
        * for (Entity entity : response.getEntities()) {
        *     System.out.println(entity.getValue());
        * }
        */

        return map;
    } catch (BadRequestException e) {
        // TODO: handle exception
        JOptionPane.showMessageDialog(null, "Veuillez vérifier le texte saisi!");
        System.out.println("MyWatsonConversation.getMapDateLieuHeure() " + e.getMessage());
        e.printStackTrace();
        return null;
    }
} // getStringDateLieuHeure
} // MyWatsonConversation

```

7.1.3.1 Explication du code

MyWatsonConversation est la classe qui utilise le service « Conversation » de Watson. C'est un singleton pour les mêmes raisons citées en haut.

La méthode getMapDateLieuHeure prend en paramètre le texte fourni par l'utilisateur et démarre le service avec une instance de ConversationService et retourne une HashMap contenant la réponse de la machine, le lieu, la date et l'heure du rendez-vous.

```

ConversationService service = new ConversationService("2017-05-26");
service.setUsernameAndPassword(MY_USERNAME, MY_PASSWORD);

```

Le constructeur de cette classe prend en paramètre la version du service et le username et le password obtenus au moment de la création service sur notre compte Bluemix afin de permettre à l'instance de se connecter au bon service.

La classe MessageRequest permet de construire la requête avec comme input le texte fourni par l'utilisateur.

Et enfin l'exécution de la méthode message avec comme paramètre l'id de notre Workspace et une instance de MessageRequest, nous retourne une réponse de type MessageResponse qui contient la réponse du service et les différentes entités trouvées dans le texte.

Ces entités sont le lieu, la date et l'heure du rendez-vous.

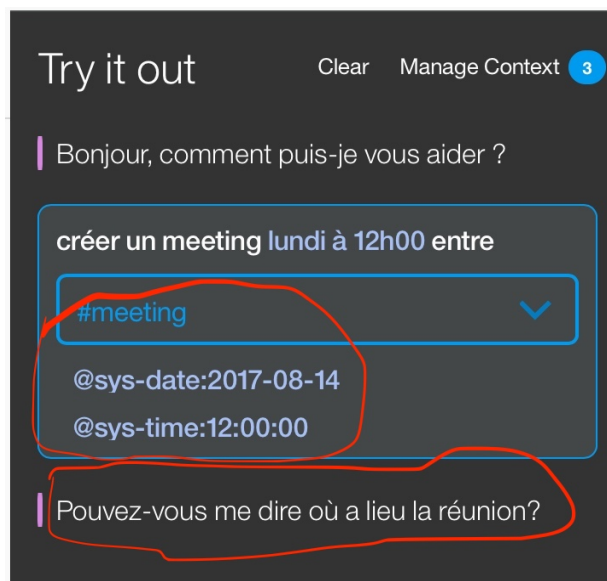
La documentation complète de cet API est disponible sur l'adresse suivante : <https://www.ibm.com/watson/developercloud/conversation/api/v1/>

7.1.3.2 Les Limites de l'API « Conversation service »

Il arrive dès fois que l'API se trompe sur les dates, par exemple entre lundi de la semaine passée et lundi de la semaine suivante.

Exemple : On a effectué ce test sur notre compte Bluemix le 16.08.2017.

Figure 15 : Démo 4 en ligne du service « conversation » créé sur Bluemix



On voit ici que la machine se trompe de date au lieu de prendre lundi de la semaine prochaine (21-08-2017), elle prend le lundi de la semaine dernière (14-08-2017). Mais c'est un cas rare.

7.2 Utilisation de EWS JAVA API

Pour réaliser les fonctionnalités de l'application, nous avons besoin d'un outil qui nous permet d'accéder aux ressources de Microsoft Exchange server.

En effet ce service est utilisé pour manipuler ou créer des informations dans une boîte aux lettres de Microsoft Exchange server grâce à une instance de la classe ExchangeService.

Pour instancier cette classe, il faut intégrer le SDK⁴ de cet API dans les librairies du projet.

Dans ce travail, nous nous sommes intéressé sur quatre points :

- Etablir la connexion entre l'application et le serveur Exchange
- Chercher l'email des participants sur le serveur

⁴ Software Development Kit

- Chercher les disponibilités des participants sur le serveur
- Créer un rendez-vous sur les agendas des participants

7.2.1 Etablir la connexion entre l'application et le serveur Exchange

Il y a plusieurs façons d'établir cette connexion avec le serveur Exchange et d'obtenir l'instance de ExchangeService. Ces différentes sont exposées en annexe de ce document.

ConnexionServeur est la classe qui nous fournit la connexion au serveur Exchange. Elle est codée de la façon suivante :

```

1 package servicesExchange;
2
3 import microsoft.exchange.webservices.data.autodiscover.IAutodiscoverRedirectionUrl;
4
5 /**
6  * Réalisé dans le cadre de mon Travail de Bachelor 2017
7  *
8  * Classe permettant d'établir la connexion entre l'application et le serveur Exchange
9  * et retourner de retourner une instance de ExchangeService
10  *
11  * @author Lo Omar - HEG-Genève
12  * @version 1.0
13  */
14
15 public class ConnexionServeur {
16
17     private static ConnexionServeur instance = new ConnexionServeur();
18     private ExchangeService service;
19     private ExchangeCredentials credentials;
20
21     /* Constructeur */
22     private ConnexionServeur() {}
23
24     public static ConnexionServeur getInstance() {
25         return instance;
26     } // getInstance
27
28     public ExchangeService getService(String nomUtilisateur, String password) {
29         try {
30             /*
31              * Alternatif à autodiscoverUrl si on connaît l'url du server
32              * service = new ExchangeService(ExchangeVersion.Exchange2007_SP1);
33              * credentials = new WebCredentials("omar.lo", "pwd", "hes");
34              * service.setUrl(new URI("https://m.hesge.ch/ews/Exchange.asmx"));
35              * service.setExchange2007CompatibilityMode(true);
36              */
37             service = new ExchangeService();
38             credentials = new WebCredentials(nomUtilisateur, password);
39             service.setCredentials(credentials);
40             service.autodiscoverUrl(nomUtilisateur + "@etu.hesge.ch", new RedirectionUrlCallback());
41             return service;
42
43         } catch (Exception e) {
44             System.out.println("Connexion.getService(): " + e.getMessage());
45             e.printStackTrace();
46             return null;
47         }
48     } // getService
49
50     // Pour chercher l'url du server sur la base du email du user
51     private class RedirectionUrlCallback implements IAutodiscoverRedirectionUrl {
52
53         public boolean autodiscoverRedirectionUrlValidationCallback(String redirectionUrl) {
54             return redirectionUrl.toLowerCase().startsWith("https://");
55         } // autodiscoverRedirectionUrlValidationCallback
56     } // RedirectionUrlCallback
57
58 } // Connexion

```

7.2.1.1 Explication du code de la classe ConnexionServeur

ConnexionServeur est la classe qui permet d'établir la connexion avec le serveur Exchange et de retourner une instance de la classe ExchangeService. C'est une classe singleton pour les mêmes raisons évoquées en haut.

La méthode `getService` avec comme paramètre le nom d'utilisateur et le mot de passe de l'utilisateur qui utilise l'application, démarre le service et retourne une instance la classe `ExchangeService`. On a deux façons d'instancier cette classe :

- soit avec un constructeur vide

```
ExchangeService service = new ExchangeService();
```

- soit avec un constructeur paramétré par la version du serveur qu'on veut joindre

```
ExchangeService service = new ExchangeService(ExchangeVersion.Exchange2007_SP1);
```

Ensuite il faut :

- instancier la classe `ExchangeCredentials` avec les identifiants de l'utilisateur.

On peut aussi au moment de l'instanciation de cette classe passer en plus des identifiants de l'utilisateur, du nom de domaine du serveur qu'on veut joindre mais cela n'est pas obligatoire.

```
ExchangeCredentials credentials = new WebCredentials("username", "pwd", "domaine name du serveur");
```

- setter l'instance de `ExchangeCredentials` :

Pour joindre maintenant le serveur, on a deux solutions :

- soit setter manuellement l'URL du serveur si on le connaît, comme le montre le code suivant :

```
service.SetUrl(new Uri("https://m.hesge.ch/ews/Exchange.asmx"));
```

- soit faire de la découverte automatique d'URL avec la méthode `autodiscoverUrl` avec en paramètre l'email de l'utilisateur et une instance de la classe `RedirectionUrlCallback` pour rediriger l'Url au cas échéant où le service n'arrive pas joindre le serveur.

```
service.autodiscoverUrl(nomUtilisateur + "@etu.hesge.ch", new RedirectionUrlCallback());
```

Avec ça, on obtient une instance complète de la classe `ExchangeService` qu'on peut utiliser pour manipuler les ressources de Microsoft Exchange server.

7.2.2 Chercher l'email des participants sur le serveur

Pour récupérer l'email d'une personne sur le serveur, on a besoin d'une instance de la classe `NameResolutionCollection`. La classe qui permet de retrouver les emails des participants est codée de la façon suivante :


```

1 package servicesExchange;
2
3 import domaine.Participant;
4
5 /**
6  * Réalisé dans le cadre de mon Travail de Bachelor 2017
7  * Classe permettant de retrouver l'email d'un participant sur le serveur Exchange et de le retourner
8  * @author Lo Omar - HEG-Genève
9  * @version 1.0
10 */
11
12 public class MyGetEmailParticipant {
13
14     private static MyGetEmailParticipant instance = new MyGetEmailParticipant();
15     private Data data = Data.getUniqueInstance();
16     private ExchangeService service;
17
18     /* Constructeur */
19     private MyGetEmailParticipant() {
20         super();
21         service = data.getService();
22     }
23
24     public static MyGetEmailParticipant getInstance() {
25         return instance;
26     } // getInstance
27
28     public String getEmailParticipant(Participant p) {
29         try {
30             NameResolutionCollection resolveName = service.resolveName(p.toString(),
31                 ResolveNameSearchLocation.DirectoryOnly, true);
32             // System.out.println(resolveName.getCount());
33             NameResolution nameResolution = resolveName.nameResolutionCollection(0);
34             String strAdresse1 = nameResolution.getContact().getEmailAddresses()
35                 .getEmailAddress(EmailAddressKey.EmailAddress1).getAddress();
36             String strAdresse2 = nameResolution.getContact().getEmailAddresses()
37                 .getEmailAddress(EmailAddressKey.EmailAddress2).getAddress();
38
39             String[] tabAdresse1 = strAdresse1.split(":");
40             String[] tabAdresse2 = strAdresse2.split(":");
41             if (tabAdresse1[1].endsWith("tech")) {
42                 return tabAdresse2[1];
43             } else {
44                 return tabAdresse1[1];
45             }
46         } catch (Exception e) {
47             System.out.println("MyGetEmailParticipant.getEmailParticipant() " + e.getMessage());
48             e.printStackTrace();
49             return null;
50         }
51     } // getEmailParticipant
52
53 } // MyGetEmailParticipant

```

7.2.2.1 Explication du code de la classe MyGetEmailParticipant

MyGetEmailParticipant est la classe qui nous fournit l'email d'une personne sur le serveur. C'est une classe singleton pour les mêmes raisons citées en haut.

La méthode getEmailParticipant avec en paramètre le participant démarre le service avec une instance de NameResolutionCollection et retourne l'email du participant en question si ce dernier est sur le serveur.

La méthode resolveName avec en paramètre le nom et le prénom de la personne, une constante resolveNameSearchLocation.DirectoryOnly pour parcourir toute la hiérarchie (contacts et annuaire) pour retrouver la bonne personne et la valeur « true » pour activer la recherche, retourne une collection d'information liée à la personne.

Pour rappel dans le serveur mail de la HEG, chaque personne à deux adresses mails et il fallait faire un test pour trouver la bonne adresse et la retourner.

Cette classe permet en plus de retrouver l'email d'une personne mais aussi de tester la validité d'un participant. Autrement dit la méthode getEmailParticipant retourne soit un String correspondant à l'email du participant, soit null si le participant n'existe pas sur le serveur.

Pour faire la validation, on teste si le String reçu est différent de null, si oui on sette son email et on ajoute le participant à la liste des participants, sinon on arrête le processus de création de rendez-vous et on affiche un message pour informer l'utilisateur.

Voici le code en question qui permet de vérifier la validité d'un participant :

```
private void recupererListeParticipants() {
    // TODO Auto-generated method stub
    treeParticipants = new TreeSet<Participant>();
    listeParticipants = myUnderstanding.getListeParticipants(texteBrut);
    if (listeParticipants != null) {
        for (Participant participant : listeParticipants) {
            String emailParticipant = myGetEmailParticipant.getEmailParticipant(participant);
            if (emailParticipant != null) {
                participant.setEmail(emailParticipant);
                treeParticipants.add(participant);
            } else {
                JOptionPane.showMessageDialog(null, participant.toString()
                    + " est inconnu du système. Veuillez vérifier le nom des participants!");
            }
        }
    }
}

} // recupererListeParticipants
```

7.2.3 Chercher les disponibilités des participants sur le serveur

La méthode getUserAvailability de la classe ExchangeService nous retourne une collection de type GetUserAvailabilityResults contenant deux listes :

- une liste des plages horaires occupées
- une liste des plages horaires libres c'est-à-dire non occupées

Dans ce travail, nous nous sommes intéressés au traitement des plages horaires libres. La classe qui permet de retourner cette collection est codée de la façon suivante :

```
1 package servicesExchange;
2
3 import java.text.SimpleDateFormat;
4
5 /**
6  * Réalisé dans le cadre de mon Travail de Bachelor 2017
7  *
8  * Classe permettant de chercher la liste des disponibilités communes des participants et de la retourner
9  *
10  * @author Lo Omar - HEG-Génève
11  * @version 1.0
12  */
13
14 public class MyGetPlagesHoraires {
15
16     private static MyGetPlagesHoraires instance = new MyGetPlagesHoraires();
17     private Data data = Data.getUniqueInstance();
18     private ExchangeService service;
19
20     /* Constructeur */
21     private MyGetPlagesHoraires() {
22         super();
23         service = data.getService();
24     }
25
26     public static MyGetPlagesHoraires getInstance() {
27         return instance;
28     }
29
30     public GetUserAvailabilityResults getPlagesHoraires(RecherchePlageHoraire recherchePlageHoraire) {
31
32         try {
33             // Create a list of attendees for which to request availability
34             // information and meeting time suggestions.
35
36             List<AttendeeInfo> attendees = new ArrayList<AttendeeInfo>();
37
38             for (Participant p : recherchePlageHoraire.getParticipants()) {
39                 attendees.add(new AttendeeInfo(p.getEmail()));
40             }
41
42             SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
43
44             // minimum time frame allowed by API is 24 hours
45             Date start = formatter.parse(recherchePlageHoraire.getDateDebut());
46             Date end = formatter.parse(recherchePlageHoraire.getDateFin());
47
48             // Specify options to request free/busy information and suggested
49             // meeting times.
50             AvailabilityOptions availabilityOptions = new AvailabilityOptions();
51             availabilityOptions.setGoodSuggestionThreshold(49);
52             availabilityOptions.setMaximumNonWorkHoursSuggestionsPerDay(0);
53             // Pas besoin de setter 60 minutes car c'est la valeur par défaut
54             // de MeetingDuration.
55             availabilityOptions.setMeetingDuration(60);
56             // availabilityOptions.setMergedFreeBusyInterval(60);
57             availabilityOptions.setMinimumSuggestionQuality(SuggestionQuality.Good);
58             availabilityOptions.setDetailedSuggestionsWindow(new TimeWindow(start, end));
59             availabilityOptions.setRequestedFreeBusyView(FreeBusyViewType.FreeBusy);
60
61         } catch (Exception e) {
62             // TODO Auto-generated catch block
63             e.printStackTrace();
64         }
65
66         return service.getUserAvailability(start, end, attendees, availabilityOptions);
67     }
68 }
```



```

        // Return free/busy information and a set of suggested meeting
        // times.
        return service.getUserAvailability(attendees, availabilityOptions.getDetailedSuggestionsWindow(),
            AvailabilityData.FreeBusyAndSuggestions, availabilityOptions);

    } catch (Exception e) {
        System.out.println("MyGetPlagesHoraires.getPlagesHoraires() " + e.getMessage());
        e.printStackTrace();
        return null;
    }
} // getPlagesHoraires

} // MyGetPlagesHoraires

```

7.2.3.1 Explication du code

MyGetPlagesHoraires est la classe qui permet de retourner une instance de GetUserAvailabilityResults. C'est un singleton pour les mêmes raisons citées en haut. Le paramètre de la méthode getPlagesHoraires est une instance de la classe RecherchePlageHoraire qui encapsule la plage horaire (date de début et date de fin) et la liste des participants. Pour trouver les plages horaires, il faut :

- constituer la liste des attendees (participants) avec les emails des participants
- des dates de début et de fin du meeting
- une instance de la classe AvailabilityOptions pour setter les différentes options

En appelant la méthode getUserAvailability avec comme paramètre la liste des attendees, la plage horaire, une constante pour spécifier les événements recherchés et les options de la recherche, nous retournons la collection des plages horaires occupées et non occupées.

Voici le code permettant de formater les informations de la collection et de les afficher en français sur l'écran de l'utilisateur :

```

59 private void rechercherPlagesHoraires() {
60     Calendar cal = Calendar.getInstance();
61     SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
62     Date dateBase = data.getDateBaseMeetingCorrecte();
63
64     String strDateDebut = formatter.format(dateBase);
65
66     cal.setTime(dateBase);
67     cal.add(Calendar.DATE, 1);
68     String strDateFin = formatter.format(cal.getTime());
69
70     RecherchePlageHoraire recherchePlageHoraire = new RecherchePlageHoraire(strDateDebut, strDateFin, participants);
71     GetUserAvailabilityResults results = myGetPlagesHoraires.getPlagesHoraires(recherchePlageHoraire);
72     PlageHoraireSuggerees = new ArrayList<PlageHoraireSuggere>();
73     try {
74         for (Suggestion suggestion : results.getSuggestions()) {
75             for (TimeSuggestion timeSuggestion : suggestion.getTimeSuggestions()) {
76                 PlageHoraireSuggere plageHoraireSuggere = new PlageHoraireSuggere(
77                     timeSuggestion.getMeetingTime().toString(), timeSuggestion.getQuality().toString(),
78                     timeSuggestion.getMeetingTime());
79                 plageHoraireSuggerees.add(plageHoraireSuggere);
80             }
81         }
82     } catch (Exception e) {
83         System.out.println("SmartAgendaViewSuggestionHoraire.rechercherPlagesHoraires() " + e.getMessage());
84         e.printStackTrace();
85     }
86     afficherLesSuggestions();
87
88 } // rechercherPlagesHoraires

```

```

91 private void afficherLesSuggestions() {
92     lstSuggestions.removeAll();
93     if (plageHoraireSuggeres.size() > 0) {
94         for (PlageHoraireSuggere plage : plageHoraireSuggeres) {
95             String nomJourFrench = "";
96
97             String strTime[] = plage.getHeureSuggere().split(" ");
98             String nomJour = strTime[0];
99             if (nomJour.startsWith("Mon")) {
100                 nomJourFrench = "Lundi";
101             } else if (nomJour.startsWith("Tue")) {
102                 nomJourFrench = "Mardi";
103             } else if (nomJour.startsWith("Wed")) {
104                 nomJourFrench = "Mercredi";
105             } else if (nomJour.startsWith("Thu")) {
106                 nomJourFrench = "Jeudi";
107             } else if (nomJour.startsWith("Fri")) {
108                 nomJourFrench = "Vendredi";
109             } else if (nomJour.startsWith("Sat")) {
110                 nomJourFrench = "Samedi";
111             } else {
112                 nomJourFrench = "Dimanche";
113             }
114
115             String mois = strTime[1];
116
117             String moisFrench;
118             if (mois.startsWith("Jan")) {
119                 moisFrench = "Janvier";
120             } else if (mois.startsWith("Feb")) {
121                 moisFrench = "Février";
122             } else if (mois.startsWith("Mar")) {
123                 moisFrench = "Mars";
124             } else if (mois.startsWith("Apr")) {
125                 moisFrench = "Avril";
126             } else if (mois.startsWith("May")) {
127                 moisFrench = "Mai";
128             } else if (mois.startsWith("Jun")) {
129                 moisFrench = "Juin";
130             } else if (mois.startsWith("Jul")) {
131                 moisFrench = "Juillet";
132             } else if (mois.startsWith("Aug")) {
133                 moisFrench = "Août";
134             } else if (mois.startsWith("Sep")) {
135                 moisFrench = "Septembre";
136             } else if (mois.startsWith("Oct")) {
137                 moisFrench = "Octobre";
138             } else if (mois.startsWith("Nov")) {
139                 moisFrench = "Novembre";
140             } else {
141                 moisFrench = "Décembre";
142             }
143
144             String jour = strTime[2];
145             String time = strTime[3];
146             String annee = strTime[5];
147             String strTimeSuggere = nomJourFrench + " " + jour + " " + moisFrench + " " + annee + " à " + time;
148             lstSuggestions.add("Proposition de rendez-vous: " + strTimeSuggere + " " + plage.getNiveauQualite());
149         }
150     } else {
151         JOptionPane.showMessageDialog(null, "Aucune disponibilité n'est trouvée!");
152     }
153 }
154 } // afficherLesSuggestions

```

7.2.4 Créer un rendez-vous sur les agendas des participants

Pour créer un rendez-vous sur le serveur, nous avons besoin d'une instance de la classe `Appointement` avec comme paramètre une instance de la classe `ExchangeService`. La classe qui permet de créer un rendez-vous sur l'agenda des participants, est codée de la façon suivante :

