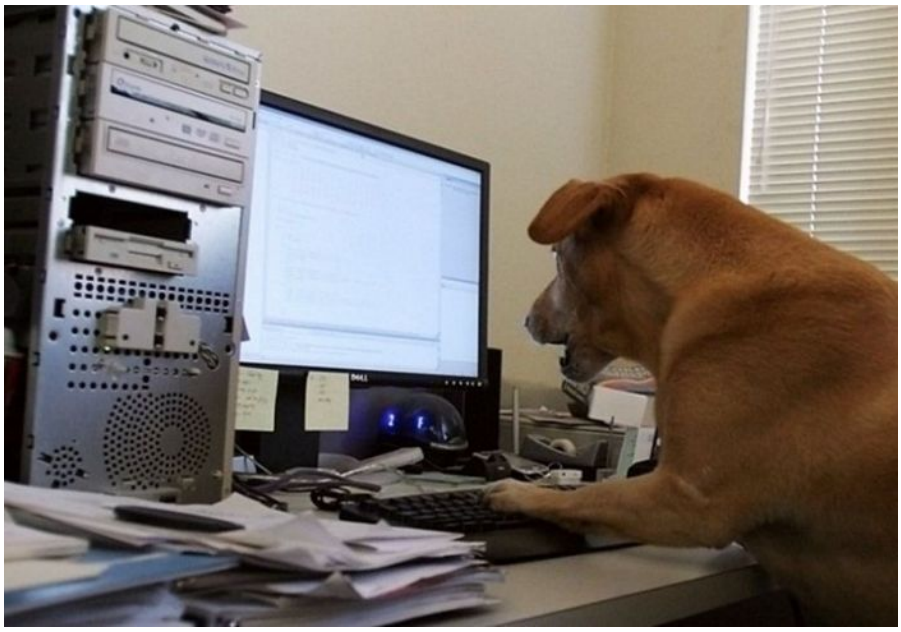
RAPPORT DE STAGE

M. Grégori Jeftic

Développement de l'application Blue Map

Paris – Août/Octobre 2016



“Mais qu’est-ce que je suis en train de faire ?!”

Table des matières

PRESENTATION GENERALE

Introduction

Ce rapport est l'achèvement de ce qui marque une grande étape dans ma vie professionnelle et personnelle.
Professionnelle, car il représente l'aboutissement final d'un long cheminement après ma décision de me reconvertir professionnellement. Sa qualité et son contenu représenteront, je l'espère, la motivation et la soif d'apprendre qui m'animent depuis que j'ai décidé vouloir devenir concepteur-développeur informatique.
Il est ainsi la synthèse de ma toute première expérience dans une entreprise en tant que tel, et marque le point d'entrée de mon tout nouveau parcours professionnel.

Et enfin personnel, dans le sens où non seulement ce stage mais aussi toute la période de formation ont été un pour moi un grand tournant dans ma vie, je me suis découvert des capacités de concentration, de rigueur, de persévérance et de résistance à l'envie de céder à mes pires émotions face à la difficulté, la fatigue et l'impression d'immensité qui s'ouvrait de plus en plus à mesure que j'apprenais de nouvelles choses...
Ces qualités que j'ai totalement découvertes car mon expérience de l'éducation nationale avait été désastreuse, et c'est dans une immense crainte, presque même à reculons que je me suis inscrit à la formation de concepteur-développeur !

Depuis ces quelques mois, je rêve régulièrement de classes, d'objets, de bases de données et de serveurs, mon vocabulaire avec mes amis est de plus en plus orienté informatique et dès mon arrivée chez moi je me plonge dans nombre de tutoriels sur les applications « MEAN » (MongoDB, Express, Angularjs, Node.js), Météor.js, React.js, Neo4j, la création d'API REST, et tout ce que je trouve de nouveau, et vais régulièrement à des meetups...

Pour résumer ce que j'ai craint et pour lequel j'ai été jusqu'à la souffrance, je ne peux plus m'en passer désormais car je suis passionné !

C'est au sortir de cette riche expérience que je tiens à partager ces quelques citations qui ont pris tout leur sens au cours de ces derniers mois :

« Le travail de la pensée ressemble au forage d'un puits, l'eau est trouble d'abord, puis elle se clarifie. » Proverbe chinois.

« Mais nan mais la constante de Planck et la gravité, donc, la flèche du temps, euh..., et donc le chaos qui crée le vide, et donc dieu ! » "Proverbe" de Fabrice, développeur talentueux, tous les soir au pub, complètement bourré.

« C'est toujours l'impatience de gagner qui fait perdre » Louis XIV cité dans «L'immortel» de FOG.

« Tout est possible à celui qui croit » La Bible.

Mon profil et mon parcours

L'informatique a toujours été quelque chose de fascinant pour moi depuis mon plus jeune âge mais les manques de moyens de ma famille m'ont longtemp maintenu éloigné des ordinateurs. J'ai finalement acquis un micro-ordinateur Atari XEGS et peu après le logiciel permettant d'accéder au Basic Atari et j'ai immédiatement été plongé dedans le plus clair de mon temps.

Mais suite à des problèmes avec l'éducation nationale, j'ai dû m'écarter du cycle général et ainsi voir mes rêves de faire de ma passion un futur, peu après le début de mes études du second cycle, les problèmes financiers de ma famille s'étant aggravés j'ai commencé à travailler.

J'ai été employé comme commercial dans les débuts de la société GrosBill Micro, et j'ai parcouru l'ensemble des métiers du monde de la sécurité pour faire 9 ans comme agent de protection rapprochée. Après une rupture avec mon ancien métier j'ai commencé à chercher les formations possibles dans le monde de l'informatique mais les portes se sont toutes refermées devant mon parcours académique et j'ai donc commencé une longue période à me rechercher.

C'est grâce à ma soeur qui a plus cru en moi que je ne l'aurais pu à ce moment là, que j'ai découvert M2I et le titre de concepteur développeur informatique qui m'était accessible dans cette école avec mon niveau scolaire!

Il ne me restait plus qu'à convaincre le Pôle Emploi de me financer cette formation, pour y arriver il me fallut trouver une entreprise leur prouvant que ce titre était intéressant pour le recrutement et c'est M. Vincent Agami qui me fit une lettre qui débloqua le processus dans lequel je suis encore engagé et qui a bouleversé ma vie et mes ambitions.

Remerciements

Je tiens tout d'abord à remercier le Pôle Emploi de Courbevoie qui a cru en mon projet de reconversion et leur réponse positive a été la première réussite dans l'accomplissement de celle-ci. Rien de tout cela n'aurait été possible sans leur financement.

Je remercie également l'entreprise F4 chez qui j'ai pu réaliser mon projet de stage, ils ont été très flexibles afin de me permettre d'adapter mon projet le plus possible aux nécessités de ce qui m'a été demandé. Ils m'ont permis de me confronter à la réalité du métier qui veut que l'on sache s'adapter car les technologies sont en perpétuelle évolutions et des techniques acquises un jour peuvent être dépréciées le lendemain par de nouvelles techniques plus adaptées, en me faisant travailler sur des technologies que l'on avait pas vues ou à peine survolées en cours !

Toute l'équipe de développeurs qui m'ont permi immédiatement de m'intégrer en me donner de bons conseils et qui on su alléger mes tensions avec de bon moments de rigolade !
Et particulièrement Vincent Agami.

Il va sans dire que ma gratitude s'adresse aussi à toute l'équipe M2I qui m'ont accompagné durant les 6 mois de formation, je pense particulièrement à Pascal Buguet mon formateur référent qui à réussi à me démarrer d'un coup de manivelle alors que je ne l'aurais pas cru moi-même possible, avec le rythme effréné de ses cours il m'a permi de me concentrer sur l'instant présent et me faire oublier la très longue ligne droite qui s'étendait à perte de vue devant moi ! Un incroyable Monsieur… Je pense aussi à Sébastien Maloron, un formateur particulièrement attentif au besoins de chacuns aussi variés soient-ils, il a su écouter mes problèmes et me tendre la main lorsque je me sentais me noyer, sa gentillesse m'a alimenté en kérosène.

Tous mes amis et ma famille qui m'ont encouragés et m'aider financièrement quand c'était limite, mon beau-frère Joel Ribis chercheur au CEA qui en relisant mon rapport m'a assuré être sur le bon chemin, et lorsque je n'en pouvais plus sous la charge de travail il su me remonter le moral en disant « c'est bon signe ça ! C'est normal et ça prouve que ce tu fais, va dans le bon sens. T'inquiète pas j'en ai fait un paquet des mémoires !… » !

Il y a mes compositeurs préférés et leurs « musiques de dégénérés » à en croire certains, grâce auxquels j'ai réussi à m'isoler, me concentrer et garder mon rythme cardiaque à 10 000 BPM !

Je vous remercie vous, examinateurs, pour le temps que vous donnez à la lecture de ce rapport, j'espère qu'il sera agréable à relire malgré tout les défauts qui m'auraient échappé, et pour la chance que vous représentez de pouvoir accéder à mon rêve.

Un grand merci à Pascal Roques qui a accepté de jeter un oeil à mon rapport !
Pour finir, rien de tout ça n'aurait été possible sans ma soeur Sanja, qui a bien plus cru en moi que je n'aurais su le faire, en m'inscrivant « à l'insu de mon plein gré ! » à un rendez-vous chez M2I…

La démarche générale

Lors de mes premiers entretiens avec Mathieu Bineau, Directeur de la société Voltalis et mon tuteur de stage Vincent Agami concernant le projet Blue Map et vu qu'il s'agissait de ma toute première expérience de réalisation d'une application de manière totalement autonome, plusieurs points ont été sources de questions.

Je me suis très vite rendu compte à quel point les discussions entre un client et un développeur peuvent être confuses et sources de malentendus même après un long échange. J'ai alors rapidement compris l'importance de la qualité de démarche d'analyse et de conception que j'allais devoir produire afin de communiquer très précisément et amener mon projet à terme avec le moins de retouches possible.

Je me suis donc inspiré de la démarche de modélisation d'une application web basée sur UML et développé dans l'ouvrage de Pascal Roques « UML2 – Modéliser une application Web (Editions Eyrolles) ».

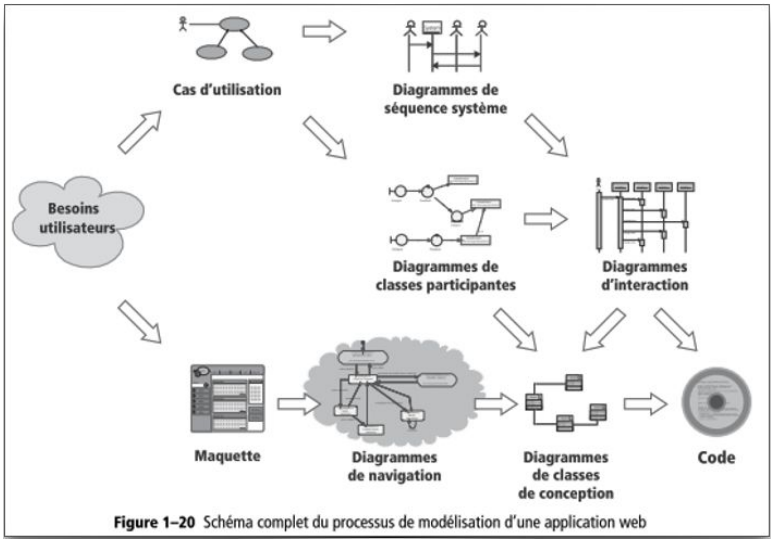


Figure 1–20 Schéma complet du processus de modélisation d'une application web

Il me fallait tout d'abord bien comprendre et décrire de façon précise les besoins de mon client et des utilisateurs, une fois validé, construire la solution. Ce fut après ces parties «analyse des besoins» et «analyse des solutions», que les tâches à accomplir devinrent beaucoup plus claires, et l'utilité de cette démarche UML si pertinente.

Tout cela fut complété par la phase de conception qui, à l'aide de certains diagrammes comme les diagrammes de séquence détaillées, m'a permis d'identifier les interactions de l'acteur avec les différentes couches systèmes, le tout me donnant un champ de vision clair et détaillé pour attaquer la partie «code». En clair, cette démarche m'a permis de résoudre la problématique «comment passer des besoins utilisateurs au code de l'application ?».

Cette démarche fut aussi enrichie par le cours de Mr. Pascal Buguet et adaptée par rapport au projet sur lequel j'ai travaillé, lequel étant entièrement en Javascript pur sous environnement Node.js m'a posé énormément de problèmes car d'abord cette technologie était nouvelle pour moi puisque non étudiée en cours, et qu'il n'y a pas de classes dans ce langage même si tout est objet!

Abstract

La société F4

Lorsque j'ai contacté M. Vincent Agami, un Directeur Technique de la société F4, il m'a fait part de leur volonté de fournir à une société soeur Voltalis une application utilisable à partir d'un prototype sommaire. Le prototype est basé sur le moteur de carte 3D interne de la société F4, F4 Map.

« La société F4 réalise en ce moment une application autour d'une API de localisation sur une carte 3D qu'il propose aux agences immobilière comme élément central pour un site concurrent à www.seloger.com, sur lequel elles peuvent envoyer leurs annonces. Les clients ont donc un aperçu des logements disponibles sur une carte 3D, et pour une partie des annonces une visite virtuelle du logement. Cette application, c'est Bien'ici : https://www.bienici.com/ »

The F4 company

When I met M. Agami Vincent, a Technical Director of the company F4, he shared with me their will to provide their sister company Voltalis a functional application from a prototype they have. This prototype is based upon the internal 3D map engine F4 produced, it's called F4 Map.

« The F4 company is working on a 3D localisation Map API which they make available as the core to a commercial website to registered real estate agencies in the form of a concurrent to sites like www.seloger.com, on which they may post their property advertisements. Customers get an insight of available housing on a 3D map, and for some of these have a virtual visit of the houses. This application, is Bien'ici : https://www.bienici.com/ »

La société Voltalis & le projet Blue Map

Le business

La société Voltalis opère en France un réseau dit « smart-grid », qui permet d'interagir directement à distance sur la consommation électrique chez les consommateurs (particuliers, industries, centres commerciaux) à la demande des producteurs d'énergie afin de lutter contre les pics de consommations qui nécessitent la mise en route de centrales thermiques très consommatrices en énergies fossiles et autrement plus polluantes, les centrales nucléaires n'ayant pas la souplesse nécessaire.

Les partenaires

Voltalis travaille avec des sociétés comme le gestionnaire du réseau de distribution RTE pour la France dont le rôle est d'équilibrer production et consommation. Un déséquilibre peut au pire faire tomber le réseau (est arrivé en Allemagne le 4 novembre 2006) ou entraîner le démarrage de centrales thermiques, car les centrales nucléaires répondent lentement.

La technique

Voltalis permet l'équilibrage du réseau électrique quasi temps réel en écrêtant les pics de consommation par une technique nommée **effacement diffus**[1]*, exemple: si on coupe pendant 5 minutes 100 000 radiateurs de 1000 W, cela fait 100 MW pendant 5 minutes, ou 25 000 (4x moins donc) pendant 20 minutes.

L'infrastructure

Pour cela Voltalis a installé des appareils (Bluepods) de mesure et coupure dans les habitations (installation gratuite) et chez les industriels.

Ces Bluepods sont des pilotes reliés à un ou plusieurs modulateurs par compteurs électrique et ne coupent que les appareils ayant un effet négligeable sur le confort: radiateurs, ballons d'eau chaude, partie réfrigération des climatiseurs industriels (les rooftops) etc.