```

1 package servicesExchange;
2
3 import java.text.SimpleDateFormat;
15 /**
16  * Réalisé dans le cadre de mon Travail de Bachelor 2017
17  *
18  * Classe permettant de créer un rendez-vous sur le serveur Exchange et d'envoyer les notifications aux participants
19  *
20  * @author Lo Omar - HEG-Genève
21  * @version 1.0
22  */
23
24 public class MyAppointement {
25
26     private static MyAppointement instance = new MyAppointement();
27     private Data data = Data.getUniqueInstance();
28     private ExchangeService service;
29
30     /* Constructeur */
31     private MyAppointement() {
32         super();
33         service = data.getService();
34     }
35
36     // Créer un rendez-vous.
37     public int creerAppointement() {
38         String dateDebut = data.getDateDebutMeeting();
39         String dateFin = data.getDateFinMeeting();
40         TreeSet<Participant> participants = data.getTreeParticipants();
41         String messageBody = data.getMessageBody();
42         String subjectMeeting = data.getSubjectMeeting();
43         String lieuMeeting = data.getMapDateLieuHeure().get(LIEU);
44         StringBuffer sb = new StringBuffer();
45         String[] tabDateDebut = dateDebut.split(" ");
46         String[] strDateAffichee = tabDateDebut[0].split("-");
47         String annee = strDateAffichee[0];
48         String mois = strDateAffichee[1];
49         String jour = strDateAffichee[2];
50         String strHeureAffichee = tabDateDebut[1];
51         sb.append("Réunion le " + jour + "-" + mois + "-" + annee + " à " + strHeureAffichee + "\n");
52         sb.append("Lieu: " + lieuMeeting + "\n");
53         sb.append("Participants: \n");
54         for (Participant p : participants) {
55             sb.append(p.toString() + "\n");
56         }
57         sb.append("Voulez-vous créer cette réunion?");
58
59         int myAction = JOptionPane.showConfirmDialog(null, sb, "Création de rendez-vous",
60             JOptionPane.YES_NO_CANCEL_OPTION);
61         switch (myAction) {
62             case 0:
63                 try {
64                     Appointment appointment = new Appointment(service);
65                     SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
66                     Date startDate = formatter.parse(dateDebut);
67                     Date endDate = formatter.parse(dateFin);
68                     // Set properties on the appointment. Add two required attendees
69                     // and
70                     // one optional attendee.
71                     appointment.setSubject(subjectMeeting);
72                     appointment.setBody(new MessageBody(messageBody));
73                     appointment.setStart(startDate);
74                     appointment.setEnd(endDate);
75                     appointment.setLocation(lieuMeeting);
76                     for (Participant p : participants) {
77                         appointment.getRequiredAttendees().add(p.getEmail());
78                     }
79
80                     // appointment.setReminderMinutesBeforeStart(60);
81                     appointment.setReminderMinutesBeforeStart(30);
82                     // Send the meeting request to all attendees and save a copy in
83                     // the
84                     // Sent Items folder.
85                     appointment.save();
86                     JOptionPane.showMessageDialog(null, "Un nouveau rendez-vous vient d'être créé dans votre agenda!");
87                 } catch (Exception e) {
88                     System.out.println("MyAppointement.creerAppointement() " + e.getMessage());
89                     e.printStackTrace();
90                 }
91
92                 return myAction;
93             case 1:
94                 return myAction;
95             case 2:
96                 return myAction;
97             default:
98                 return -1;
99         }
100     } // creerAppointement
101 } // MyAppointement

```

7.2.4.1 Explication du code

MyAppointement est la classe qui permet de créer le rendez-vous. C'est un singleton pour les mêmes raisons expliquées en haut. Pour créer un rendez-vous, il nous faut :

- La date de début et la date de fin (ex : 2017-08-26 12 :00 :00 et 2017-08-26 13 :00 :00)

- Les emails des participants
- Le message body (optionnel)
- L'objet du rendez-vous (optionnel)
- Le lieu du rendez-vous (optionnel)

Tous ces éléments sont centralisés dans la classe Data pour des raisons de maintenabilité. On a fait une fenêtre pop-up qui affiche toutes ces informations pour demander à l'utilisateur de faire une validation avant de créer le rendez-vous.

Si l'utilisateur valide en appuyant sur le bouton « Oui » de la fenêtre pop-up, le système crée le rendez-vous.

7.2.5 Les limites du SDK EWS-JAVA-API

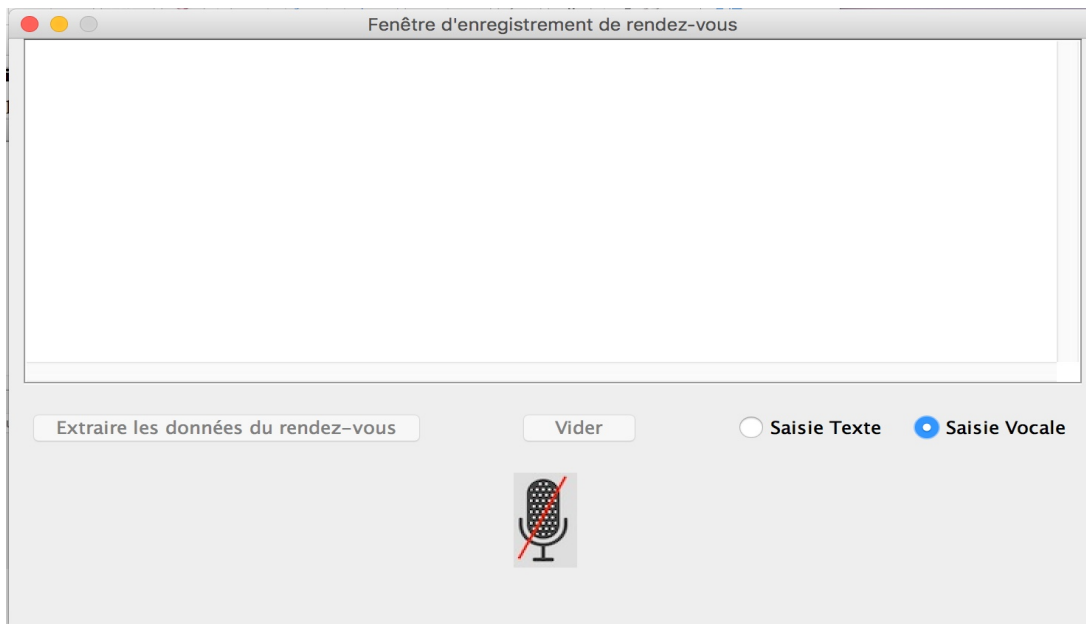
Parmi les limites de cet API, on peut citer :

- Il n'existe pas une version de cette librairie compatible avec une application mobile.
- Ce n'est pas un API REST
- La documentation complète de cet API est fournie en langage c#

8. Démonstration avec SmartAgenda pour créer un rendez-vous

Tout d'abord voici l'écran principal de l'application :

Figure 16 : Ecran principal de SmartAgenda



Cet écran est visible que si l'utilisateur est identifié sur le système.

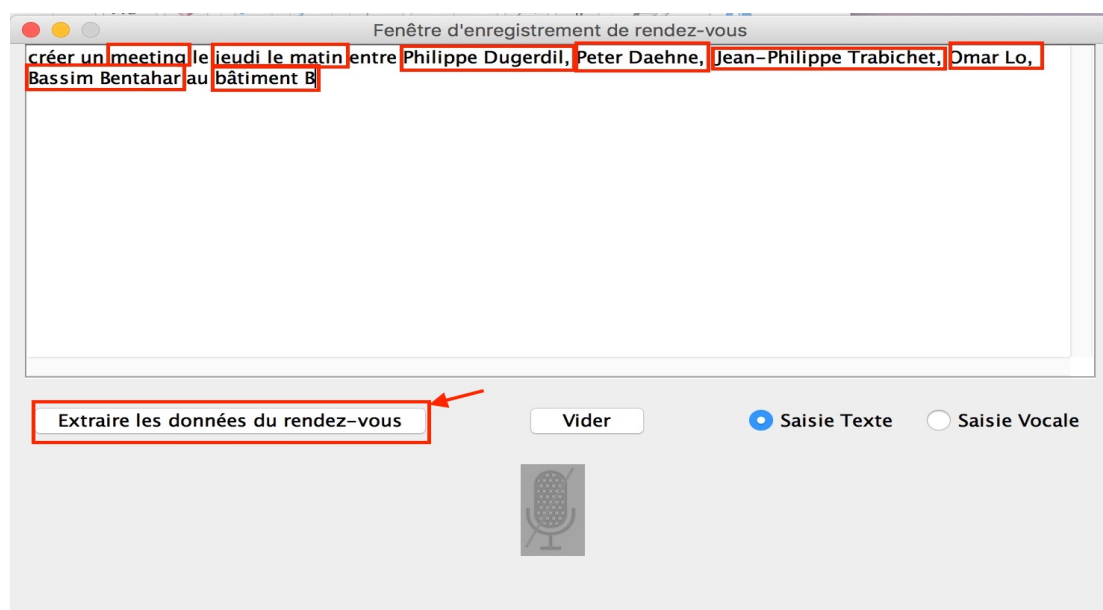
Pour créer un rendez-vous, l'utilisateur a le choix entre la saisie vocale ou textuelle des données. L'état par défaut du système est « saisie vocale ».

Voici des exemples de phrases pour créer un rendez-vous sur SmartAgenda. La liste n'est pas exhaustive :

- Créer un rendez-vous demain matin entre + liste participants + lieu
- Créer une réunion demain matin + lieu + entre+ liste participants
- Créer une réunion jeudi matin + lieu + entre+ liste participants
- Créer une réunion jeudi l'après-midi + lieu + entre+ liste participants
- Créer un meeting lundi à 12h00 + lieu + entre+ liste participants
- Créer un rendez-vous le 26/08/2017 le matin + entre+ liste participants + lieu
- Créer un meeting Mardi à 10h30 + lieu + entre+ liste participants
- Créer un rendez-vous le 26-08-2017 l'après-midi + entre+ liste participants + lieu
- Créer un rendez-vous + lieu + le 26-08-2017 l'après-midi + entre+ liste participants

8.1 Type de texte valide sur SmartAgenda

Figure 17 : Démo Ecran principal de SmartAgenda



Si l'utilisateur clique sur le bouton « Extraire les données du rendez-vous », l'application extrait les contraintes temporelles, le lieu du rendez-vous et la liste des participants. Voici le résultat obtenu :

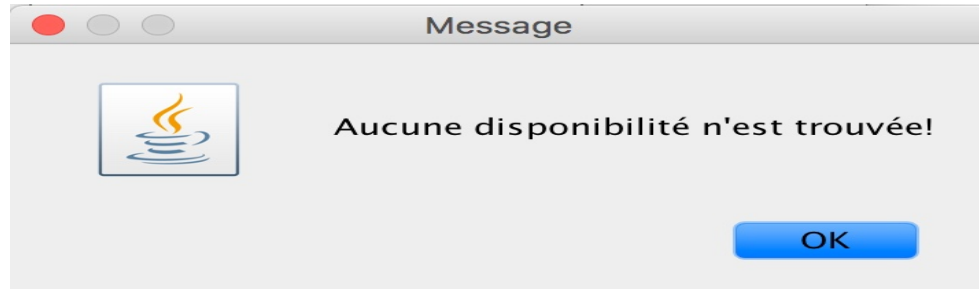
Figure 18 : Ecran de visualisation des infos du rendez-vous



Si l'utilisateur clique sur le bouton « Chercher Disponibilités », l'application lance la recherche des disponibilités communes entre les participants. Deux cas sont possibles.

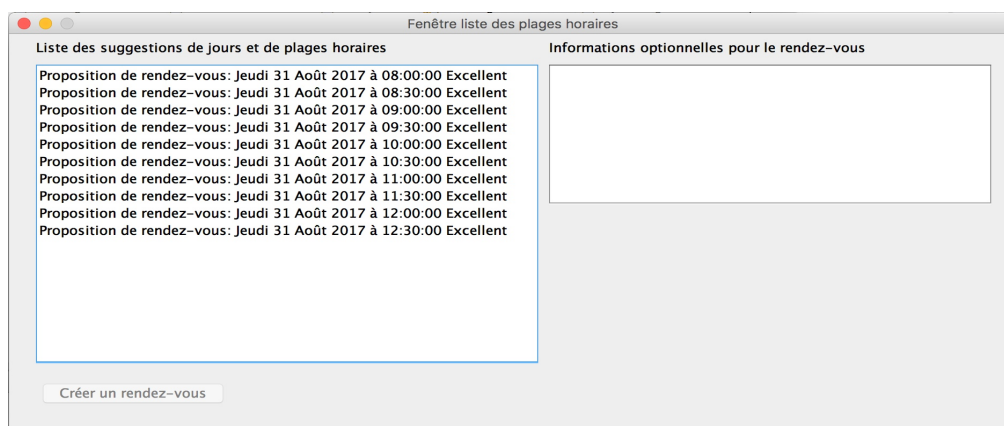
- Cas où il n'y a pas de disponibilité :

Figure 19 : Ecran pop-up



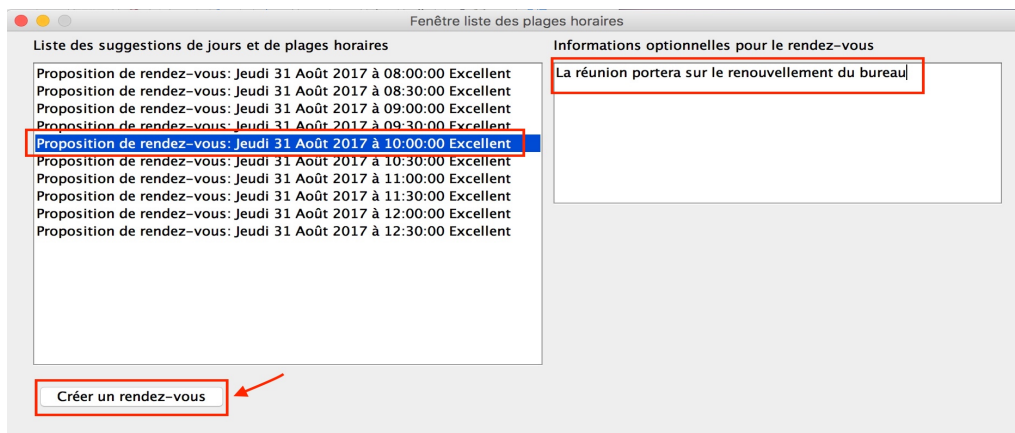
- Cas où il y a des disponibilités :

Figure 20 : Ecran de création de rendez-vous



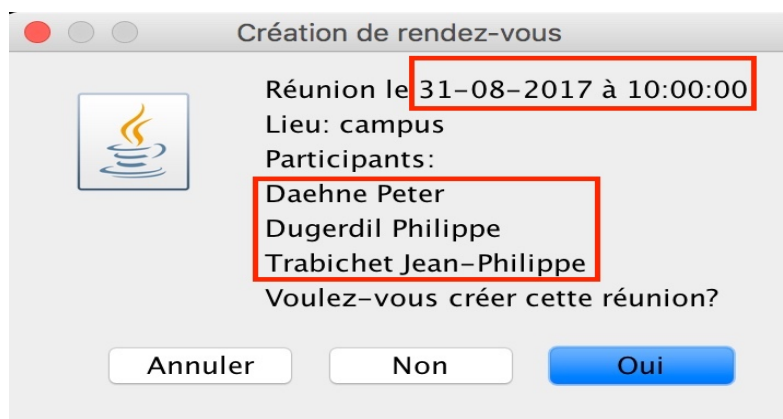