Blue Map préfigure un outil donné à GRD/RTE, pour le **pilotage de l'effacement diffus**, à partir de données de consommation géolocalisée sur une carte.

* voir annexes définitions.

[1]L'**« effacement diffus »** ou *« lissage de la courbe de charge par le pilotage de la demande »* consiste, en cas de déséquilibre offre/demande d'électricité à provisoirement réduire la consommation physique d'un site donné ou d'un groupe d'acteurs, l'effacement étant déclenché par une stimulation extérieure.

CAHIER DES CHARGES

L’application Blue Map : objectifs généraux

Le prototype Blue Map permettait d’avoir un aperçu visuel et « sexy » d’une préfiguration d'un outil donné à des gestionnaires de réseau (GRD) comme RTE pour le pilotage de l'effacement. Il est donc dans l’objectif une vitrine de leur savoir-faires à l’attention de leurs clients, comme le distributeur d’énergie RTE, pour qui la gestion en temps réel d’une smart grid est primordial.

Le président de Voltalis M. Mathieu Bineau m’a donc exprimé les points sur lesquels il souhaitait voir évoluer le prototype conçu par la société F4. Ceux qui rendraient l’utilisation de Blue Map confortable et surtout utilisable lors de présentations devant ses clients.

Objectifs immédiats

Les objectifs définis lors de cette réunion ont été :

- On souhaite pouvoir accéder à l’application Blue Map et manipuler la carte depuis le portable de M. Bineau ou d’un PC
- L’utilisateur peut piloter la carte grâce à un moteur de recherche de localités.
- Un système de support technique temps-réel utilisable par le gestionnaire de réseau afin de pouvoir communiquer avec l’équipe Voltalis.
- Voir des statistiques de consommation en sélectionnant directement sur la carte
    - un Bluepod
    - un ensemble de Bluepods en traçant un polygone, de la taille que l’on souhaite, jusqu’à la France entière.
- Il faut pouvoir sauvegarder des ensembles de Bluepods pour y revenir ensuite lors de démonstrations ultérieures: la carte se centrera automatiquement sur l’ensemble sauvegardé.
- Il faut pouvoir supprimer les ensembles sauvegardés.
- Pour rajouter des objets sur la carte 3D, comme des éoliennes, et leur associer des Bluepods et données de production associées, à titre de démonstration, pour figurer l’énergie produite.
- Un système de chat temps réel figurant un support technique immédiat apporté par Voltalis
- La vue peut être switchée entre 2D et 3D
- Dans un deuxième temps: fonctionnement partiel au moins de l’application en mode hors connexion : en clientèle, il n’y a parfois pour des raisons de sécurité aucun accès à Internet, pas même en 4G. En Chine par exemple, les firewalls de gouvernement bloquent ou ralentissent considérablement la connexion.

Exemple de présentation pour des clients

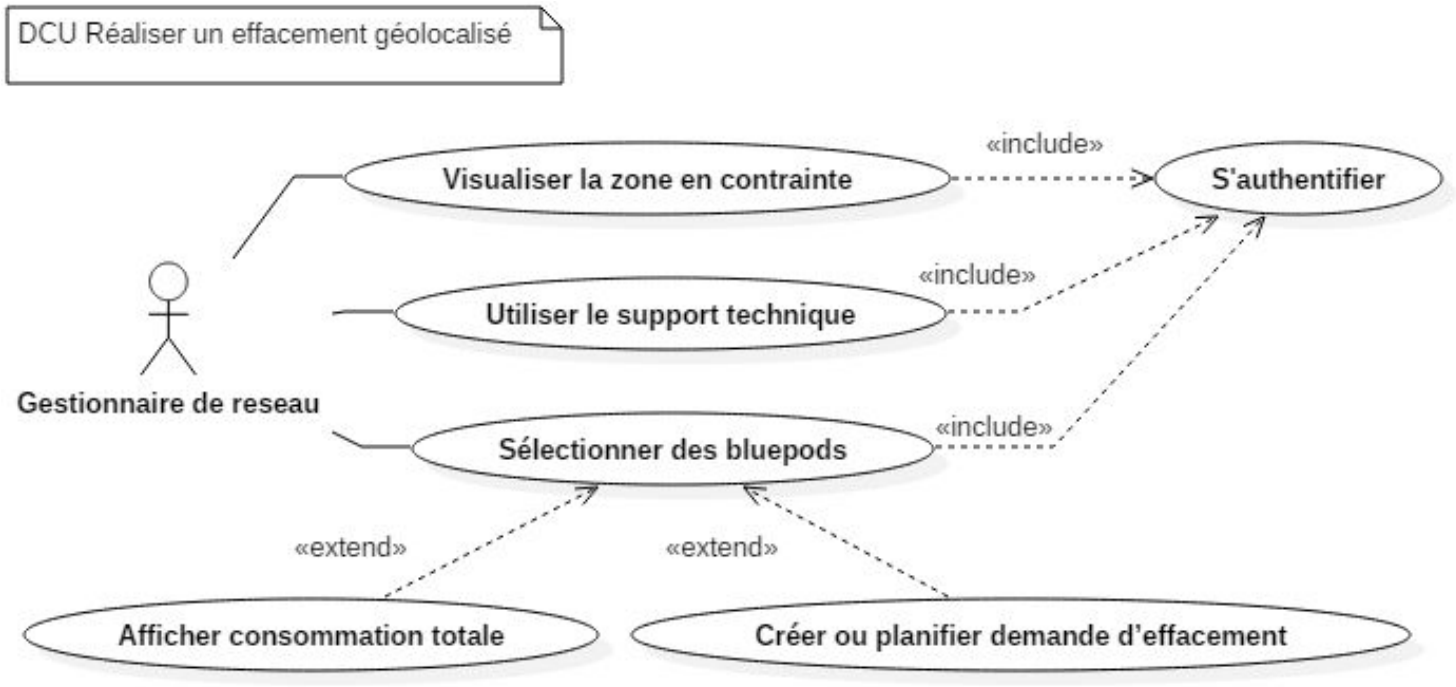
Scénario typique de démonstration

Scénario typique de démonstration d'effacement diffus que M. Bineau voudra pouvoir réaliser :

1. M. Bineau représentant un GRD, se connecte à l'application Blue Map
2. Pouvoir communiquer avec le support technique temps réel
3. Il visualiser la zone en contrainte
4. Il sélectionne des Bluepods
    a. Affiche consommation totale
    b. Il clique sur « Produire maintenant » ou « Produire plus tard »
        i. Blue Map le redirige automatiquement vers l'application ePowerPlant
            1. Il crée ou planifie une demande d'effacement.

Je vais le présenter sous la forme d'un diagramme de cas d'utilisation pour une meilleure vue générale :

Diagramme représentant quelques cas d'utilisation utiles lors d'une démonstration



Il est à noter que **par soucis de lisibilité**, j'ai réunis les cas « Produire maintenant » et « produire plus tard » en un. Ce diagramme est à titre d'exemple pour démontrer le cas de l'effacement diffus et je n'entrerai pas dans les détails car je ne maîtrise pas le développement de l'application « ePowerPlant ».

ANALYSE

Le prototype Blue Map, état des lieux

J'ai commencé par procéder à un état des lieux.

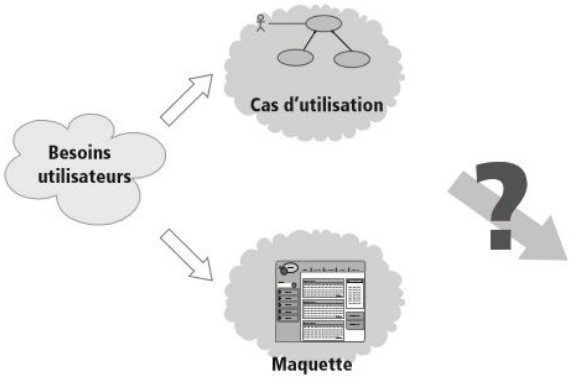
En gras, les problèmes prioritaires par rapport aux objectifs :

- ➔ **Pas de possibilité de recherche: on se déplace à la souris sur la carte pour aller où on veut, dans la France entière, ce qui est très laborieux.**
- ➔ **Besoin d'intégrer un support technique en temps réel entre le client et un ingénieur Voltalis.**
- ➔ Possibilité de sélectionner un icône BluePod et afficher un panneau de statistiques: OK, suffisant.
- ➔ Possibilité de sélectionner un polygone qui affichera l'ensemble d'icônes contenu et affichera un panneau de statistiques sur l'ensemble.
  - ◆ **Fonctionne uniquement pour une zone limitée à un milliers de Bluepods. Après ça devient très lent voire plante. Or la Bretagne comporte plus d'un tiers des 80 000 Bluepods installés en France.**
- ➔ Gestion des ensembles de sites
  - ◆ Lorsqu'un ensemble d'icônes est sélectionné un bouton « saveGroup » apparaît et sauvegarde cet ensemble en faisant un .push dans l'array « savedGroups ».
  - ◆ Un hamburger dans la liste des boutons permet d'afficher un lien pour chaque enregistrements de polygone dans « savedGroups » qui permettent de déplacer la carte sur celui-ci ou supprimer cet enregistrement via une icône à côté du lien.
- ➔ L'application prend 7 secondes à démarrer. Mais M. Bineau peut faire préparer la démonstration à l'avance.
  - ◆ Le problème de lenteur vient de la manière d'envoyer les Bluepods à la carte graphique, qui les reçoit une par une mais si on les envoyait en un bloc l'application serait totalement bloquée le temps que la carte graphique (GPU) les traite toutes! Il est donc difficile d'accélérer le démarrage de l'application.

Les diagrammes de cas d'utilisation généraux

Présentation des diagrammes de cas d'utilisation et représentation graphique

Suite à l'expression des besoins détaillés dans le cahier des charges, nous pouvons désormais spécifier ces exigences à travers une **modélisation graphique de ces besoins**. En nous inspirant partiellement de la démarche de Pascal Roques, nous allons d'abord nous attarder sur les cas d'utilisations.



Un **diagramme de cas d'utilisation (DCU)** est un ensemble de **cas d'utilisation (CU)**. Il formalise graphiquement les besoins des utilisateurs.
Un **Cas d'Utilisation** permet de formaliser une tâche fondamentale que l'utilisateur attend du système, il correspond à un de ses besoins. Il correspond à un service rendu par le système. Il permet de décrire ce que le futur système devra faire, sans spécifier comment il le fera.

Il existe trois concepts clés : l'**acteur**, le **cas d'utilisation** et l'**interaction** entre les deux. Les étapes qui nous permettront d'aboutir au modèle des cas d'utilisation sont les suivantes :

- identifier les **acteurs**,
- identifier les **cas d'utilisation**,
- structurer les cas d'utilisation en **packages**,
- ajouter les **relations** entre cas d'utilisation,
- finaliser un ou plusieurs **diagramme(s) de cas d'utilisation**.

Identification des acteurs

Il y a deux acteurs :

➔ Les Gestionnaires de Réseaux de Distribution (nommé GRD, ou gestionnaire de réseau)
➔ Les ingénieurs support de Voltalis

Identification des cas d'utilisation (avec plus de profondeur)

Je me suis limité à certaines des demandes de Mathieu Bineau.

Pour le client, gestionnaire de réseau de distribution (GRD)

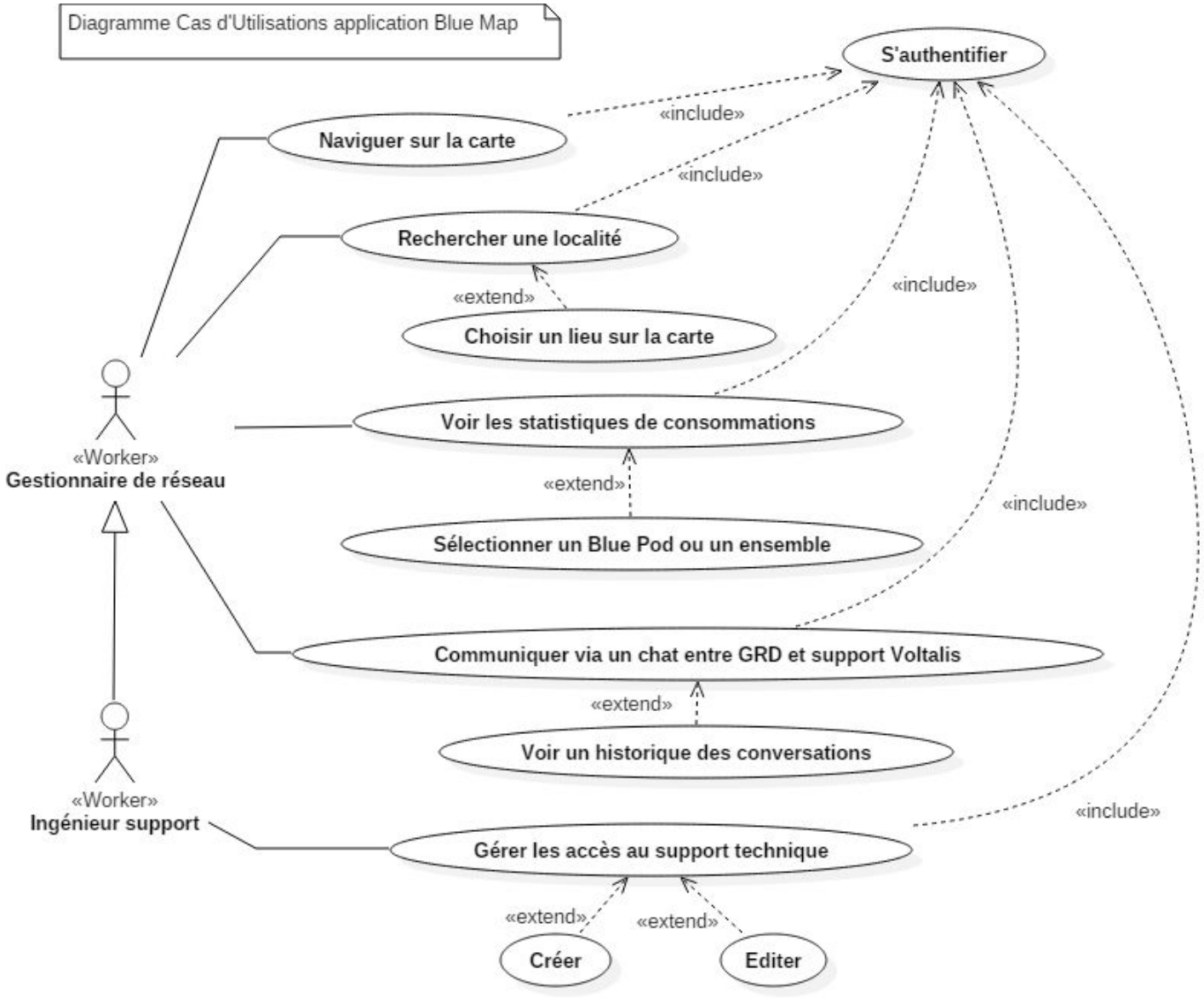
- ➔ Naviguer sur la map
  - ◆ Se géolocaliser et centrer la carte sur la position trouvée
  - ◆ Faire faire différent mouvements d'orientation à la caméra
    - Récupérer la mise à jour de l'affichage de la carte
  - ◆ Faire des rotation de la caméra, la ré-aligner vers le nord
  - ◆ Switcher entre un affichage 2D et 3D de la map
- ➔ Voir les statistiques de consommations
  - ◆ Les statistiques de consommation d'un Bluepods seul
  - ◆ D'un ensemble de Bluepods
    - Dessiner un polygone pour sélectionner un ensemble de Bluepods
    - Sauvegarder un polygone de Bluepods et lui donner un nom
    - Supprimer un ensemble sauvegardé
    - Rappeler un ensemble sauvegardé
- ➔ Rechercher une localité
  - ◆ Saisir un nom et obtenir des résultats de localités
    - La carte est automatiquement pilotée vers le premier résultat
  - ◆ Choisir une localité et...
    - voir la carte se déplacer à cet endroit
- ➔ Communiquer avec l'équipe de support Voltalis
  - ◆ Le GRD peut demander de l'aide