L'utilisateur peut à présent sélectionner une proposition et ajouter éventuellement une note (optionnelle) pour la réunion à créer, comme le montre la figure suivante.

Figure 21 : Ecran de création de rendez-vous avec sélection



Si l'utilisateur clique sur le bouton « Créer un rendez-vous », on obtient le résultat suivant :

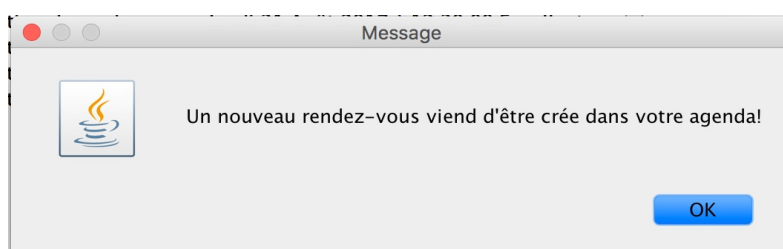
Figure 22 : Ecran pop-up de visualisation du rendez-vous à créer



Cette fenêtre permet d'afficher les informations complètes du rendez-vous à créer. L'utilisateur a dès lors trois choix possibles à faire :

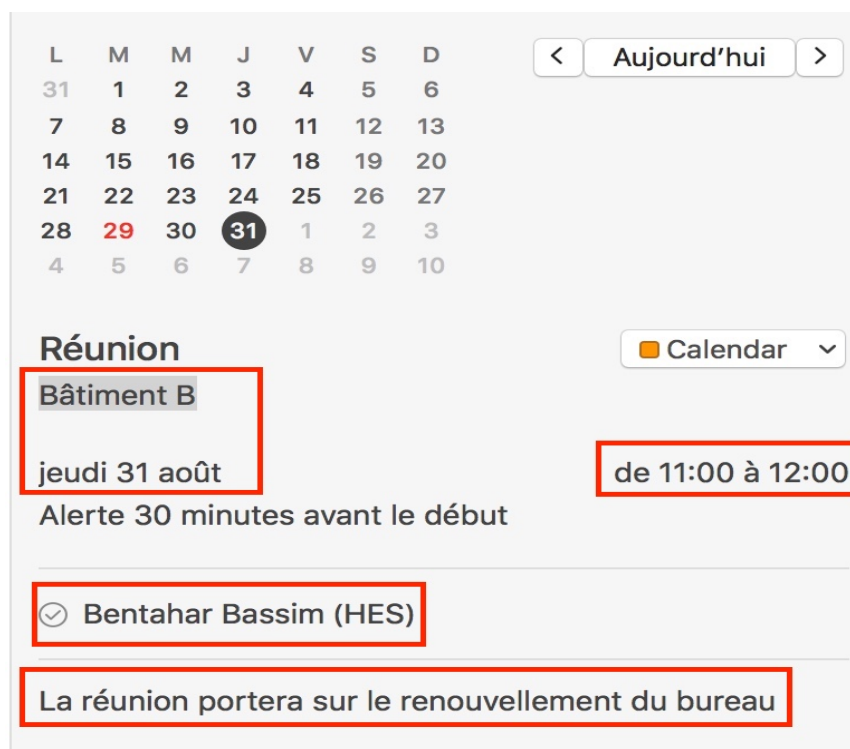
- soit cliquer sur le bouton « Non » : alors l'application affiche à nouveau la liste des propositions de rendez-vous.
- soit cliquer sur le bouton « Annuler » : alors l'application affiche à nouveau la fenêtre principale de l'application
- soit cliquer sur le bouton « Oui » : alors le rendez-vous est créé sur le serveur, comme le montre la figure qui suit.

Figure 23 : Ecran pop-up pour un rendez-vous créé



Lorsqu'un rendez-vous est créé avec SmartAgenda, tous les participants reçoivent une notification liée au rendez-vous qui vient d'être créé sur leur agenda, comme le montre la figure suivante tirée de mon agenda.

Figure 24 : Capture d'écran de mon agenda

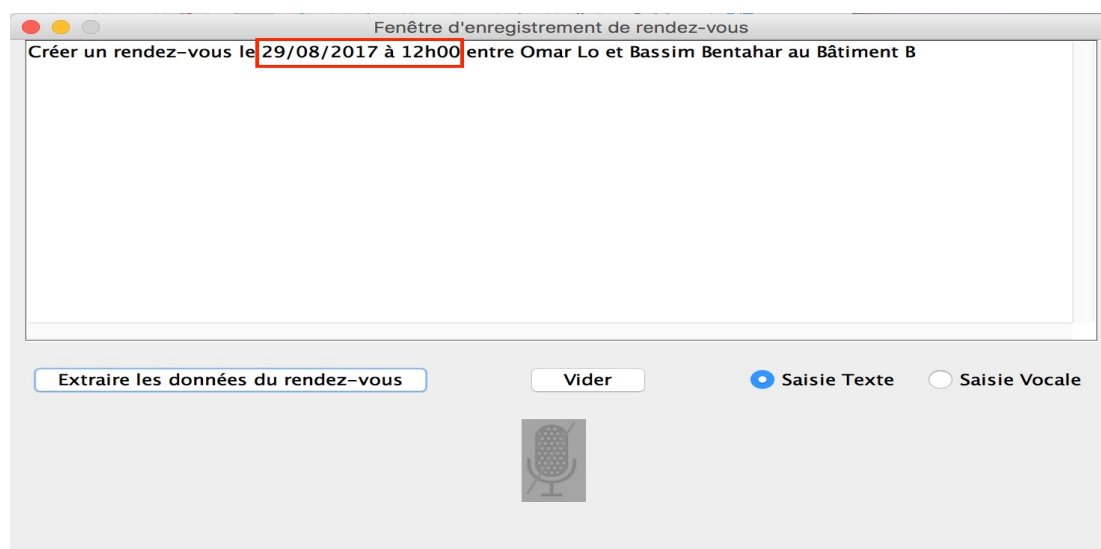


8.2 Type de texte invalide sur SmartAgenda

Voici quelques formats de texte que l'API « Conversation » a du mal à les interpréter :

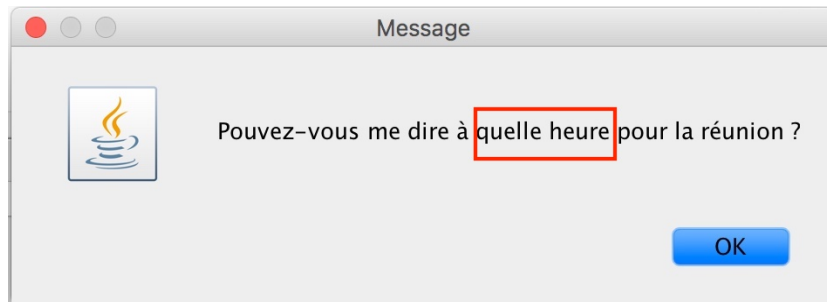
- Toute date et heure fournies ainsi : **jj/mm/aaaa à + heure** est invalide.

Figure 25 : Démo avec date et heure invalide



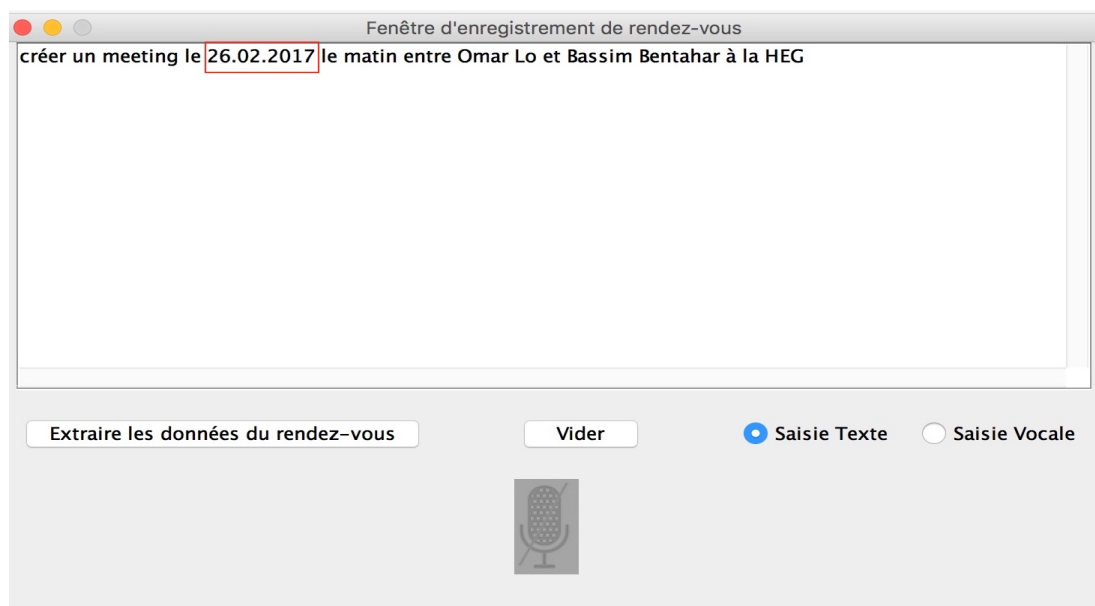
On voit que bien que le texte du rendez-vous est complet, toutes les informations sont présentes. Cependant, l'API « Conversation » a du mal à séparer la date fournie et l'heure. Il considère tout ce segment (29/08/2017 à 12h00) comme étant la date, comme le montre le message sur la figure qui suit :

Figure 26 : Ecran pop-up avec date et heure invalide



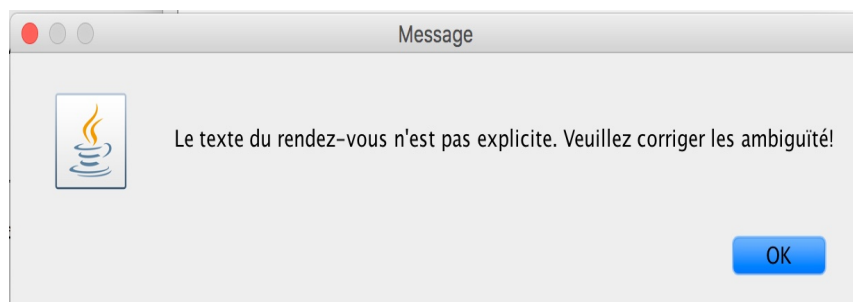
- Toute date formatée ainsi : **jj.mm.aaaa** est invalide.

Figure 27 : Démo avec date invalide



Ce format de date n'est pas supporté par l'API « Conversation » de Watson, comme le montre la figure qui suit :

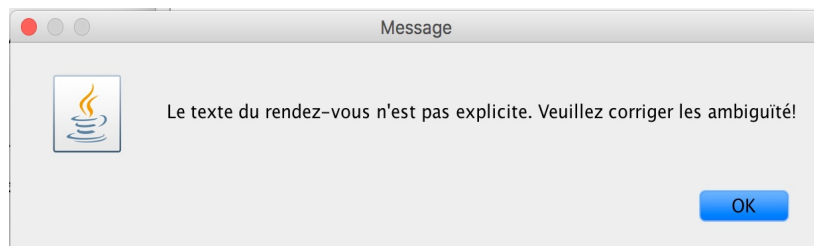
Figure 28 : Ecran pop-up avec date invalide



- On ne peut pas sur SmartAgenda, traiter les points suivants :
 - 1) Toute intervalle de date dans le texte du rendez-vous : Exemple : lundi et mardi ou mercredi et vendredi ou 26/08/2017 et 29/08/2017 etc.
 - 2) Toute intervalle d'heure dans le texte du rendez-vous : Exemple : 09h00 et 12h30 ou le matin et le soir etc.
 - 3) Si le texte comporte plus d'un lieu pour le rendez-vous
 - 4) Si le texte du rendez-vous n'a rien avoir avec le thème de l'application : Exemple : Je vais à l'école avec Omar Lo.

Ces différents cas cités, provoquent le message d'erreur suivant sur l'application :

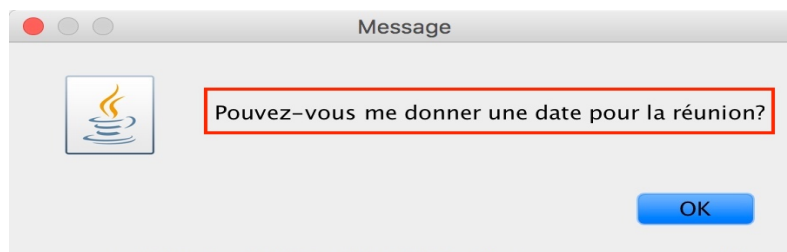
Figure 29 : Ecran pop-up avec des formats invalides



NB : Pour créer un rendez-vous sur SmartAgenda, il faut que le texte soit complet. Le texte doit comporter au moins une date, une heure, un lieu et au moins un nom complet de personne.

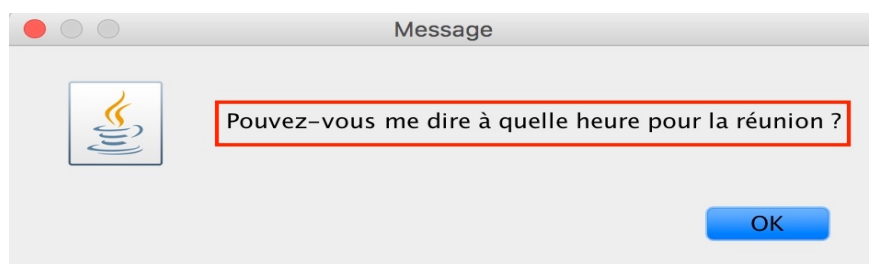
- Toute omission des informations suivantes, entraine un message d'erreur :
 - 1) Si le texte du rendez-vous ne contient pas de date ou la date n'est pas valide pour le système. Si tel est le cas, le système affiche le message d'erreur suivant :

Figure 30 : Ecran pop-up pour omission de date



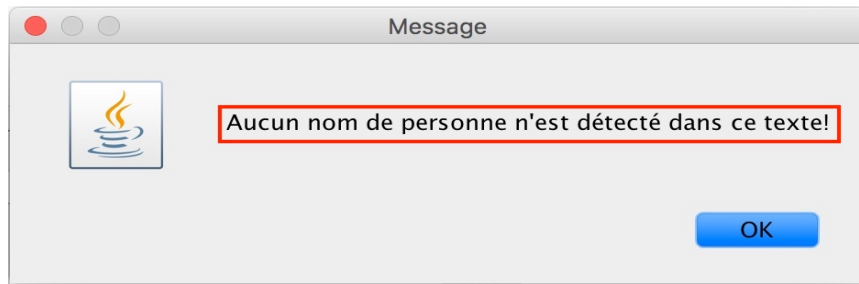
- 2) Si le texte du rendez-vous ne contient pas d'heure ou l'heure n'est pas valide pour le système. Si tel est le cas, le système affiche le message d'erreur suivant :

Figure 31 : Ecran pop-up pour omission d'heure



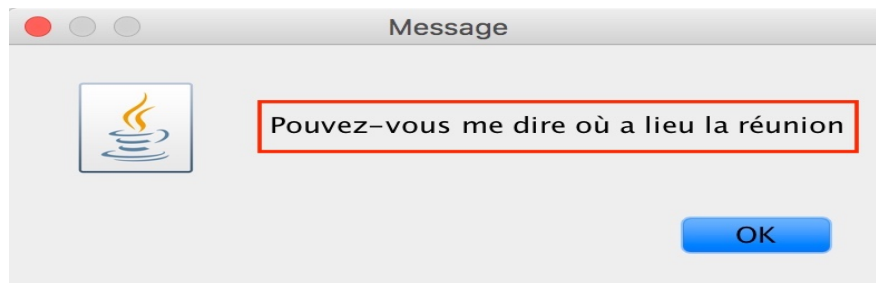
- 3) Si le texte du rendez-vous ne contient aucun participant, le système affiche le message d'erreur suivant :

Figure 32 : Ecran pop-up si aucun participant dans le texte



- 4) Si le texte du rendez-vous ne contient pas un lieu pour le rendez-vous, le système affiche le message d'erreur suivant :

Figure 33 : Ecran pop-up si omission du lieu de rendez-vous



9. Les difficultés rencontrées durant ce travail

L'utilisation dans ce travail des APIs Watson et Exchange était un champ nouveau pour nous. C'est pourquoi nous nous sommes confrontés dès le début à des difficultés liées à la compréhension et à la mise en œuvre de ces APIs pour réaliser les fonctionnalités souhaitées.

9.1 Les difficultés liées à l'utilisation des APIs Watson

L'utilisation des APIs Watson nous a créé de grosses difficultés durant la réalisation de cette application. Le fonctionnement de ces APIs est très difficile à comprendre, car ils touchent un domaine très complexe lié à l'intelligence artificielle. Les traitements cognitifs à travers des APIs REST, constituent un thème nouveau, c'est la raison laquelle, il n'existe pas beaucoup de tutoriels sur le net pouvant nous aider dans ce travail. À cela s'ajoute la mauvaise qualité des APIs « Speech To text » de manière générale et en particulier celui de Watson, que nous allons démontrer au point suivant.

9.1.1 La mauvaise qualité de l'API « Speech To text »

D'après nos expériences faites durant ce travail sur les APIs « Speech To text » qui existent sur le marché, on peut affirmer sans aucun doute qu'aujourd'hui on est très loin d'avoir un service capable de transcrire fidèlement la voix en texte. Ces services ont beaucoup de difficultés à transcrire la voix à cause :

- du bruit autour de la voix
- des accents multiples et difficiles à gérer
- des termes qui ne respectent aucune logique comme les noms de personnes, de lieux, etc.

Cependant il est aujourd'hui possible de personnaliser l'API « Speech To Text » de Watson pour améliorer le résultat de la transcription surtout avec les dates, mais ce service n'est disponible qu'en anglais, espagnol et japonais.

Pour illustrer la mauvaise qualité de l'API « Speech To text » de Watson, nous avons testé ce service en ligne et sur SmartAgenda avec la phrase vocale suivante : « **créer un meeting demain matin entre Omar Lo et Philippe Dugerdil à l'école** ». Mais les résultats obtenus sont loin d'être satisfaisants. On peut le voir sur les deux figures suivantes les résultats obtenus respectivement :

Figure 34 : Résultat obtenu avec l'API « Speech To text » de Watson en ligne

Voice Model: French broadband model (16KHz) Keywords to spot: créer un meeting demain entre, créer un rendez-vous lundi entre, à l'é

☐ Detect multiple speakers (Not supported on current model)

Record Audio Upload Audio File Play Sample 1 Play Sample 2

Text Word Timings and Alternatives Keywords (0/4) JSON

Qu' ou à l' il a l air de rien à. Un bon. De quoi.

Disponible à l'adresse :

https://speech-to-text-demo.ng.bluemix.net/?cm_mc_uid=18482519226615143151166&cm_mc_sid_50200000=1514918072&cm_mc_sid_52640000=1514918072

Figure 35 : Résultat obtenu avec SmartAgenda

Fenêtre d'enregistrement de rendez-vous

lignes rouges ont en revanche les gens il a accordé

Extraire les données du rendez-vous Vider Saisie Texte Saisie Vocale

Comme on peut le voir sur ces deux démos, l'API Watson « Speech To Text » a de grandes difficultés pour transcrire fidèlement la voix. Ce résultat est quasiment le même avec les autres APIs de transcription de la voix qui existent aujourd'hui sur le marché.

9.2 Les difficultés liées à l'utilisation de EWS-JAVA-API

Les difficultés liées à l'utilisation de ce service étaient surtout au niveau la communication entre notre application et le serveur Exchange de la HEG. Ce dernier n'était pas configuré pour accepter des communications avec des clients comme le nôtre. Il nous a fallu beaucoup de temps pour identifier ce problème. Une fois le problème identifié, nous avons soumis au service informatique de la HES, une demande afin que d'autoriser la communication entre le serveur Exchange de la HEG et notre application.

On peut aussi noter parmi les difficultés rencontrées avec l'utilisation de ce service, le fait que la documentation complète de cet API est fournie en langage `c#` qui était aussi un champ nouveau pour nous.

10. Conclusion

La réalisation de ce travail avec les technologies nouvelles sur le marché de Watson et les technologies de Microsoft Exchange server a été un complément important de ma formation en informatique à la HEG.

En effet les outils liés à l'intelligence artificielle sont aujourd'hui incontournables dans le domaine de l'informatique.

J'ai toujours souhaité à la fin de ma formation, réaliser un outil informatique en mettant en œuvre les compétences que j'ai acquises tout au long de mon cursus à la HEG.

SmartAgenda a été une expérience très enrichissante car elle m'a permis de résoudre un besoin réel identifié avec le mandat en apportant une solution sur mesure. Cette solution permet d'automatiser les processus de création de rendez-vous entre des collègues de travail d'une institution. Cette application peut être plus intéressante dans le futur si les améliorations suivantes sont faites :

- Améliorer les qualités de la transcription de l'audio en injectant la liste des prénoms et noms des contacts du serveur dans les mots clés de l'API « Speech To Text ». De ce fait au moment de l'enregistrement audio, on a besoin de passer juste les prénoms des participants. Car il est plus facile pour l'API « Speech To Text » de transcrire les prénoms. Pour la transcription des noms, il passe souvent à côté.
- Développer une version mobile de cette application de telle sorte qu'elle soit plus accessible

Bibliographie

Catalogue Watson. *Catalogue-IBM Cloud* [en ligne]. [Consulté le 10 juillet 2017]. Disponible à l'adresse : <https://console.bluemix.net/catalog/?taxonomyNavigation=apps&category=watson>

Documentation IBM Cloud / Conversation. *IBM Cloud Docs* [en ligne]. Dernière modification de la page le 27 juillet 2017. [Consulté le 9 août 2017]. Disponible à l'adresse : <https://console.bluemix.net/docs/services/conversation/index.html#about>

Prise en main des applications clientes EWS. *Microsoft Office | Dev Center* [en ligne]. Dernière modification de la page le 19 juin 2014. [Consulté le 10 juillet 2017]. Disponible à l'adresse : [https://msdn.microsoft.com/fr-fr/library/office/dn789003\(v=exchg.150\).aspx](https://msdn.microsoft.com/fr-fr/library/office/dn789003(v=exchg.150).aspx)

Processus cognitifs. *Wikipedia : l'encyclopédie libre* [en ligne]. Dernière modification de la page le 1 juillet 2017 à 14 :22. [Consulté le 2 août 2017]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/Processus_cognitifs

Un ordinateur nommé Watson. *IBM* [en ligne]. [Consulté le 6 juillet 2017]. Disponible à l'adresse : <http://www-03.ibm.com/ibm/history/ibm100/fr/fr/icons/watson/>

Java SDK to use the IBM Watson services. *watson-developer-cloud / java-sdk sur GitHub* [en ligne]. [Consulté le 5 juillet 2017]. Disponible à l'adresse : <https://github.com/watson-developer-cloud/java-sdk>

A java client library to access Exchange web services. *OfficeDev / ews-java-api sur GitHub* [en ligne]. [Consulté le 10 juillet 2017]. Disponible à l'adresse : <https://github.com/OfficeDev/ews-java-api/wiki/Getting-Started-Guide>

Annexe 1 : Accès et utilisation des APIs Watson

Table des matières

1. Introduction.....	58
2. Création d'un compte développeur sur IBM Cloud	59
2.1 Accès aux APIs Watson	60
3. Utilisation des APIs Watson	60
3.1 API Watson « Conversation »	60
3.1.1 Construction du service « Conversation » sur Buemix	61
3.1.2 Test en ligne du service « Conversation » créé sur Bluemix.....	74
3.1.3 Utilisation de l'API « Conversation » dans le code Java.....	78
3.2 API Watson « Speech To text ».....	83
3.3 API Watson « Natural Language Understanding »	85
4. Configuration du SDK de Watson dans notre projet Java.....	87

1. Introduction

Pour accéder et utilisation des APIs Watson, il faut d'abord se rendre sur IBM Cloud pour créer son propre compte.

2. Création d'un compte développeur sur IBM Cloud

Pour cela, il faut se rendre sur l'adresse suivante : <https://console.bluemix.net> et cliquez sur « Créer un compte gratuit », comme le montre la figure suivante :

Figure 36 : Page d'accueil pour créer un compte sur Bluemix



(IBM Cloud)

Une fois le compte créé, on peut voir les informations suivantes :

Figure 37 : Information de mon compte sur Bluemix

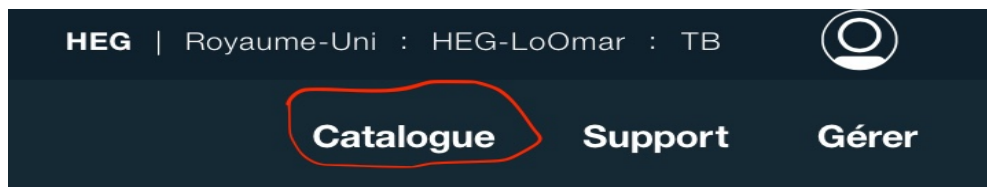


2.1 Accès aux APIs Watson

Pour accéder aux services, il faut se connecter sur notre compte Bluemix précédemment créé. Voici la démarche à suivre après la connexion :

- 1) **Affichage de la liste de tous les services** : En haut et à droite sur la barre des tâches de la page d'accueil, cliquer sur **"Catalogue"**, comme le montre la figure suivante :

Figure 38 : Page d'accueil de mon compte sur Bluemix



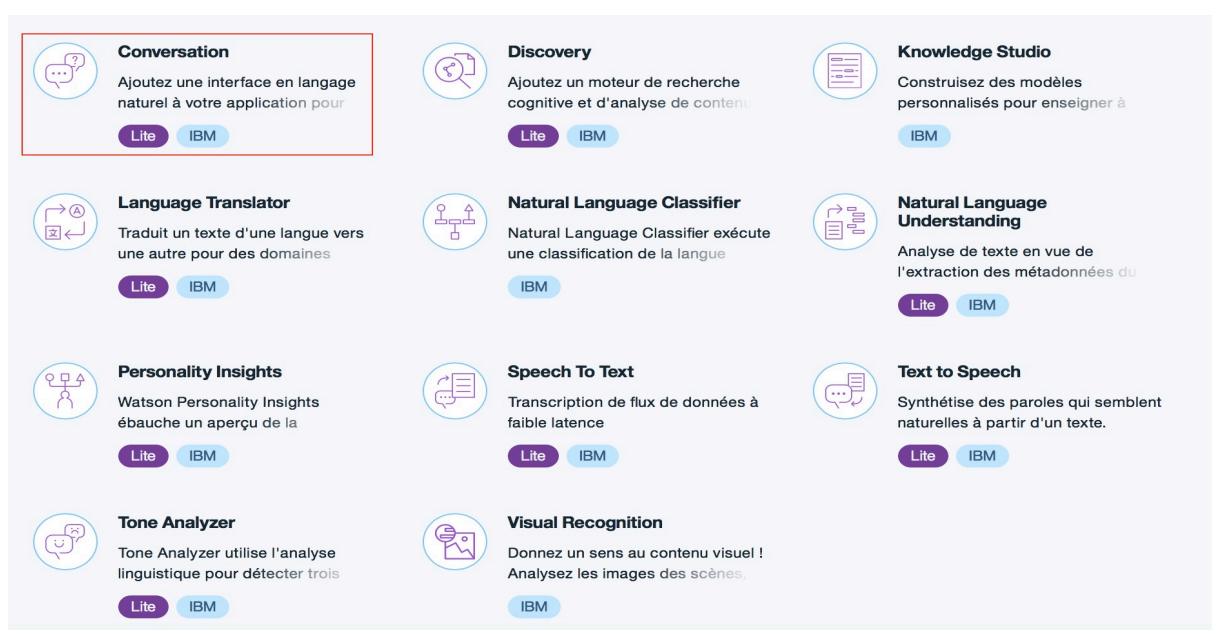
- 2) **Affichage des services Watson** : À gauche sur la liste des menus de cette page, cliquer sur **"Watson"** pour apercevoir la liste complète des APIs Watson, comme on peut le voir sur la figure 1.

3. Utilisation des APIs Watson

3.1 API Watson « Conversation »

Pour utiliser l'API Watson « Conversation », on doit aller sur la liste complète des APIs Watson sur notre compte Bluemix, comme expliqué au point « 2.1 1) et 2) » de la section « **Accès aux APIs Watson** » et cliquer sur le service de notre choix, comme le montre la figure suivante :

Figure 39 : Sélection de l'API Watson « Conversation »



3.1.1 Construction du service « Conversation » sur Buemix

Voici la démarche à suivre pour construire un service « Conversation » de Watson sur Buemix :

- 1) **Renommer le nom du service ou garder les valeurs par défaut** : Ici j'ai gardé les valeurs par défaut, comme le montre la figure qui suit :

Figure 40 : Page de création du service « Conversation »

The screenshot shows the 'Conversation' service creation page. On the left, there is a descriptive text about adding a natural language interface. The main form contains the following fields:

- Nom du service :** Conversation-62
- Nom des données d'identification :** Credentials-1
- Sélectionnez une région dans laquelle effectuer le déploiement :** Royaume-Uni
- Choisissez une organisation :** HEG-LoOmar
- Choisissez un espace :** TB
- Connecter à :** Laisser non lié