Pour l'ingénieur support Voltalis

- ➔ Répondre aux demandes d'aide des GRD
  - ◆ Le premier ingénieur disponible et compétent répond
    - Discussion sous forme d'un chat
    - L'ingénieur peut rediriger la discussion vers un autre ingénieur
    - Le gestionnaire peut décider de marquer la discussion comme « résolue »
- ➔ Gérer les accès au support technique
  - ◆ Gérer les personnes :
    - Gérer les client
    - Gérer les membre de l'équipe de support technique
  - ◆ Gérer les sociétés

Diagramme des Cas d'Utilisation de l'application Blue Map

Après avoir analysé les différents cas d'utilisation, nous allons pouvoir représenter graphiquement le diagramme des cas d'utilisations qui incorporera aussi les **relations** entre les cas d'utilisations, et correspondra au **processus métier**.



Afin de vous faciliter la lecture, j'ai composé « sélectionner un Bluepod » et « sélectionner un ensemble de Blue Pods » en « sélectionner un Bluepod ou un ensemble », j'ai également placé les stéréotypes « include » à droite de leurs association et les stéréotypes « extend » à gauche de leurs association.

Fonctionnalité de support technique

Cas d'utilisation « Communiquer avec le support Voltalis via support technique »

Scénario nominal

- Sur le site Blue Map, il y a toujours un bouton « Demander de l'aide » (un « ? »)
- Le GRD clique sur ce bouton, une fenêtre apparaît contenant
    - Une liste de l'historique de ses tickets
    - Une zone de saisie et un historique de conversation pour un ticket dans une liste
    - Une case à cocher permettant de classe la conversation comme « résolue »
- Il tape du texte pour poser sa question
- Il appuie sur entrée pour l'envoyer
    - Le message est stocké dans son historique
    - Il est envoyé à tous les ingénieurs support de Voltalis connectés.
- Le message arrive dans une liste de « discussions en attente » chez les ingénieurs support,
- Un ingénieur du support Voltalis ouvre la conversation
- Pour les autres ingénieurs connectés, elle sort de la liste d'attente.
- Si deux ingénieurs on cliqué presque en même temps, le 2ème reçoit le message : « conversation déjà prise par untel » .
- Si le GRD considère la réponse comme satisfaisante, il classe la conversation comme « résolue »
    - Le système lui demande confirmation
    - S'il confirme, la conversation est archivée.

Scénario alternatif : l'ingénieur support a besoin d'un autre ingénieur support pour apporter une réponse

Le GRD pose une question, un ingénieur connecté prends la question, s'ensuit des échanges.

- A un moment donné l'ingénieur remet la conversation en attente prioritaire :
    - Bouton transferer
        - Actuellement, la question est remise en attente mais prioritaire.
- Le GRD est informé et mis en attente
- Un autre ingénieur prend la conversation
- Le GRD est averti qu'un ingénieur support s'est connecté à sa demande :
    - L'ingénieur obtient l'historique complet
- L'ingénieur cible reçoit une conversation prioritaire dans la liste des conversations en attente
    - Il l'ouvre et le GRD est averti de la présence du nouvel ingénieur
    - ...Lequel lui répond à nouveau.

Scénario d'erreur : déconnexion ou plantage du gestionnaire de réseau

Un GRD est en train d'échanger avec un ingénieur. Déconnexion, par exemple à cause d'une coupure de réseau.

- Le GRD reçoit un message du système l'avertissant qu'il est déconnecté.
- L'ingénieur reçoit un message du système l'avertissant que le GRD est déconnecté.
    - L'ingénieur peut finir sa réponse, et l'envoyer.
    - La réponse de l'ingénieur est enregistrée dans la base de données.
- Lorsque le GRD se reconnecte, les derniers messages lui sont envoyés depuis la base.
- L'échange peut reprendre.

Scénario d'erreur : déconnexion ou plantage de l'ingénieur support

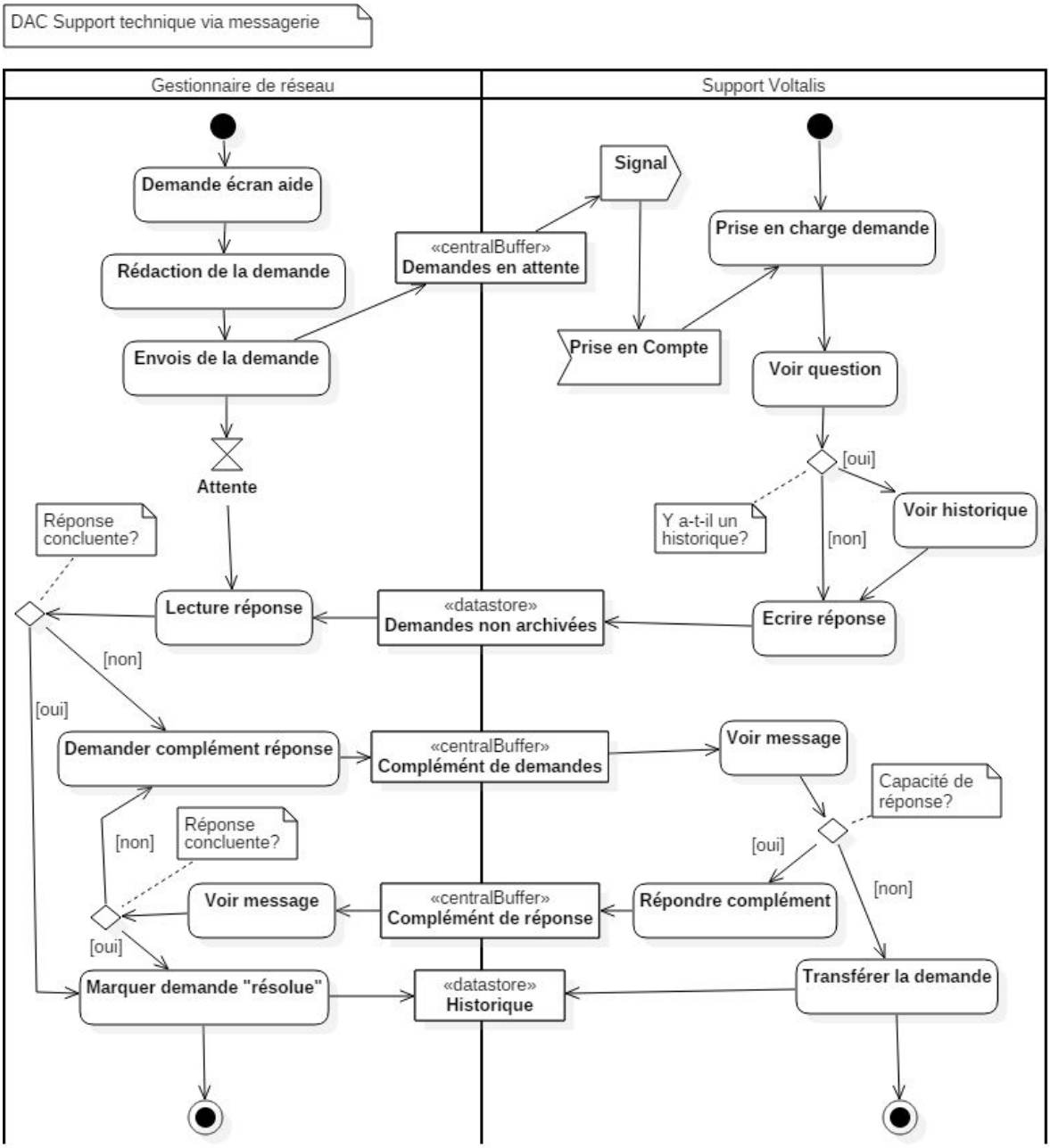
Le GRD reste connecté, le système continue de fonctionner, c'est uniquement l'ingénieur support qui est déconnecté, par exemple sa machine a planté.