At the bottom, there are links for 'Afficher la documentation', 'Vous avez besoin d'aide ?', 'Prenez contact avec le service commercial Bluemix', 'Estimer le coût mensuel', 'Calculateur de coût', and a 'Créer' button.

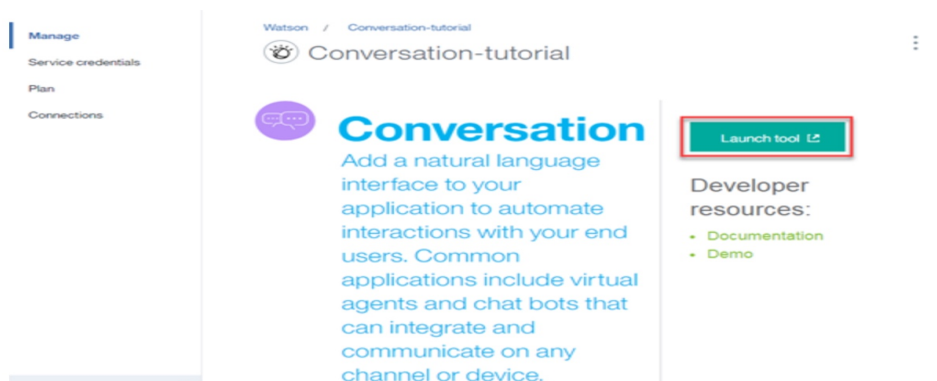
- 2) **Cliquer sur « Créer » pour créer le service** : Après avoir cliqué sur ce bouton, on arrive sur la page d'accueil du service, comme on peut le voir sur la figure suivante :

Figure 41 : Page de paramétrage de l'API Watson « Conversation »

The screenshot shows the Watson Conversation service dashboard. On the left, there is a sidebar with links: 'Gérer', 'Données d'identification pour le service', 'Plan', and 'Connexions'. The main area displays the service name 'Conversation-ck' and a description: 'Add a natural language interface to your application to automate interactions with your end users. Common applications include virtual agents and chat bots that...'. There is a 'Launch tool' button and a 'Developer resources' section with links to 'Documentation' and 'Demo'.

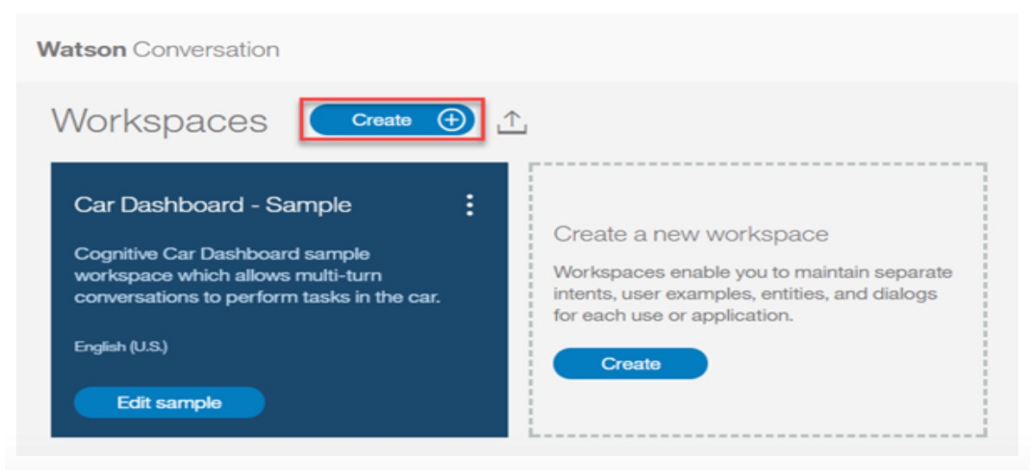
- 3) **Paramétrage du service** : Pour paramétrer le service « Conversation » de Watson, il faut cliquer sur « Launch tool » qui est à droite de la page d'accueil du service, comme le montre la figure suivante :

Figure 42 : Clic sur « Launch tool » pour paramétrer « Conversation »



4) Créer un Workspace pour héberger notre dialogue : C'est un contenir qui permet de référence notre service « Conversation » créé sur Bluemix. Autrement dit le Workspace que nous créons, aura un id qui est unique sur Bluemix. Cet id unique nous permet d'appeler le service créé à distance, c'est-à-dire depuis notre code Java. Pour créer un Workspace, on clique sur « Create », comme le montre la figure suivante :

Figure 43 : Création d'un Workspace



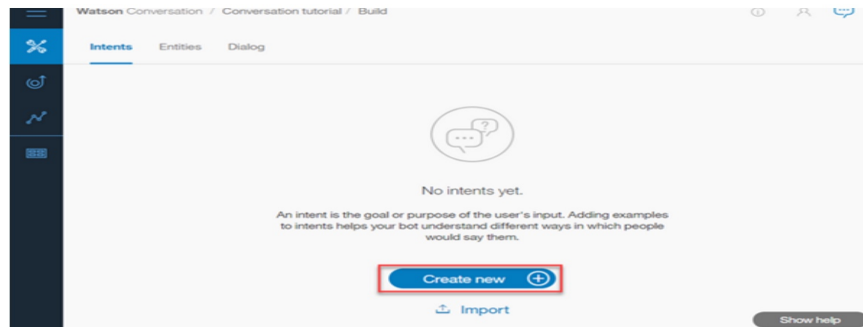
5) Configuration du Workspace : Pour cela on donne un nom à ce Workspace et on choisit la langue du dialogue, comme le montre la figure qui suit :

Figure 44 : Configuration de notre Workspace

The screenshot shows the 'Create a workspace' form. It includes a sub-header 'Create a workspace' and a descriptive sentence. There are three input fields: 'Name' with the value 'TB-omar', 'Description' (empty), and 'Language' with a dropdown menu set to 'French'. A red box highlights the 'Name' field. At the bottom right, there is a blue 'Create' button.

6) Créer les Intents : Les Intents sont les intentions de notre dialogue. Nous allons expliquer dans le point suivant pourquoi il faut créer les Intents. On peut avoir plusieurs intents dans une même application. Pour créer l'intent, il faut cliquer sur « Create new », comme on peut le voir sur la figure suivante :

Figure 45 : Création d'un intent



7) Donner un nom à l'intent : Le nom de l'intent doit avoir un sens dans l'application qu'on va développer. Dans notre cas, le but est de créer une application de prise de rendez-vous automatique. Nous avons donc nommé notre intent « #meeting », comme le montre la figure suivante :

Figure 46 : Configuration de l'intent



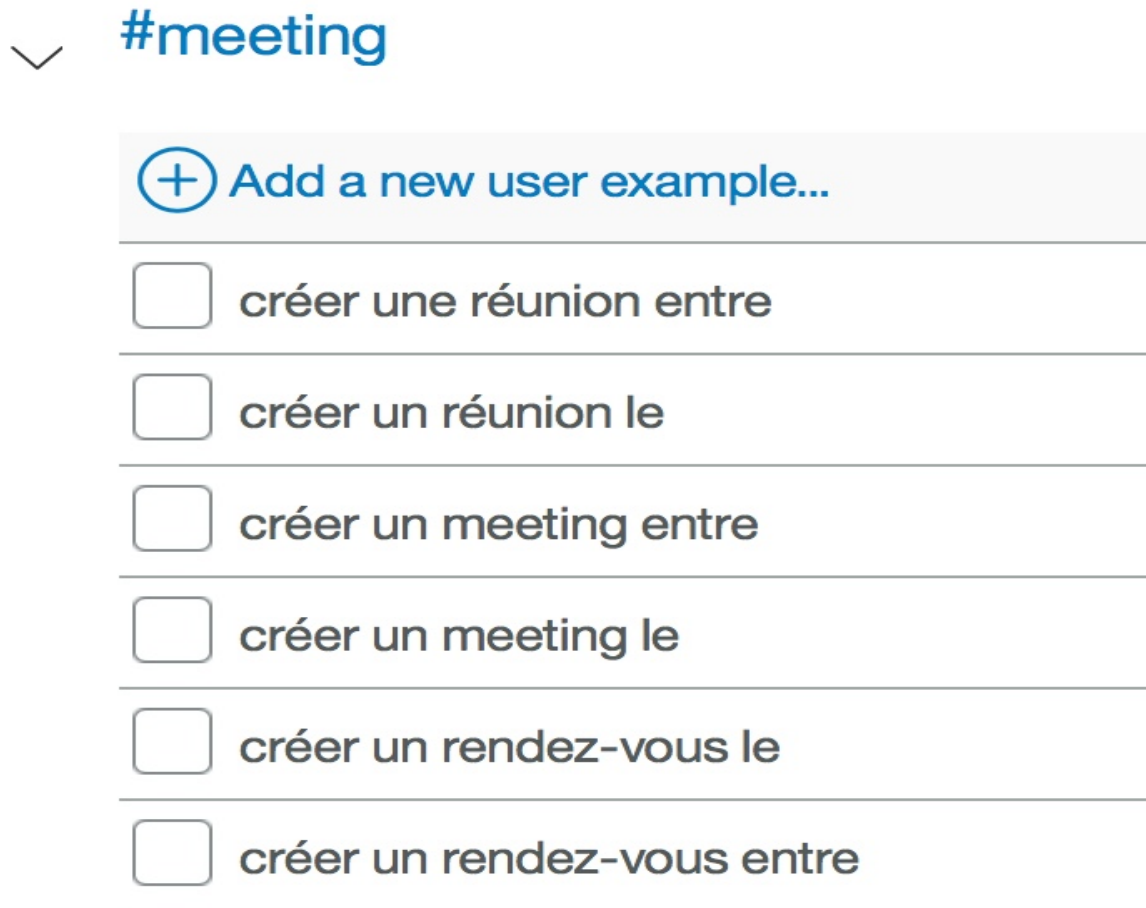
8) Ajouter des débuts de phrase du dialogue avec l'utilisateur (user example) : Pour dialoguer avec la machine, l'utilisateur doit tout le temps utiliser ces débuts de phrases injectés dans l'intent. Pour ajouter un début de phrase, il faut cliquer sur le symbole « + » à droite, comme le montre la figure suivante :

Figure 47 : Ajout d'un début de phrases dans l'intent



Pour un intent, on peut avoir plusieurs débuts de phrase. Dans notre cas on a créé plusieurs débuts de phrase pour élargir le champ du dialogue, comme on peut le voir sur la figure suivante :

Figure 48 : Liste de nos intents



The image shows a user interface for managing intents. At the top, there is a blue header with a downward arrow and the text '#meeting'. Below this, there is a light gray button with a plus icon and the text 'Add a new user example...'. Underneath, there is a list of six intents, each preceded by a square checkbox:

- ☐ créer une réunion entre
- ☐ créer un réunion le
- ☐ créer un meeting entre
- ☐ créer un meeting le
- ☐ créer un rendez-vous le
- ☐ créer un rendez-vous entre

9) Enregistrement de l'intent : Pour enregistrer l'intent nouvellement créé, il faut cliquer sur « Done », comme nous pouvons le voir sur la figure suivante :

Figure 49 : Enregistrement de l'intent « meeting »

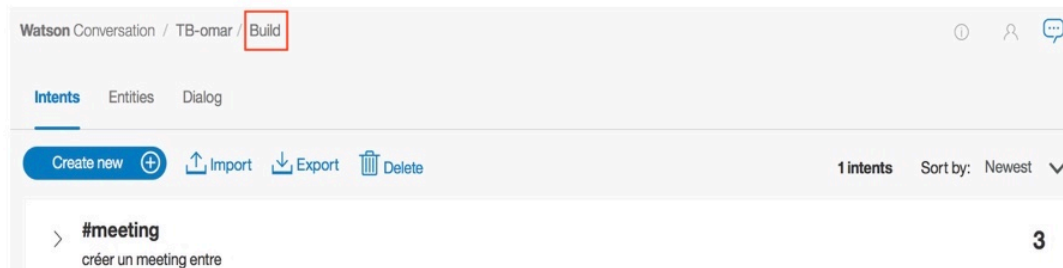


The image shows a user interface for managing intents. At the top, there is a header with three tabs: 'Intents', 'Entities', and 'Dialog'. The 'Intents' tab is selected. Below the tabs, there is a text input field labeled 'Intent name' containing the text '#meeting'. To the right of the input field, there is a blue button with the text 'Done'.

Une fois qu'on a enregistré l'intent, toutes les modifications sont alors prises en compte par la machine.

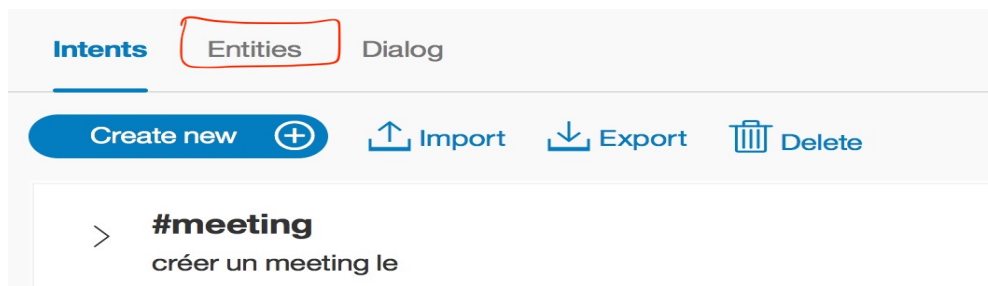
- 10) Entraîner la machine** : On doit maintenant entraîner la machine avec les nouveaux intents pour permettre reconnaître les intentions du dialogue avec les utilisateurs de notre système. Pour cela, on doit cliquer sur « Build », comme nous pouvons le constater sur la figure suivante :

Figure 50 : Entraînement de la machine avec les nouveaux intents



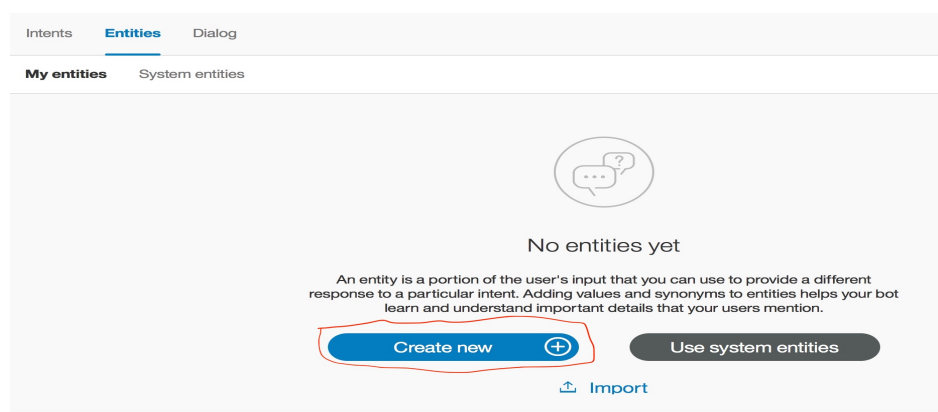
- 11) Ajout des « Entities »** : Les « Entities » sont les entités qui sont manipulées dans le dialogue avec la machine. Ces entités peuvent être un lieu, des produits, etc. Pour créer une entité, on doit cliquer sur « Entities », comme on peut le constater sur la figure suivante :

Figure 51 : Ajout des entité pour le dialogue



- 12) Créer un entité** : Dans ce travail, nous a besoin de trois entities pour le lieu du rendez-vous. Ces entities sont : la date, l'heure et le lieu du rendez-vous. Cependant nous avons créé seulement l'entité « lieu » car l'entité « date » et l'entité « time » sont déjà inclus par défaut dans le système. Pour créer un nouvel entitie, il faut cliquer sur « Create new », comme le montre la figure qui suit :

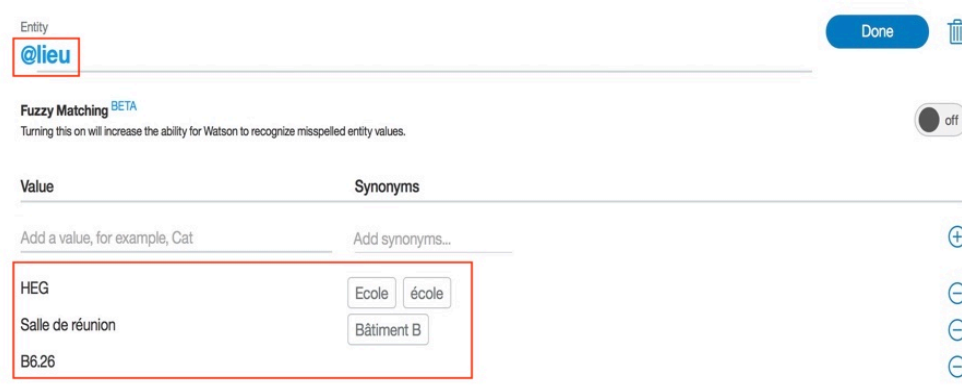
Figure 52 : Création d'entité « lieu » pour le dialogue



13) Donner un nom à l'entité créé et lui fournir les différentes values souhaitées :

Chaque entité créée doit avoir un certain nombre de values. Ces values seront les différents termes liés à une entité que la machine va repérer dans le texte de l'utilisateur. On peut aussi attribuer à chaque value, un certain nombre de synonymes. Dans notre cas nous avons créé une entité « #lieu » avec un certain nombre de values et de synonymes, comme nous pouvons le constater sur la figure suivante :

Figure 53 : Configuration de l'entité « lieu » pour le dialogue



14) Activer les modifications : Pour permettre à la machine de prendre en compte les nouvelles valeurs de l'entité nouvellement créé, on doit activer le matching, comme on peut le voir sur la figure suivante :

Figure 54 : Activation du matching

Entity
@lieu

Fuzzy Matching BETA
Turning this on will increase the ability for Watson to recognize misspelled entity values.

Value **Synonyms**

Value	Synonyms	
Add a value, for example, Cat	Add synonyms...	+
HEG	Ecole école	-
Salle de réunion	Bâtiment B	-
B6.26		-

15) Enregistrement de l'entité : Pour enregistrer l'entité nouvellement créé, il faut cliquer sur « Done », comme nous pouvons le constater sur la figure suivante :

Figure 55 : Enregistrement de l'entité « lieu »

Entity
@lieu

Fuzzy Matching BETA
Turning this on will increase the ability for Watson to recognize misspelled entity values.

Value **Synonyms**

Value	Synonyms	
Add a value, for example, Cat	Add synonyms...	+
HEG	Ecole école	-
Salle de réunion	Bâtiment B	-
B6.26		-

Une fois qu'on a enregistré l'entité, toutes les modifications sont alors prises en compte par la machine. Après l'enregistrement on obtient le résultat suivant :

Figure 56 : Entité « lieu » enregistré dans le système

Intents **Entities** Dialog

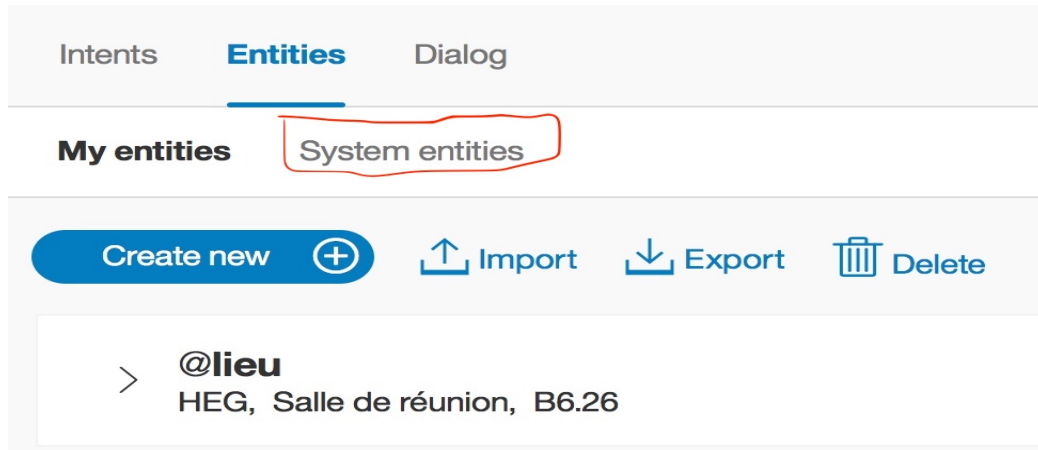
My entities System entities

Create new (+) Import Export Delete

> **@lieu**
HEG, Salle de réunion, B6.26

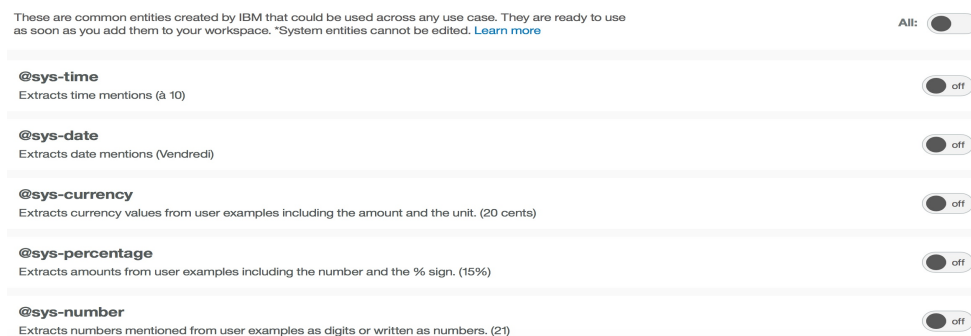
Comme nous l'avons précédemment, il existe des entités par défaut dans le système, appelés « System entities » que nous pouvons réutiliser dans notre dialogue. Pour voir ces entités, il faut cliquer sur « System entities », comme on peut le voir sur la figure suivante :

Figure 57 : System entities



Après avoir cliqué sur « System entities », on obtient la liste des entités du système, comme le montre la figure suivante :

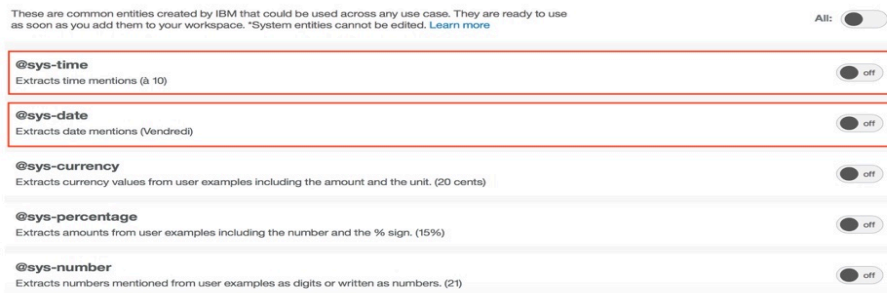
Figure 58 : Liste des entités du système



Par défaut, tous ces entités sont désactivés. On peut soit les activer tous, soit sélectionner juste ceux qui nous intéressent. Dans notre cas on a juste sélectionné l'entité time et l'entité date qui permet à la machine de chercher dans le texte de l'utilisateur la date et l'heure du rendez-vous et de nous les retourner.

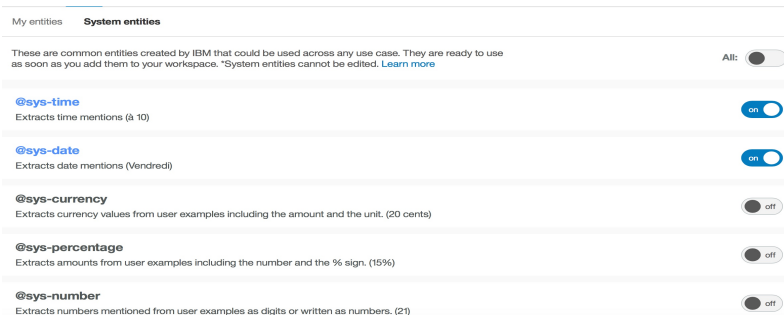
16) Sélection de quelques entités du système : Nous allons juste choisir l'entités time et date du système, comme le montre la figure suivante :

Figure 59 : Sélection de l'entité time et date du système



17) **Activation des entités du système choisis** : La figure suivante nous montre le résultat de deux entités activées :

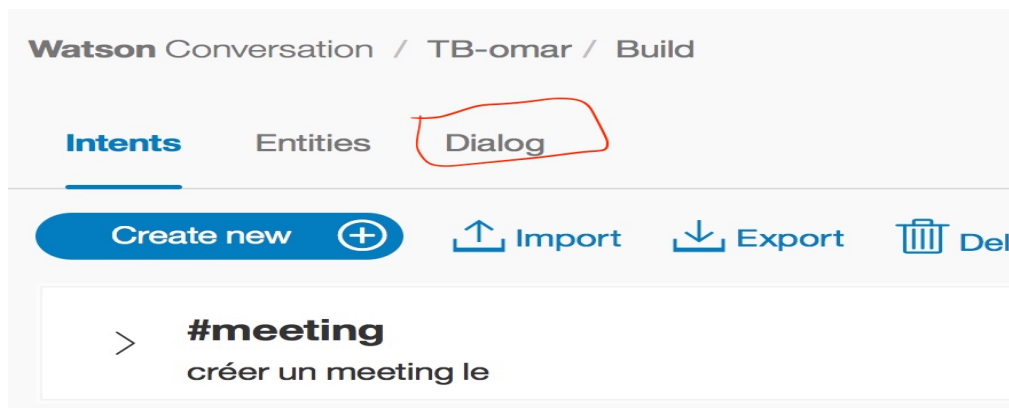
Figure 60 : Activation de l'entité time et date du système



18) **Entraîner encore la machine** : Faites la même chose qu'au point 10), c'est-à-dire cliquer sur « Build » pour permettre à la machine de s'entraîner avec les nouveaux éléments du dialogue nouvellement enregistrés dans le système.

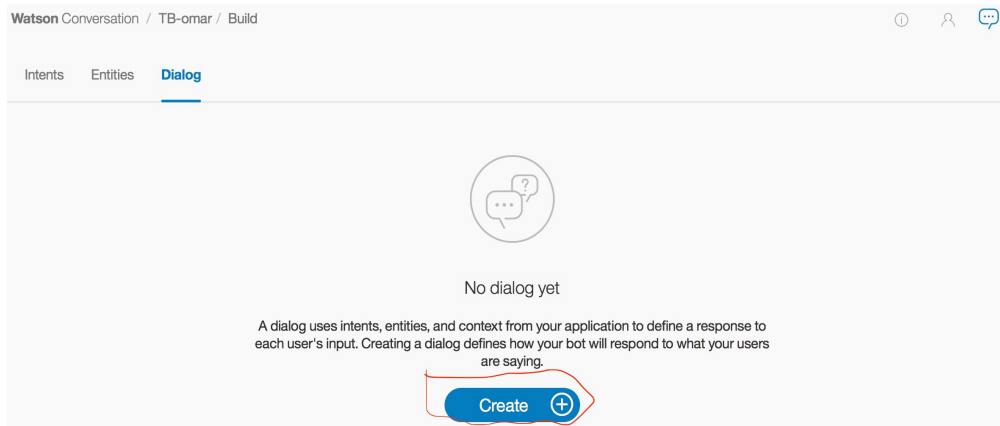
19) **Construction du dialogue dans le système** : Il nous faut à présent construire le dialogue proprement dit avec la machine. Pour cela on doit cliquer sur « Dialog », comme le montre la figure suivante :

Figure 61 : Construction d'un dialogue dans le système



20) **Créer un nouveau dialogue** : Pour cela, il faut cliquer sur « Create », comme le montre la figure suivante :

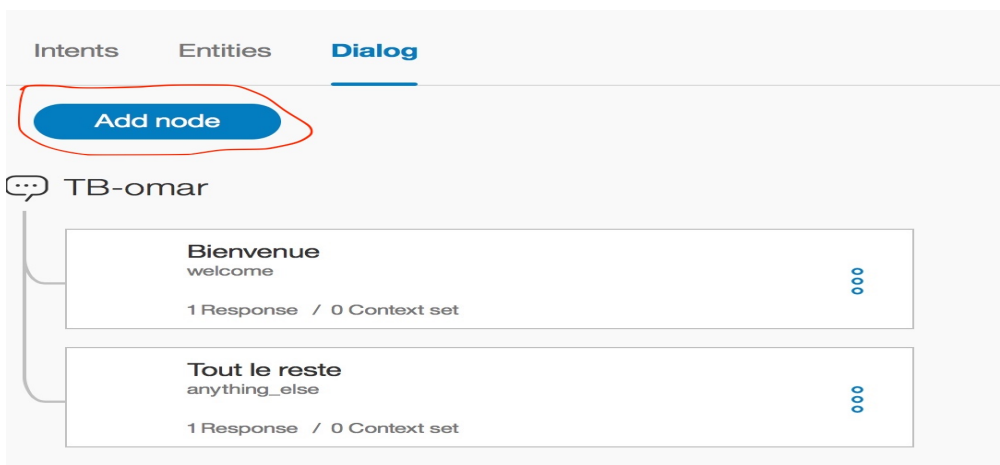
Figure 62 : Créer un dialogue dans le système



Chaque dialogue créé, doit avoir au moins un ou plusieurs nœuds. Chaque nœud correspond à un intent (intention du dialogue) créé. Dans notre cas on a juste besoin d'un nœud qui va correspondre à notre intent « #meeting ».

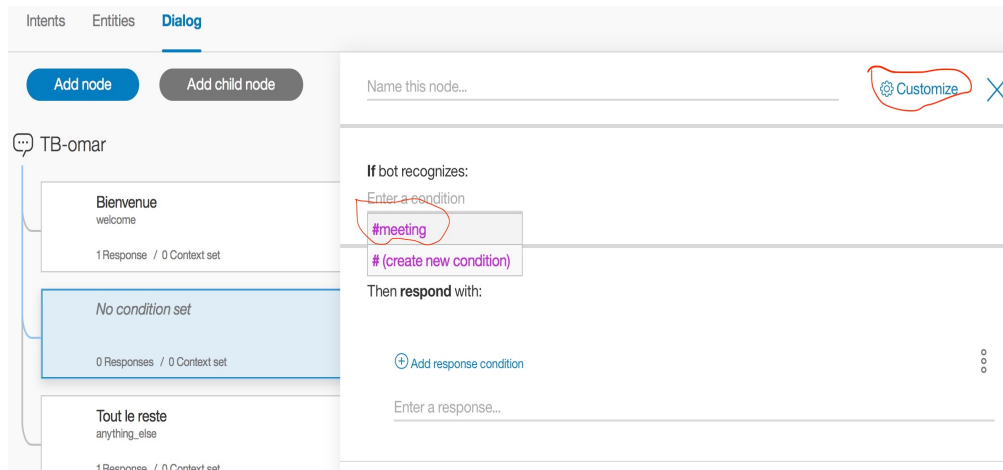
21) Ajout d'un nœud à notre dialogue : Pour ajouter un nouveau nœud à notre dialogue, on doit cliquer sur « Add node », comme le montre la figure suivante :

Figure 63 : Créer un nouveau nœud



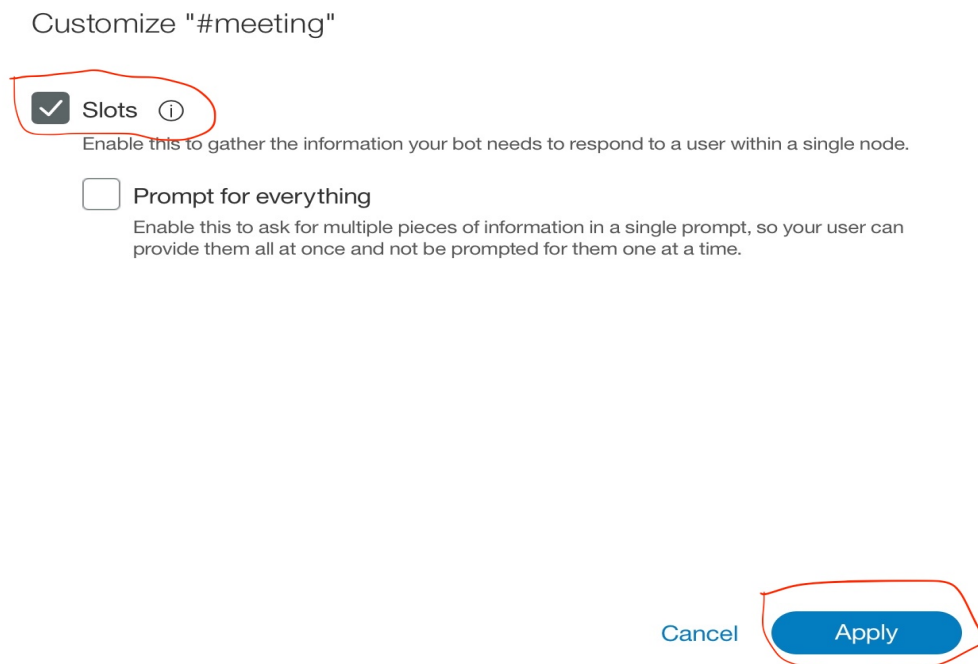
22) Configuration du nœud : pour configurer un nœud, il faut lui assigner un intent et cliquer sur « Customize ». Dans notre cas nous avons assigné à notre nœud l'intent « #meeting », comme le montre la figure qui suit :

Figure 64 : Configuration du nœud



23) Cocher la case « Slots » et enregistrer : Cocher la case « Slots » permet à la machine de tenir compte des informations du nœud lors du dialogue avec l'utilisateur. Pour cela il faut se référer sur la figure suivante :

Figure 65 : Enregistrement de notre nœud



À présent, on peut construire les conditions du dialogue entre la machine et l'utilisateur du système. Pour cela on doit récupérer nos trois entités (date, heure et lieu) pour permettre au système de les rechercher dans le texte de l'utilisateur. Si une entité est absente dans le texte de l'utilisateur, la machine doit pouvoir demander à l'utilisateur de lui fournir cette entité pour faire son travail. C'est la raison pour laquelle, chaque entité est associée à une question.

24) Configuration de notre dialogue : Pour cela, on choisit le bon nœud correspondant à l'intent et remplir les champs vides. Ces champs correspondent à chaque entité, sa valeur et la question qui lui est associé, comme le montre la figure suivante :

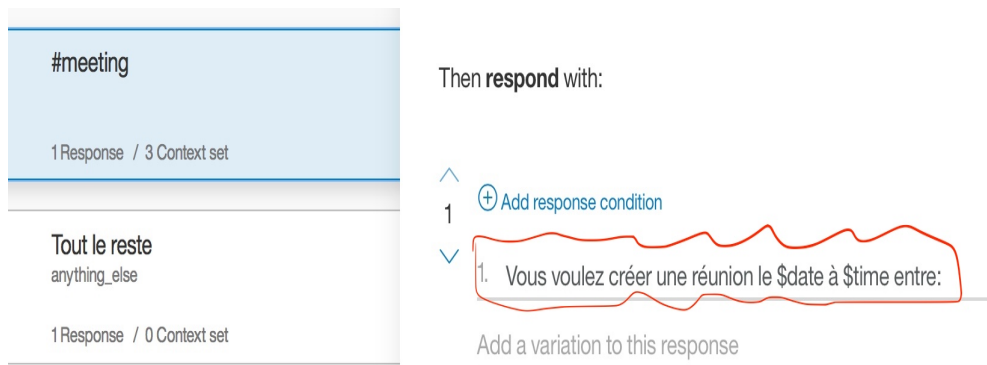
Figure 66 : Configuration du dialogue

Une fois que ces différents champs remplis, on obtient un résultat semblable à la figure suivante :

Figure 67 : Configuration du dialogue

25) Construire la réponse du système : Une fois qu'on a fini de remplir les différentes conditions du dialogue, on doit construire la réponse définitive du système si toutes les conditions sont réunies dans le texte de l'utilisateur, comme le montre la figure qui suit :

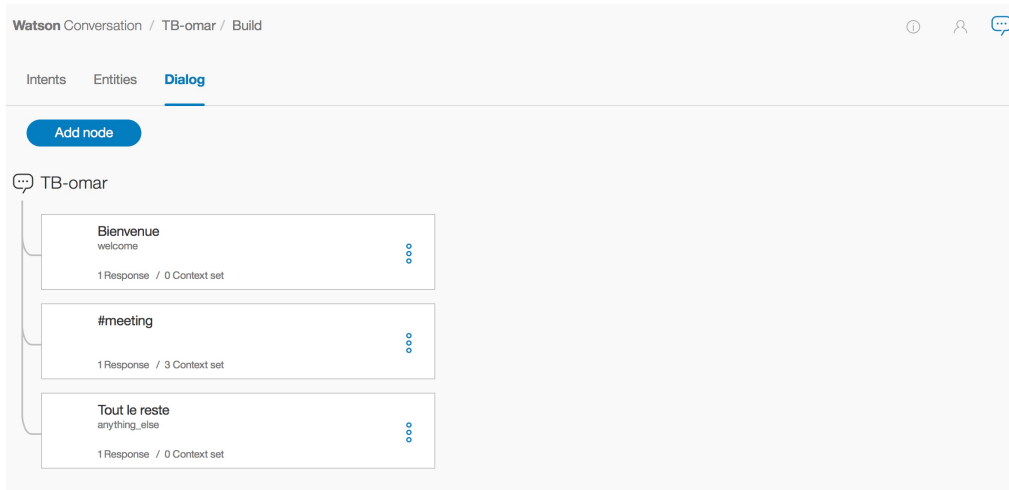
Figure 68 : Configuration du dialogue



26) Entraîner encore la machine : Faites la même chose qu'au point 10), c'est-à-dire cliquer sur « Build » pour permettre à la machine de s'entraîner avec les nouveaux éléments du dialogue nouvellement enregistrés dans le système.

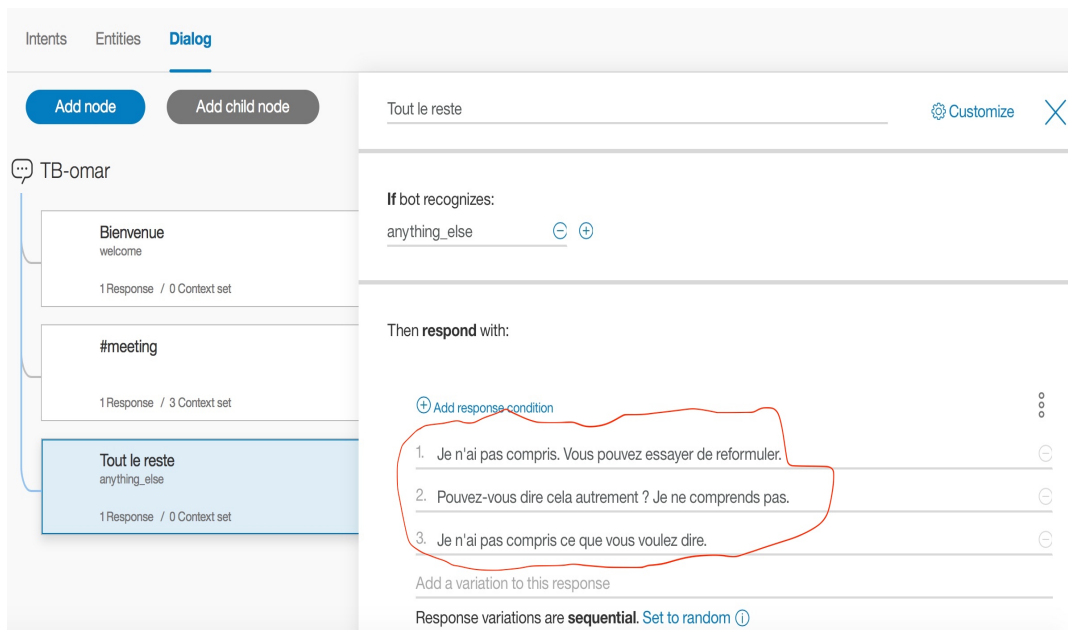
Après avoir entraîné la machine, on obtient le résultat suivant sur cette figure :

Figure 69 : Liste des nœuds du dialogue



Notre dialogue est composé de trois nœuds. Le nœud de « Bienvenue » qui contient la phrase de bienvenue que le système affiche dès que l'utilisateur ouvre la boîte de dialogue. Le nœud « #meeting » qui correspond à notre intent et le nœud « Tout le reste ». Le nœud « Tout le reste » contient toutes les phrases que le système doit répondre à l'utilisateur si elle ne comprend pas sa demande. Les nœuds « Bienvenue » et « Tout le reste » peuvent être personnalisés, comme le montre la figure suivante correspondant à la personnalisation du nœud « Tout le reste ».

Figure 70 : Personnalisation du nœud « Tout le reste »



Nous avons à présent terminé la construction notre dialogue avec le service « Conversation » de Watson. On peut maintenant le tester soit en ligne ou dans notre code Java.

3.1.2 Test en ligne du service « Conversation » créé sur Bluemix


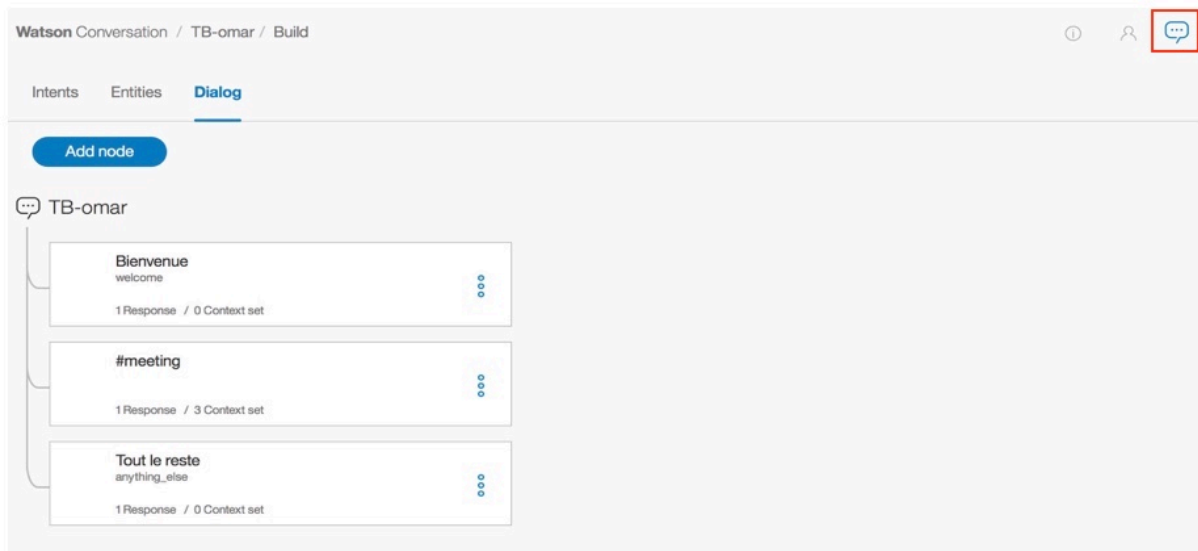
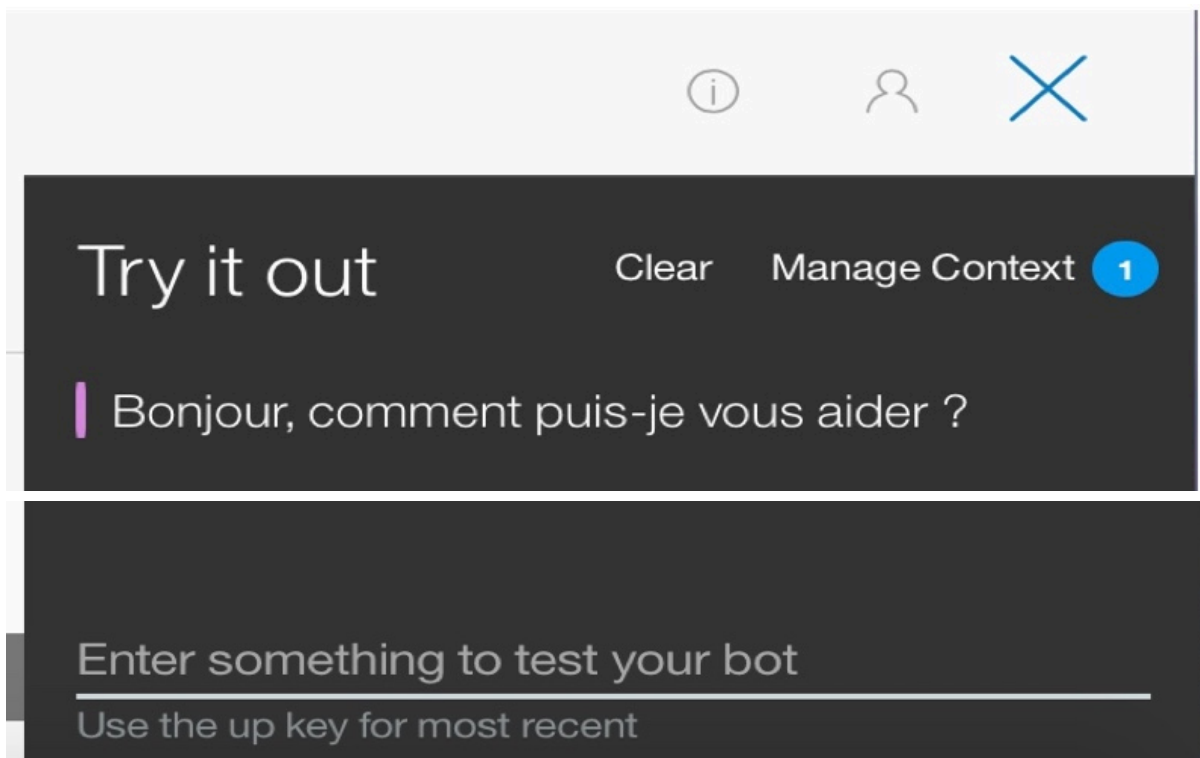
Pour tester le service « Conversation » que nous avons créé sur notre compte Bluemix, il faut cliquer sur cette icône , comme le montre la figure suivante :

Figure 71 : Lancement du dialogue en ligne



Une fois qu'on a cliqué l'icône, la boîte de dialogue s'ouvre, comme nous pouvons le voir sur la figure suivante :

Figure 72 : Page d'accueil du dialogue en ligne

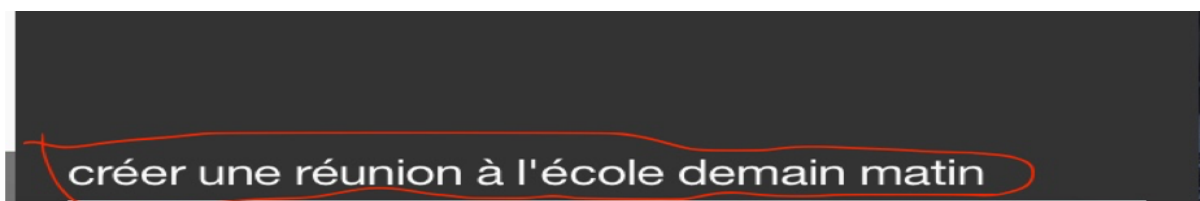