- Le GRD est averti que son correspondant est déconnecté
    - Le message lui dit qu'il sera remis en relation avec un ingénieur aussi vite que possible
- Sa question est replacée dans la liste des requêtes par ordre chronologique
- Les ingénieurs connectés voient la requête réapparaître comme prioritaire.
- Un autre ingénieur la récupère et voit l'historique.

Les diagrammes d’activité

Scénario nominal et alternatif

Il me semble ici intéressant de compléter le cas d'utilisation présenté plus haut (« Communiquer avec le support technique Voltalis via une messagerie ») par son diagramme d'activité correspondant. Celui permet de représenter le déroulement de ce cas d'utilisation.



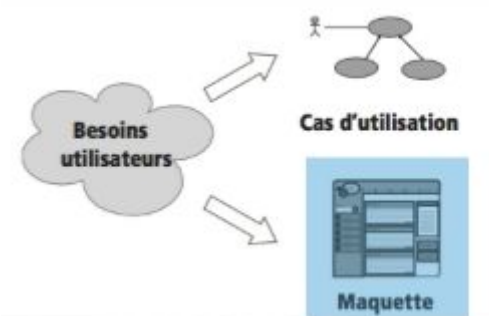
Scénarios d’erreur

A la demande mon maître de stage, les scénarios d’erreur feront partie d’une version ultérieure de l’application.

Les maquettes

Introduction

Toujours dans la logique de la démarche de Pascal Roques, et ayant déterminé en détails les différents cas d'utilisation et relations entre eux, l'étape suivante est la réalisation de **maquettes**.



J'ai réalisé à quel point celles-ci étaient cruciales, car elles permettent de mettre visuellement les différents cas d'utilisations et les liens IHM permettant de les rendre accessible, et permettent donc de plus facilement repérer les erreurs de conception ainsi que les points de désaccord entre le client et le développeur.

C'est à la suite de leur réalisation que les dernières demandes de modification furent faites par Vincent Agami : sans elles des erreurs se seraient immiscées dans mes interface et potentiellement se seraient reflétées dans la conception de mes entités cela auraient demandées des remaniements bien plus tard pendant la phase de développement, ce qui aurait entraîné une perte de temps dommageable !

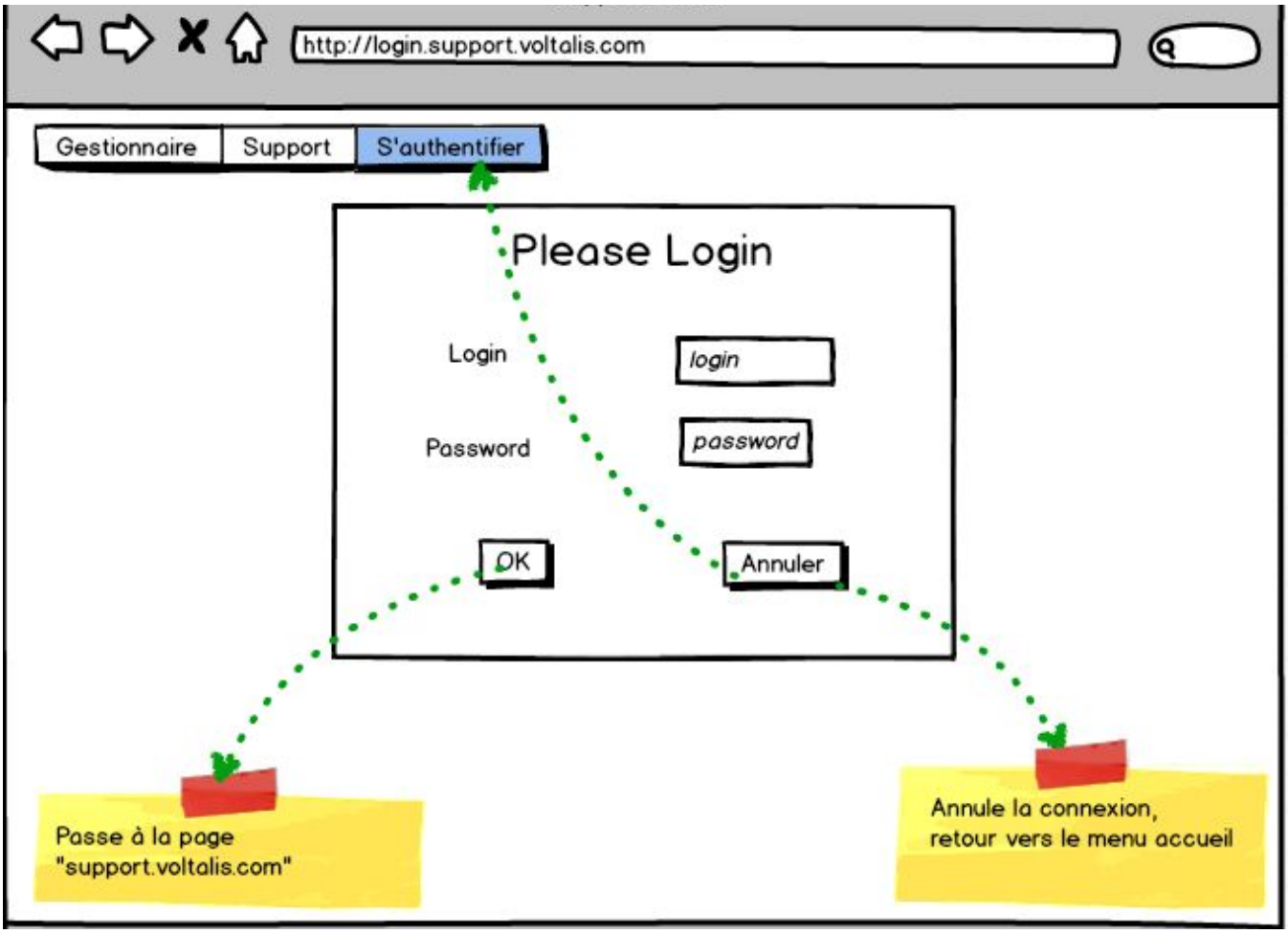
Note : Comme je possédais déjà un prototype de l'application (du moins l'affichage de la carte), j'ai utilisé balsamiq.com pour réaliser ces maquettes, elles permettaient à mon tuteur de mieux visualiser les éléments nouveaux dans un contexte qu'il connaissait déjà. De plus ce logiciel en ligne est gratuit et très simple à prendre en main, ce qui m'a permis de gagner du temps.

Voici ci-dessous un exemple de maquette correspondant aux cas d'utilisation « **Communiquer avec le support technique Voltalis via messagerie** ».