On peut à présent dialoguer avec le système via cette interface. On voit cet écran la phrase de Bienvenue qui correspond premier nœud de notre dialogue et un champ permettant de saisir le texte de l'utilisateur pour dialoguer avec le système.

Voici nos différents tests effectués sur ce service :

1) **Question 1** : créer une réunion demain matin à l'école.

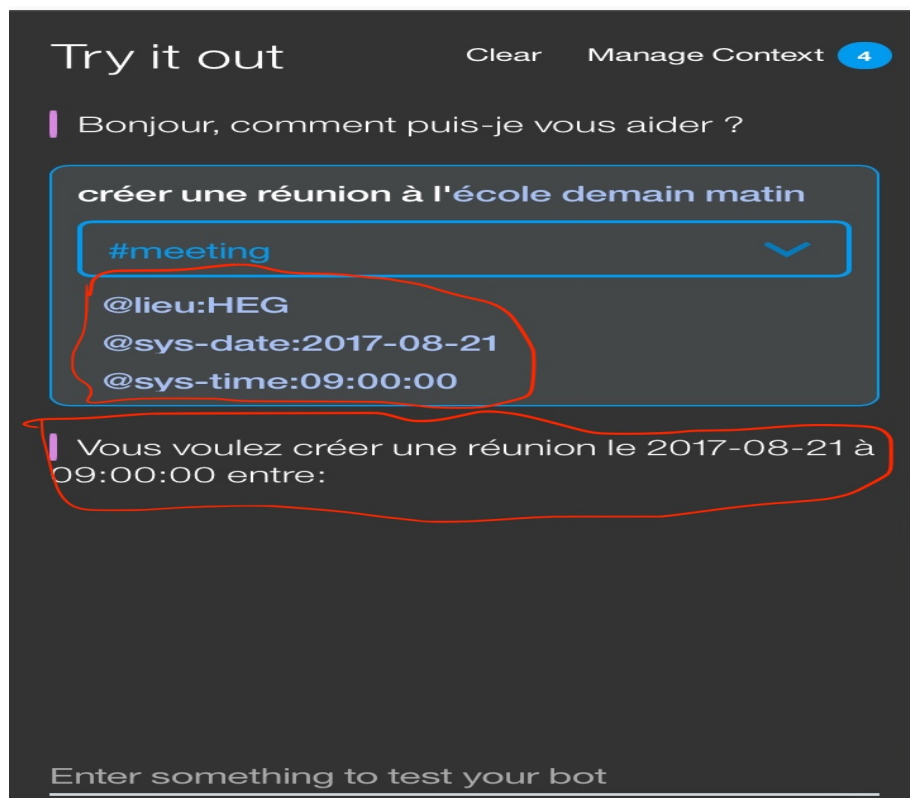
Figure 73 : Question1



La figure suivante montre la réponse du système. On voit sur cette figure que la machine a réussi à extraire la date, l'heure et le lieu à partir de notre texte et à nous fournir sa réponse définitive, comme nous pouvons le voir.

NB : le mot « **matin** » correspond à « **09 :00 :00** » pour le système et « **l'après-midi** » à « **18 :00 :00** ». Je n'ai aucune explication pour ce choix.

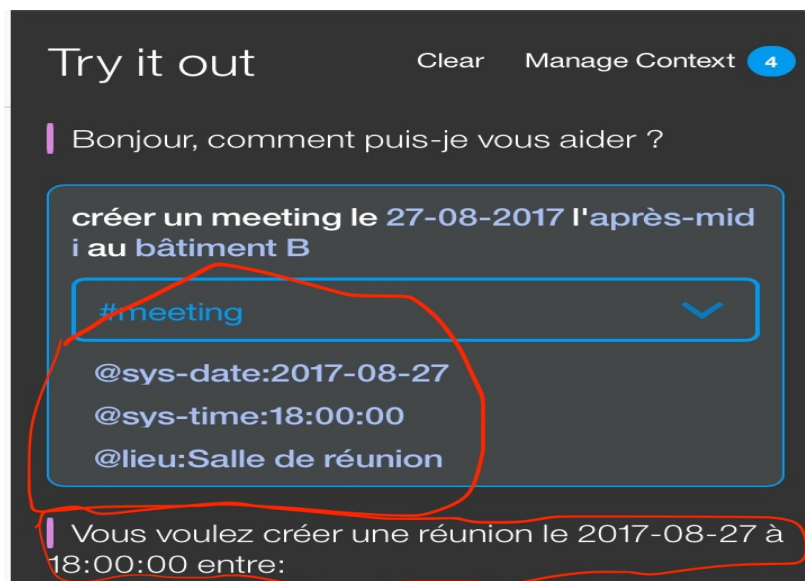
Figure 74 : Réponse du système à la question1



2) **Question 2** : Créer un meeting le 27-08-2017 l'après-midi au bâtiment B.

La figure suivante montre la réponse du système à cette question.

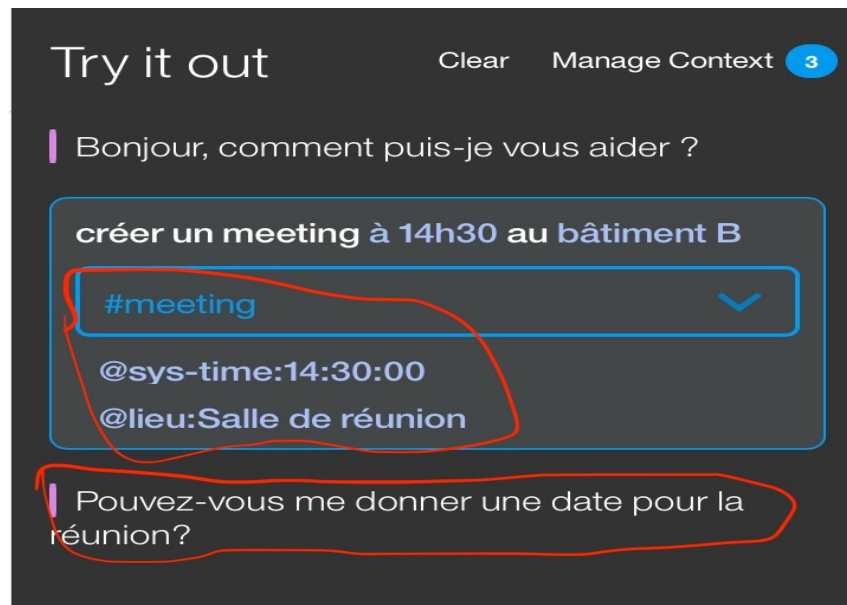
Figure 75 : Réponse du système à la question 2



3) **Question 3** : créer un meeting à 14h30 au bâtiment B.

La figure suivante montre la réponse du système à cette question.

Figure 76 : Réponse du système à la question 3

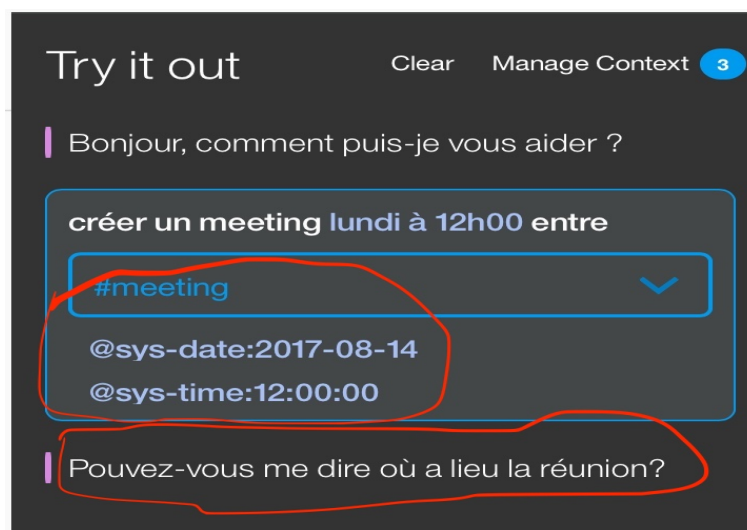


On voit que l'utilisateur n'a pas fourni la date du rendez-vous, du coup le système lui demande de fournir la date pour la réunion, comme nous pouvons le constater la figure 76.

4) Question 4 : créer un meeting lundi à 12h00 entre.

Ce test a été effectué le mercredi 16.08.2017 sur le système. La figure suivante montre la réponse du système à cette question.

Figure 77 : Réponse du système à la question 4



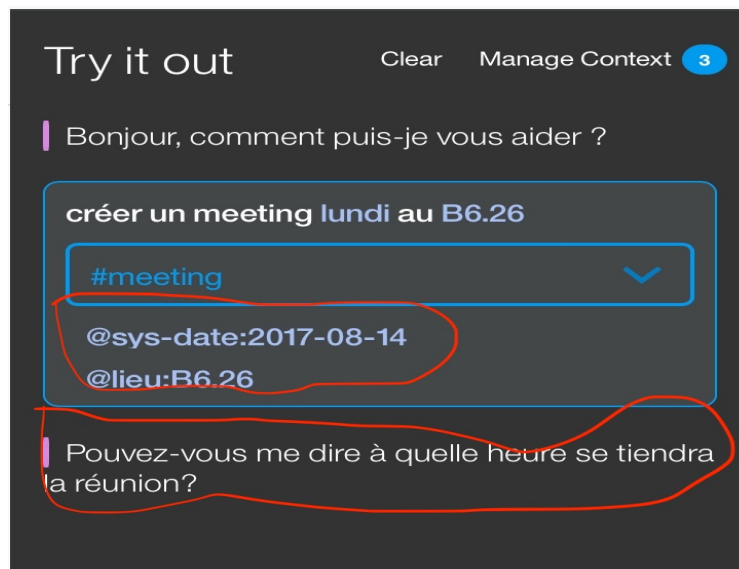
On voit ici que la machine se trompe de date, au lieu de prendre lundi de la semaine prochaine (21-08-2017), elle prend le lundi de la semaine dernière (14-08-2017). Mais c'est un cas très rare. Cependant ce bug nous l'avons résolu dans le code notre application. Nous avons à

chaque fois que la machine nous propose une date, la comparer avec la date du jour si elle est avant la date du jour, nous ajoutons à cette date 7 jours pour trouver la bonne date. Voir la classe **SmartAgendaDisplayInfoRv** : ligne 53 à 80.

5) Question 5 : créer un meeting lundi au B6.22.

La figure suivante montre la réponse du système à cette question.

Figure 78 : Réponse du système à la question 5



Ici aussi même constate, l'utilisateur n'a pas fourni l'heure du rendez-vous, du coup la machine lui demande de fournir pour la réunion.

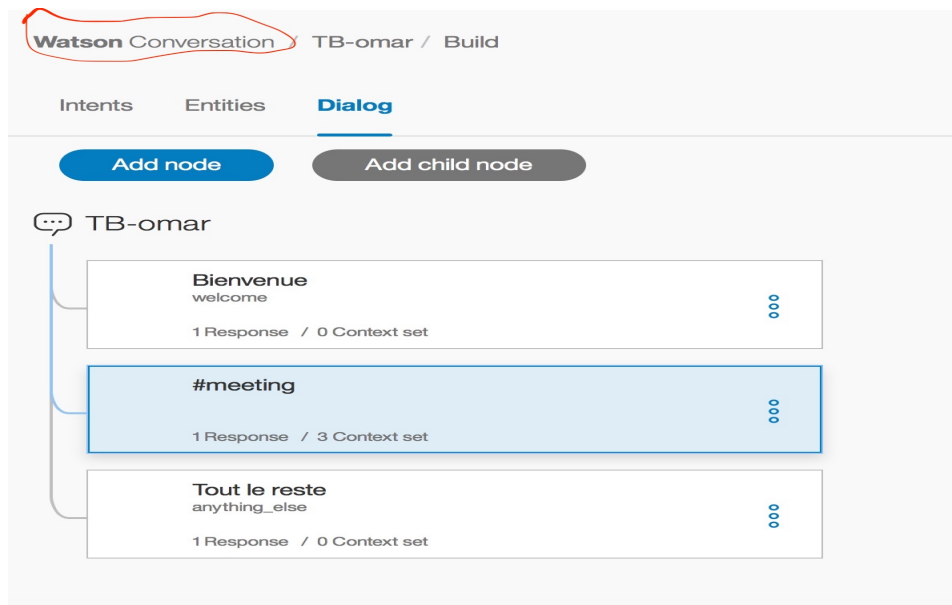
3.1.3 Utilisation de l'API « Conversation » dans le code Java

Pour implémenter ce service dans notre application SmartAgenda, nous avons besoin de l'id du Workspace comme expliqué précédemment. Pour obtenir cet id, voici la démarche à suivre :

1) Obtenir l'id du Workspace : Pour cela, on a deux choix.

- Soit aller dans la page de configuration du dialogue et cliquer sur « Watson Conversation », comme nous le montre la figure suivante :

Figure 79 : Page de configuration du dialogue



- Soit aller à la page d'accueil de notre compte Bluemix et sélectionner le service « Conversation » dans la liste de nos services créés, comme le montre la figure suivante :

Figure 80 : Liste de nos services Watson créés sur Bluemix

IBM Bluemix

Tableau de bord

Tous les services (9)

Création de service

Services 9/10 utilisé(s)

20 Items per page | 1-9 of 9 items

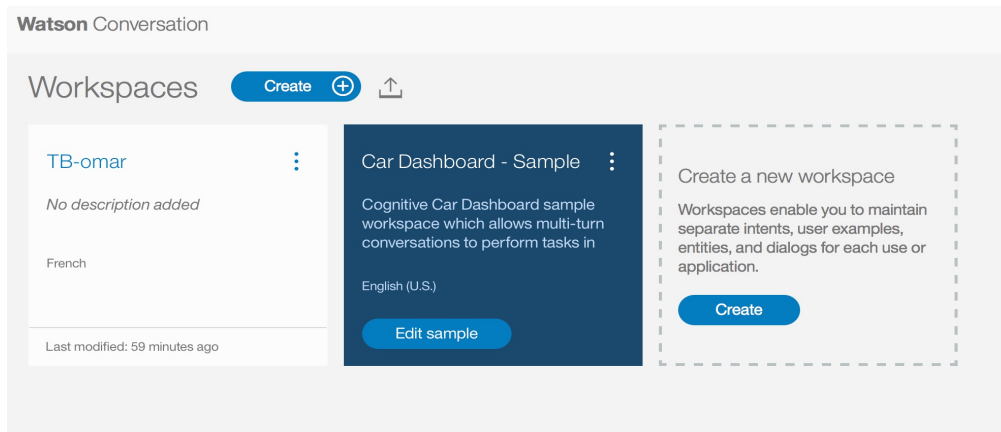
1 of 1 pages

NOM	OFFRE DE SERVICES	PLAN	ACTIONS
Conversation-ck	Conversation	Lite	<div></div>
Conversation-mtg	Conversation	Lite	<div></div>
Conversation-vq	Conversation	Lite	<div></div>
Natural Language Understanding-1k	Natural Language Understanding	Lite	<div></div>
Natural Language Understanding-3s	Natural Language Understanding	Lite	<div></div>
Natural Language Understanding-7r	Natural Language Understanding	Lite	<div></div>
omar-cloudantNoSQLDB	Cloudant NoSQL DB	Lite	<div></div>

Après avoir cliqué sur le service « Conversation », on arrive sur sa page de configuration, comme le montre la figure 41. Une fois sur cette page, on doit cliquer sur « Launch tool ».

Dans les deux cas de figure, on arrive sur cette même page, comme le montre la figure suivante :

Figure 81 : Workspace de service « Conversation »




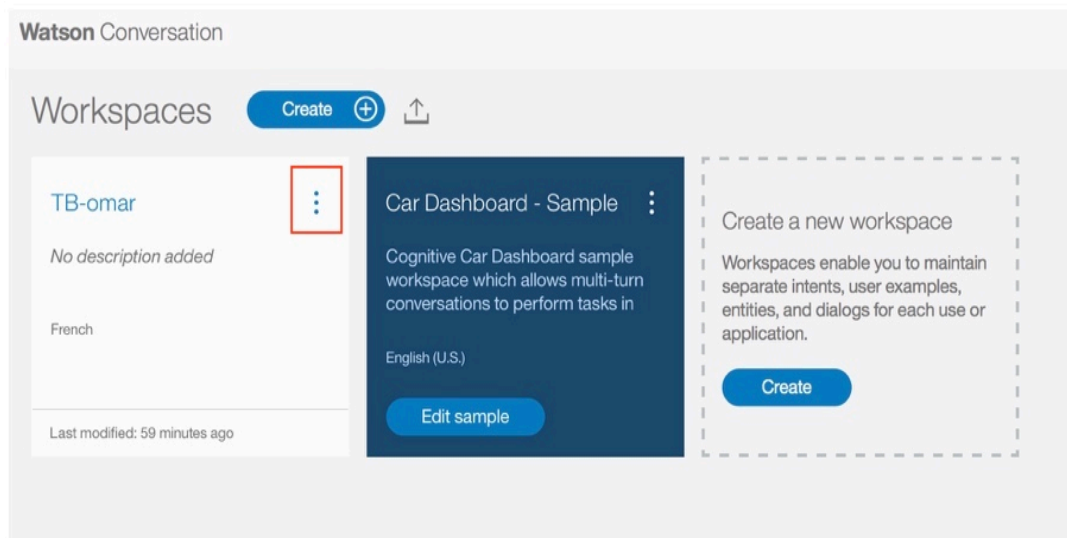
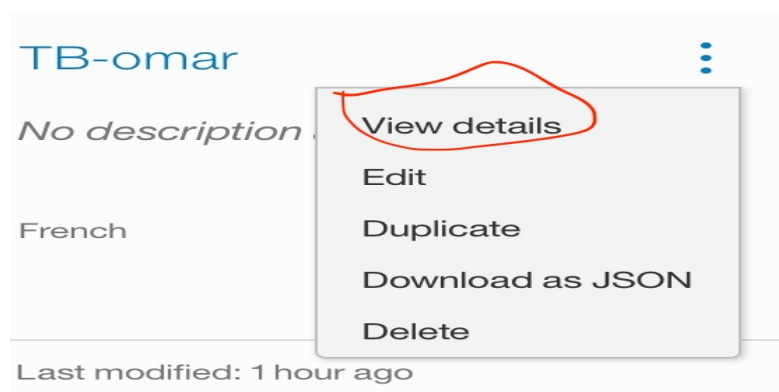
2) Cliquer sur cette icône , comme le montre la figure suivante :

Figure 82 : Clic sur l'icône sélectionnée



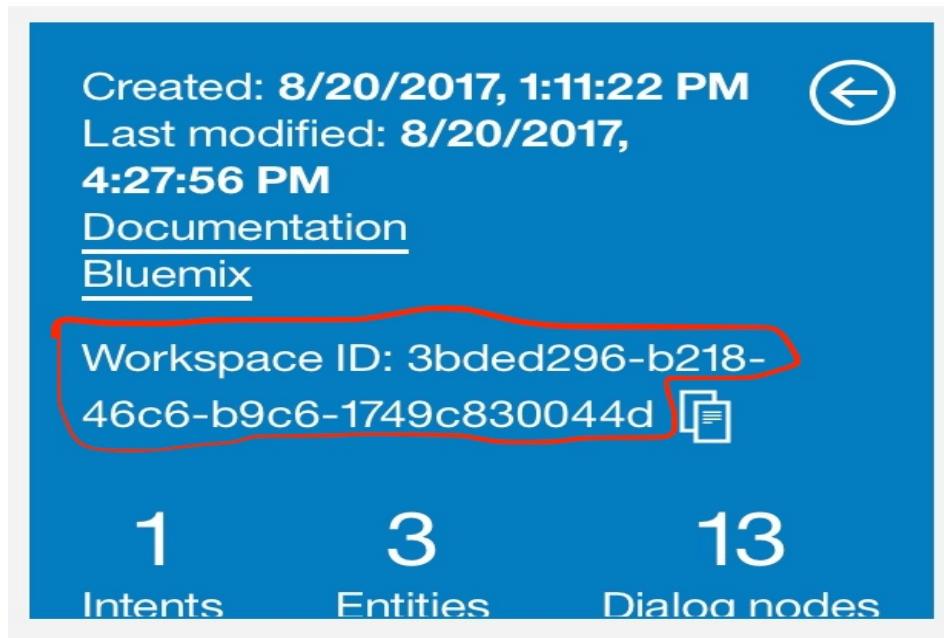
3) **Afficher les détails du Workspace** : Pour cela, on doit cliquer sur « Views details, comme le montre la figure suivante :

Figure 83 : Clic sur « View details »



4) Copier l'id du Workspace : Nous devons maintenant copier l'id du Workspace de notre service « Conversation » car nous l'aurons besoin pour appeler le service depuis notre code Java. Après avoir cliqué sur « View details », on arrive sur les détails du Workspace, dont l'id, comme le montre la figure suivante :

Figure 84 : Détails du Workspace



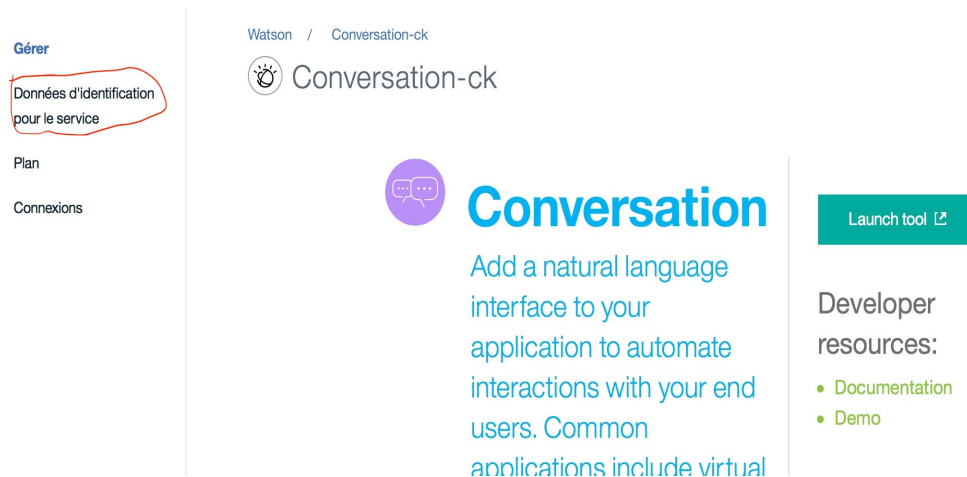
Après l'obtention de l'id du Workspace, il nous faut ensuite les données d'identification (Credentials) pour assurer l'authentification de l'utilisation des services Watson à distance. Autrement dit pour appeler le service « Conversation » à distance, il faut fournir le username et le password (Credentials) et l'id du Workspace.

Pour obtenir les données d'identification de ce service, voici la démarche à suivre :

1) Obtention des Credentials : Pour obtenir les Credentials, on doit revenir sur la page d'accueil de notre compte Bluemix et sélectionner le service « Conversation » en question de la liste de nos services créés, comme le montre la figure 80.

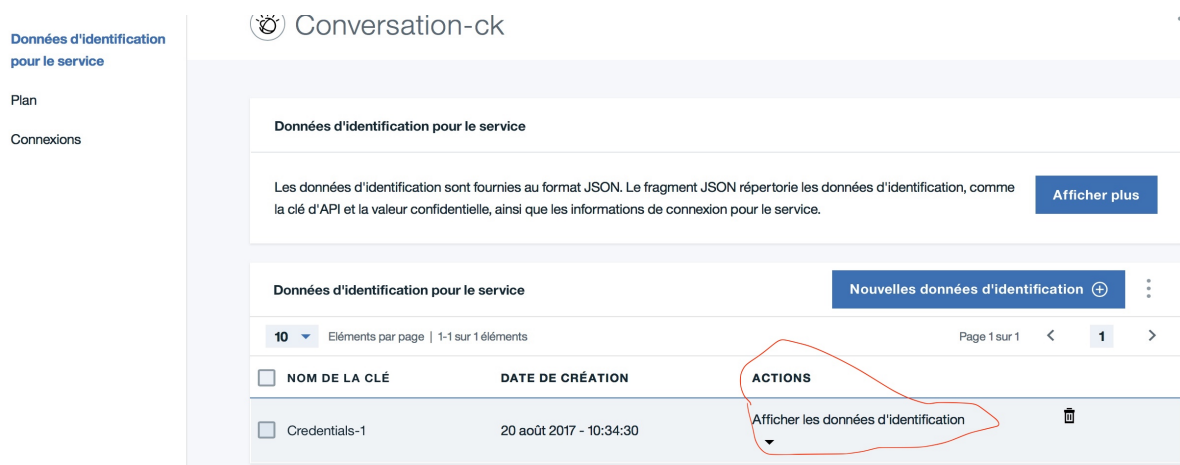
2) Cliquer sur Données d'identification pour le service : Pour ouvrir la page suivante, il faut cliquer sur « Données d'identification pour le service », comme le montre la figure qui suit :

Figure 85 : Informations pour le service « Conversation »



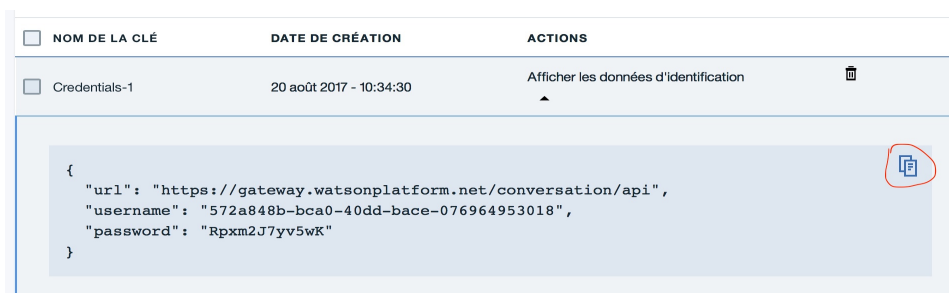
3) Afficher les données d'identification : Pour afficher les Credentials, il faut cliquer sur « Afficher les données d'identification », comme le montre la figure qui suit :

Figure 86 : Données d'identification du service « Conversation »



Une fois qu'on a cliqué sur « Afficher les données d'identification », on obtient les Credentials permettant d'utiliser ce service à distance. Il suffit tout juste de les copier et de les garder soigneusement quelque part, afin de les utiliser dans le code pour appeler le service à distance, comme le montre la figure qui suit :

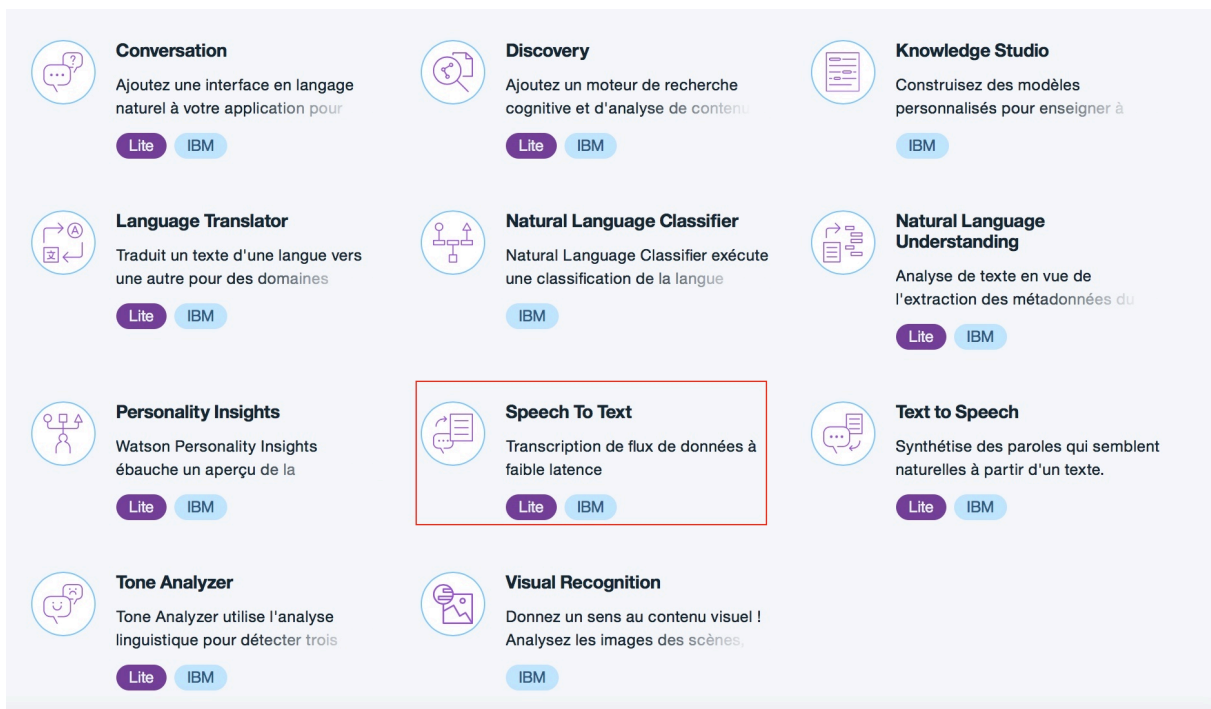
Figure 87 : Les Credentials pour le service « Conversation »



3.2 API Watson « Speech To text »

Pour utiliser l'API Watson « Speech To text », on doit aller sur la liste complète des APIs Watson sur notre compte Bluemix, comme expliqué au point « 2.1 1) et 2) » de la section « **Accès aux APIs Watson** » et cliquer sur le service de notre choix, comme le montre la figure suivante :

Figure 88 : Sélection du service « Speech To text »