J'ai utilisé trois couleurs de flèches, afin de vous faciliter la lecture, les couleurs : elle permette d'une manière plus clair de suivre un scénario du regard, et d'une manière générale :

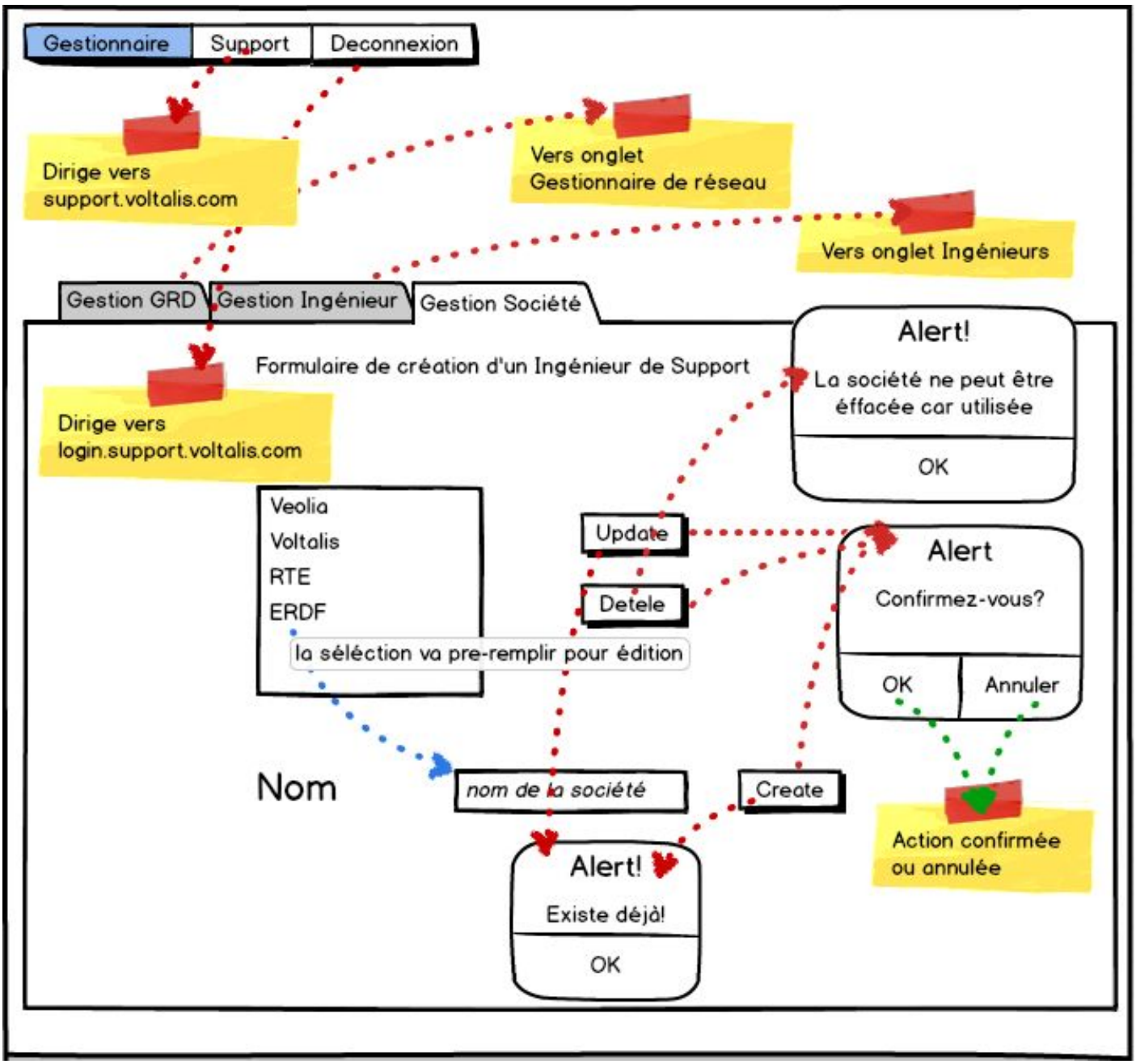
- **Rouge** : l'ouverture de fenêtre, frame, modale
- **Bleu** : ce qui permet de remplir, pré-remplir des champs de texte
- **Vert** : plutôt une action logique : archiver, annuler, ou autres actions sans rapport aux routes