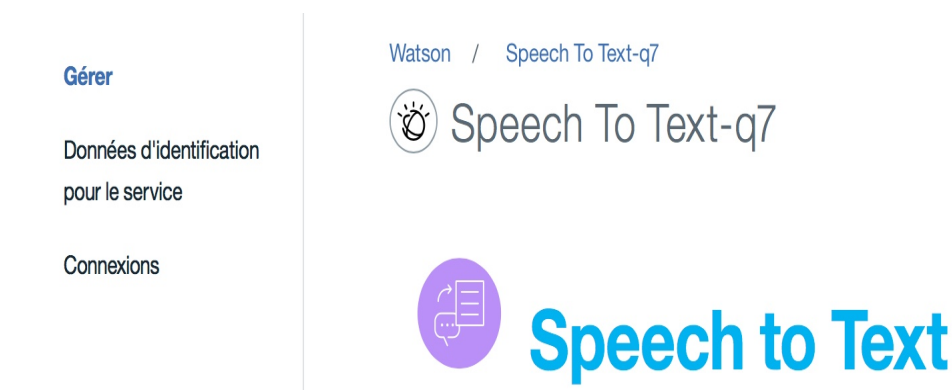
1) Renommer le nom du service ou garder les valeurs par défaut : Ici j'ai gardé les valeurs par défaut, comme le montre la figure qui suit :

Figure 89 : Page de création du service « Speech To text »

The screenshot shows the 'Speech To Text' service creation form. On the left, there is a descriptive text about the service. The main form area contains several fields: 'Nom du service' (Speech To Text-q7), 'Nom des données d'identification' (Credentials-1), 'Sélectionnez une région dans laquelle effectuer le déploiement' (Royaume-Uni), 'Choisissez une organisation' (HEG-omar), 'Choisissez un espace' (dev), and 'Connecter à' (Laisser non lié). At the bottom right is a blue 'Créer' button. There are also links for 'Vous avez besoin d'aide ?' and 'Estimer le coût mensuel'.

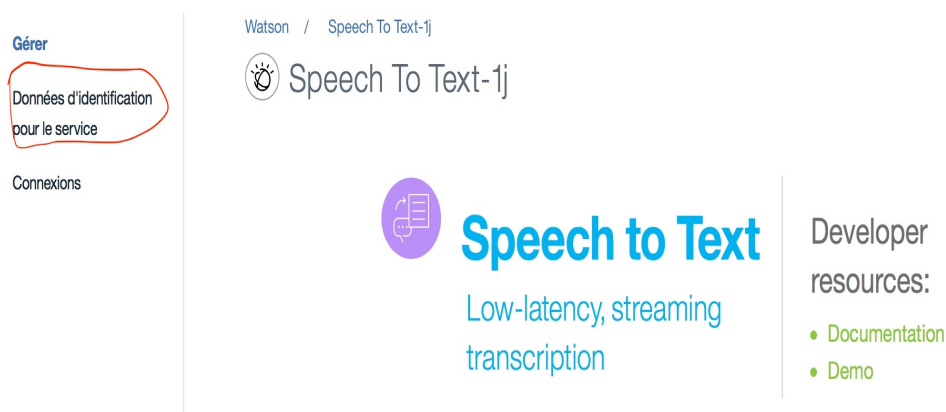
2) Cliquer sur « Créer » pour créer le service : Après avoir cliqué sur ce bouton, on arrive sur la page d'accueil du service, comme on peut le voir sur la figure suivante :

Figure 90 : Page d'accueil du service « Speech To text »



3) **Cliquer sur Données d'identification pour le service** : Pour ouvrir la page suivante, il faut cliquer sur « Données d'identification pour le service », comme le montre la figure qui suit :

Figure 91 : Informations du service « Speech To text »



4) **Afficher les données d'identification** : Pour afficher les Credentials, il faut cliquer sur « Afficher les données d'identification », comme le montre la figure qui suit :

Figure 92 : Données d'identification du service « Speech To text »



Une fois qu'on a cliqué sur « Afficher les données d'identification », on obtient les Credentials permettant d'utiliser ce service à distance. Il suffit tout juste de les copier et de les garder soigneusement quelque part, afin de les utiliser dans le code pour appeler le service à distance, comme le montre la figure qui suit :

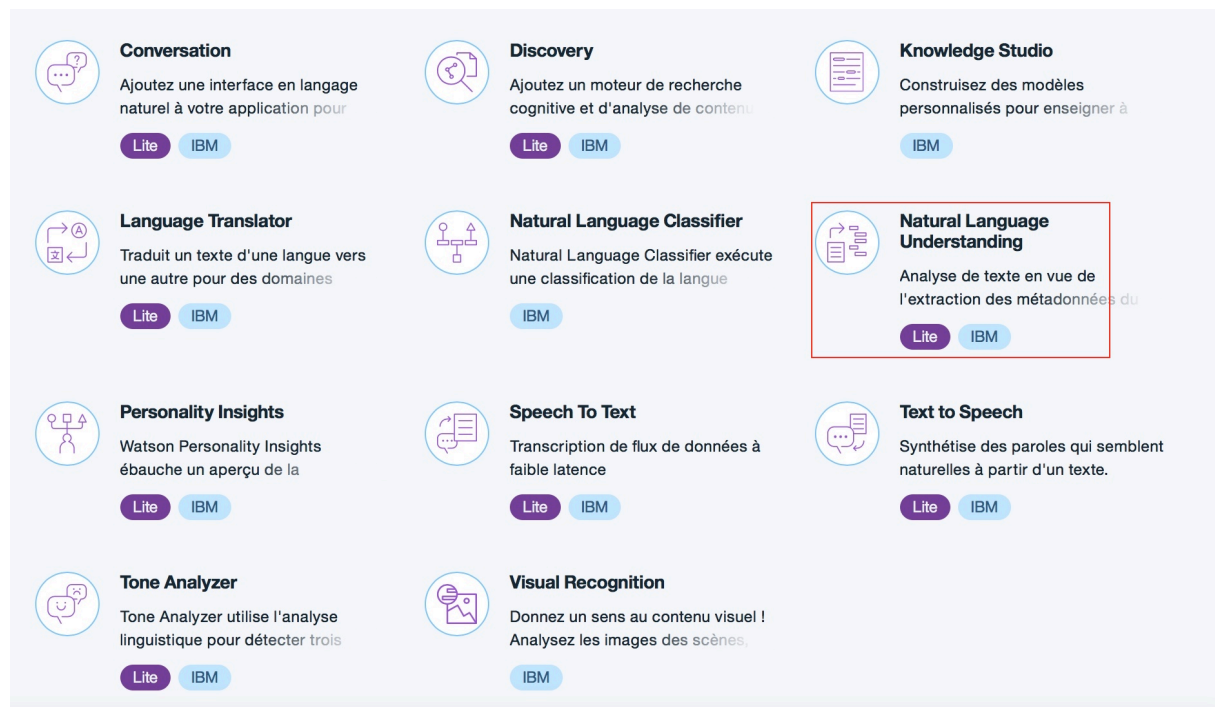
Figure 93 : Les Credentials pour le service « Speech To text »



3.3 API Watson « Natural Language Understanding »

Pour utiliser l'API Watson « Natural Language Understanding », on doit aller sur la liste complète des APIs Watson sur notre compte Bluemix, comme expliqué au point « 2.1 1) et 2) » de la section « Accès aux APIs Watson » et cliquer sur le service de notre choix, comme le montre la figure suivante :

Figure 94 : Sélection du service « NLU »



- 1) **Renommer le nom du service ou garder les valeurs par défaut** : Ici j'ai gardé les valeurs par défaut, comme le montre la figure qui suit :

Figure 95 : Page de création du service « NLU »

Natural Language Understanding

Analyse de texte en vue de l'extraction des métadonnées du contenu, comme les concepts, les entités, les mots clés, les catégories, les sentiments, les émotions, les relations, les rôles sémantiques, à l'aide de la compréhension du langage naturel. À partir de modèles d'annotation personnalisés développés à l'aide de Watson Knowledge Studio, identifiez les relations et entités spécifiques au secteur d'activité/domaine dans du texte non structuré.

Nom du service :
Natural Language Understanding-7r

Nom des données d'identification :
Credentials-1

Sélectionnez une région dans laquelle effectuer le déploiement :
Royaume-Uni

Choisissez une organisation :
HEG-LoOmar

Choisissez un espace :
TB

Connecter à :
Laisser non lié

[Afficher la documentation](#)

Vous avez besoin d'aide ?
[Prenez contact avec le service commercial Bluemix](#)

Estimer le coût mensuel
[Calculateur de coût](#)

Créer

- 2) **Cliquer sur « Créer » pour créer le service** : Après avoir cliqué sur ce bouton, on arrive sur la page d'accueil du service, comme on peut le voir sur la figure suivante :

Figure 96 : Page d'accueil du service « NLU »

Gérer

Données d'identification pour le service

Plan

Connexions

Watson / Natural Language Understanding-7r

Natural Language Understanding

Natural language processing for advanced text analysis

Developer resources:

- Documentation
- Demo

- 3) **Cliquer sur Données d'identification pour le service** : Pour ouvrir la page suivante, il faut cliquer sur « Données d'identification pour le service », comme le montre la figure qui suit :

Figure 97 : Informations du service « NLU »

Gérer

Données d'identification pour le service

Plan

Connexions

Watson / Natural Language Understanding-7r

Natural Language Understanding

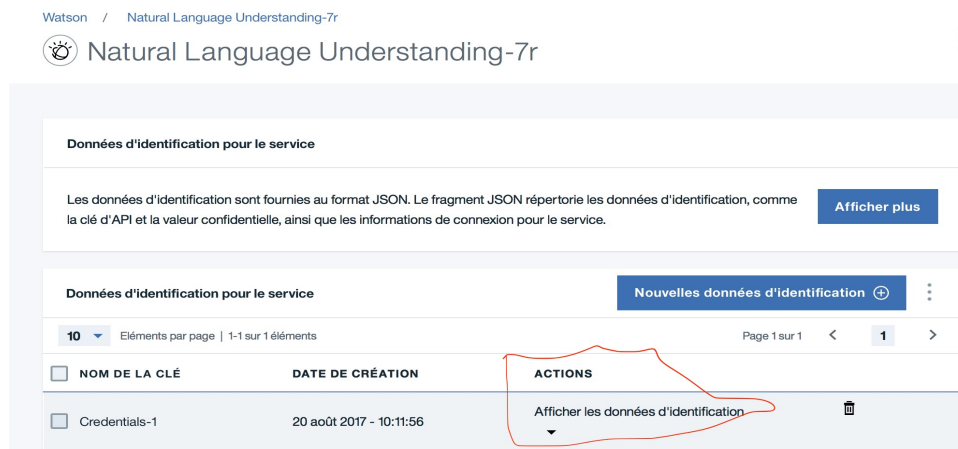
Natural language processing for advanced text analysis

Developer resources:

- Documentation
- Demo

4) Afficher les données d'identification : Pour afficher les Credentials, il faut cliquer sur « Afficher les données d'identification », comme le montre la figure qui suit :

Figure 98 : Données d'identification du service « NLU »



Une fois qu'on a cliqué sur « Afficher les données d'identification », on obtient les Credentials permettant d'utiliser ce service à distance. Il suffit tout juste de les copier et de les garder soigneusement quelque part, afin de les utiliser dans le code pour appeler le service à distance, comme le montre la figure qui suit :

Figure 99 : Les Credentials pour le service « NLU »

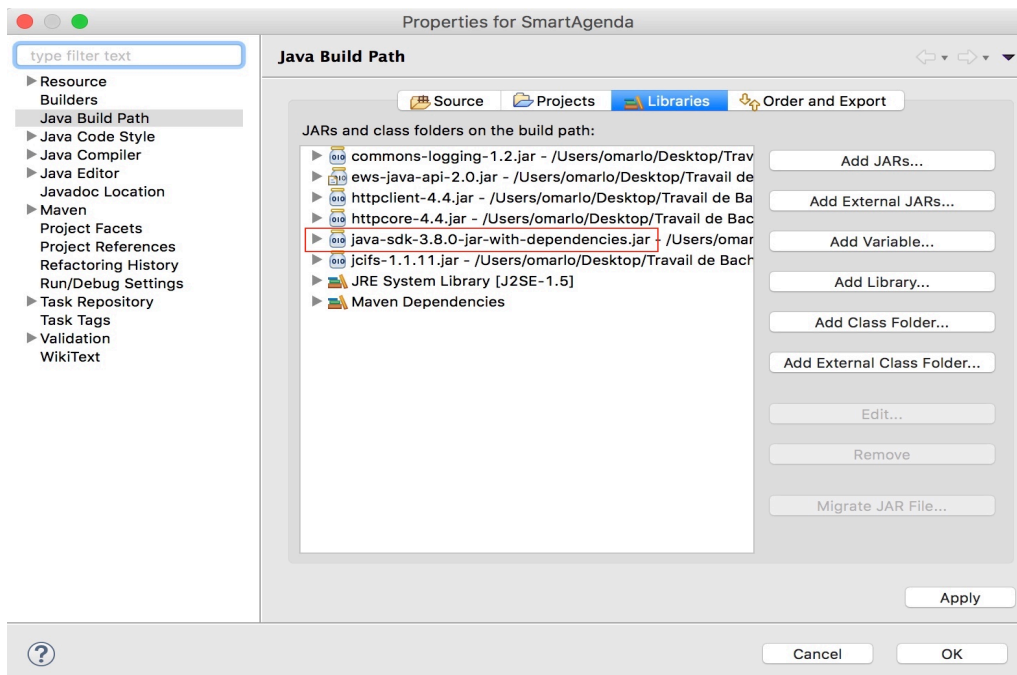


Il reste maintenant à ajouter le SDK des APIs Watson dans les librairies de notre projet Java, pour importer les différents packages nécessaires pour utiliser les différents services de Watson.

4. Configuration du SDK de Watson dans notre projet Java

Pour utiliser les APIs Watson, on doit ajouter le SDK des APIs Watson dans les librairies de notre projet Java. Pour cela, il faut aller dans le « Java Build Path » du projet Java (Maven Project) et ajouter le SDK de Watson, comme le montre la figure suivante :

Figure 100 : Configuration du SDK des APIs Watson



La dernière version de ce jar est disponible à l'adresse :
http://search.maven.org/#search%7Cga%7C1%7Cg%3A%22com.ibm.watson.developer_cloud%22%20a%3A%22java-sdk%22

Il faut maintenant, pour terminer la configuration, ajouter ce bout de code dans le fichier pom.xml du projet Java (Maven Project), au niveau des dépendances :

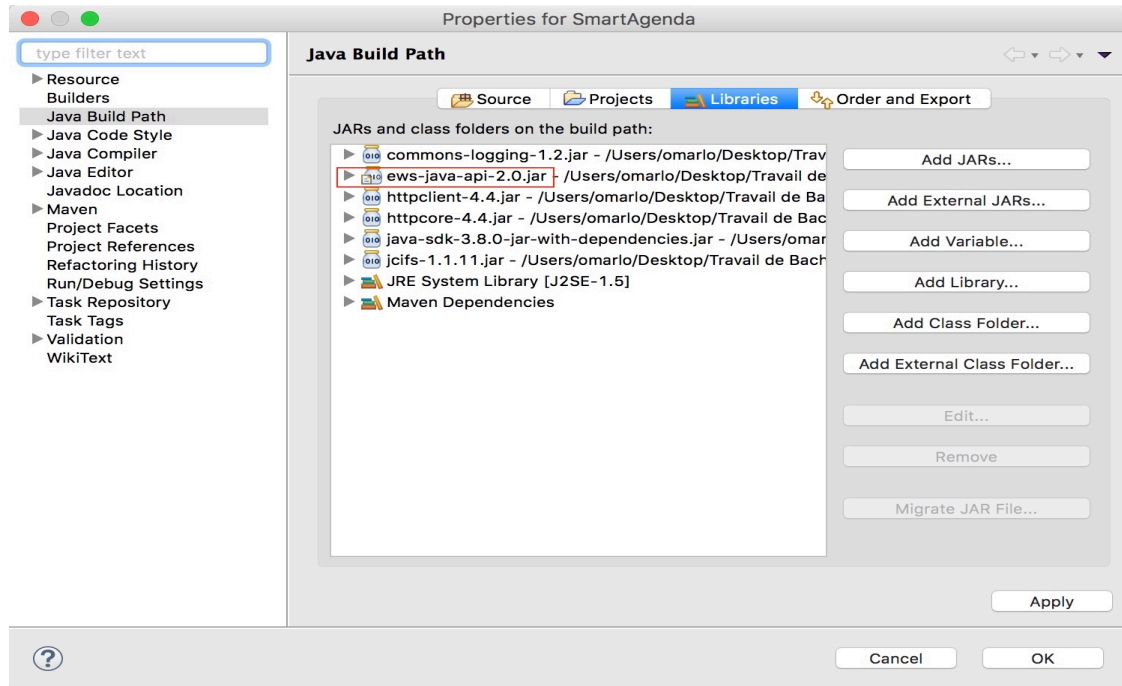
```
<dependency>
  <groupId>com.ibm.watson.developer_cloud</groupId>
  <artifactId>java-sdk</artifactId>
  <version>4.2.1</version>
</dependency>
```

(watson-developer-cloud / java-sdk GitHub)

Annexe 2 : Utilisation de EWS-JAVA-API

Pour manipuler les ressources de Microsoft Exchange Server dans le projet Java à travers EWS-JAVA-API, on doit configurer le « Java Build Path » du projet en ajoutant dans les librairies le SDK de Exchange web services, comme le montre la figure suivante :

Figure 101 : Configuration du SDK de ews-java-api



La dernière version de ce jar est disponible à l'adresse : <http://search.maven.org/#search%7Cga%7C1%7Cg%3Acom.microsoft.ews-java-api>

Il faut maintenant, pour terminer la configuration, ajouter ce bout de code dans le fichier pom.xml du projet Java (Maven Project), au niveau des dépendances :

```
<dependency>
    <groupId>com.microsoft.ews-java-api</groupId>
    <artifactId>ews-java-api</artifactId>
    <version>2.0</version>
</dependency>
```

(OfficeDev / ews-java-api sur GitHub)