Interfaces : Accueil et s'authentifier

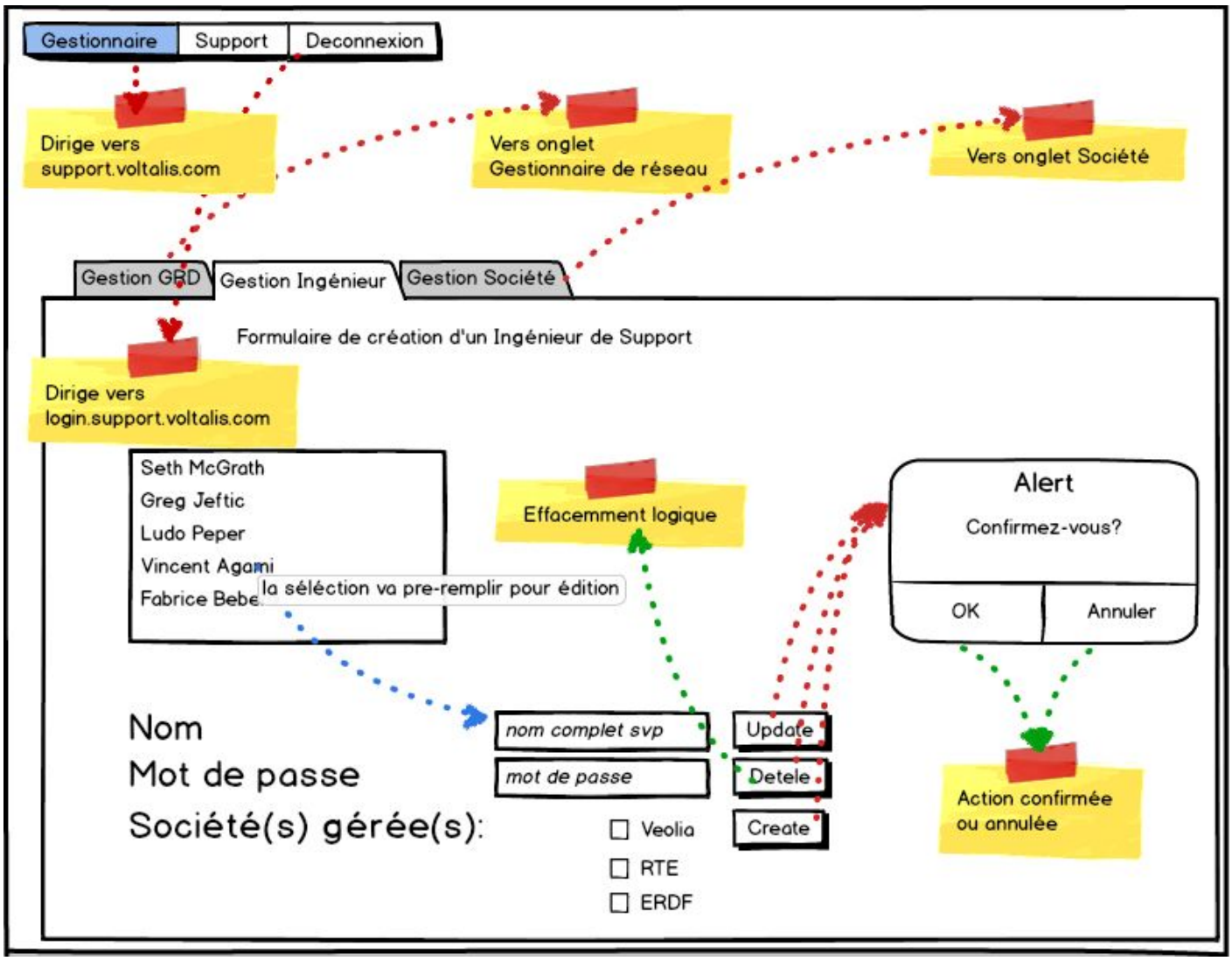


Interfaces : administrations des utilisateurs

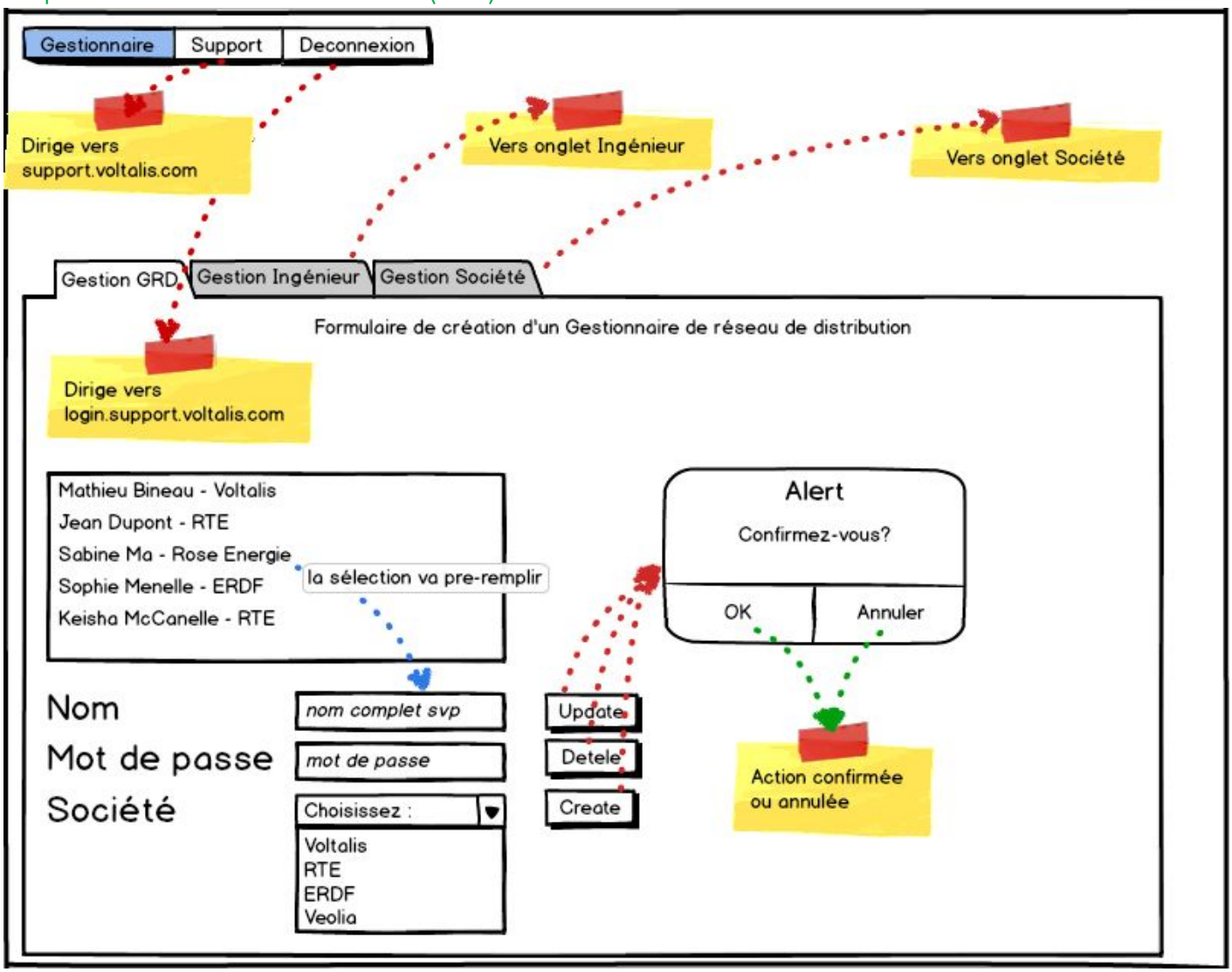
Maquette : « Gestionnaire des sociétés »



Maquette : « Gestionnaire des ingénieurs support »

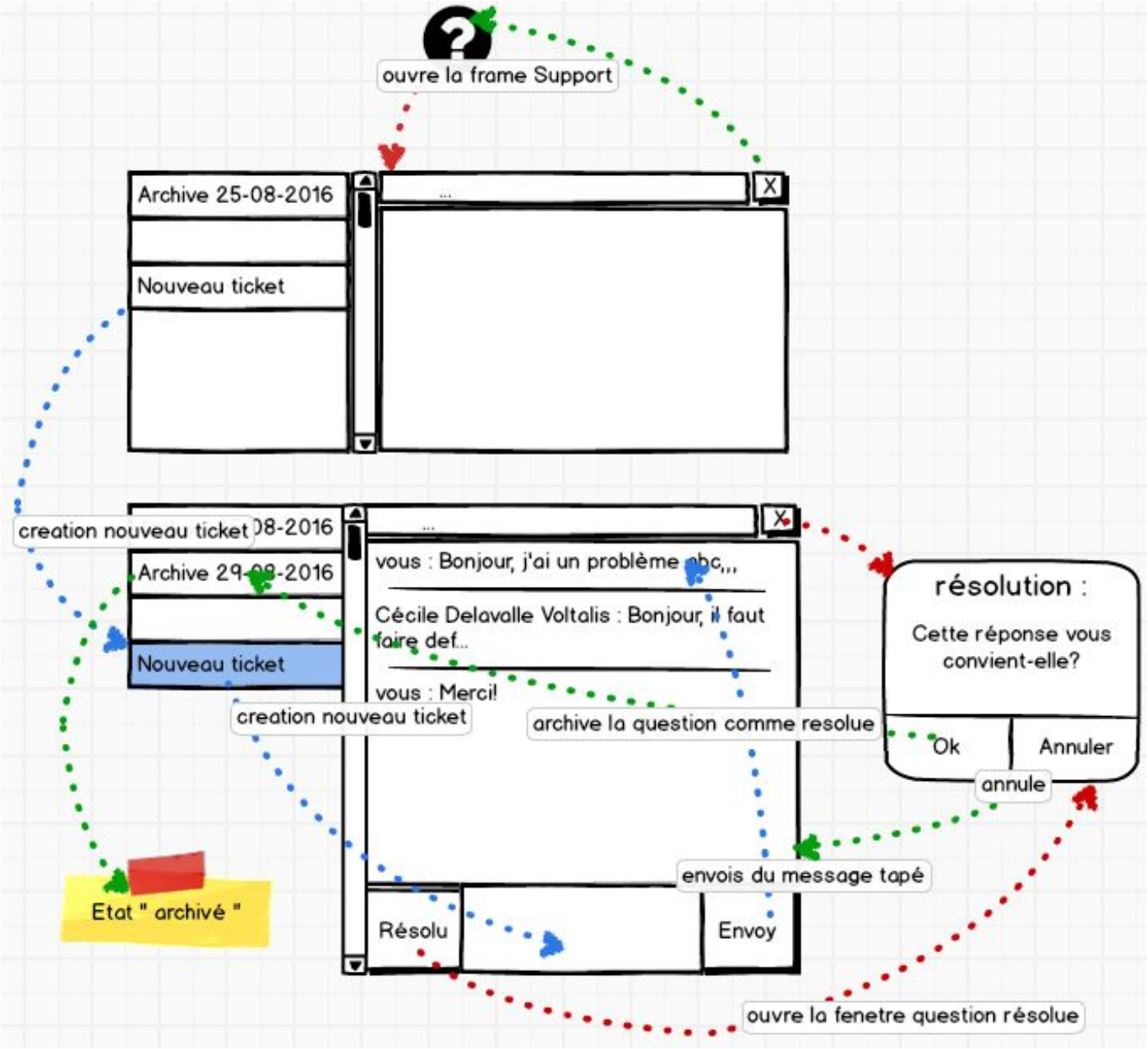


Maquette : « Gestionnaire des clients (GRD) »



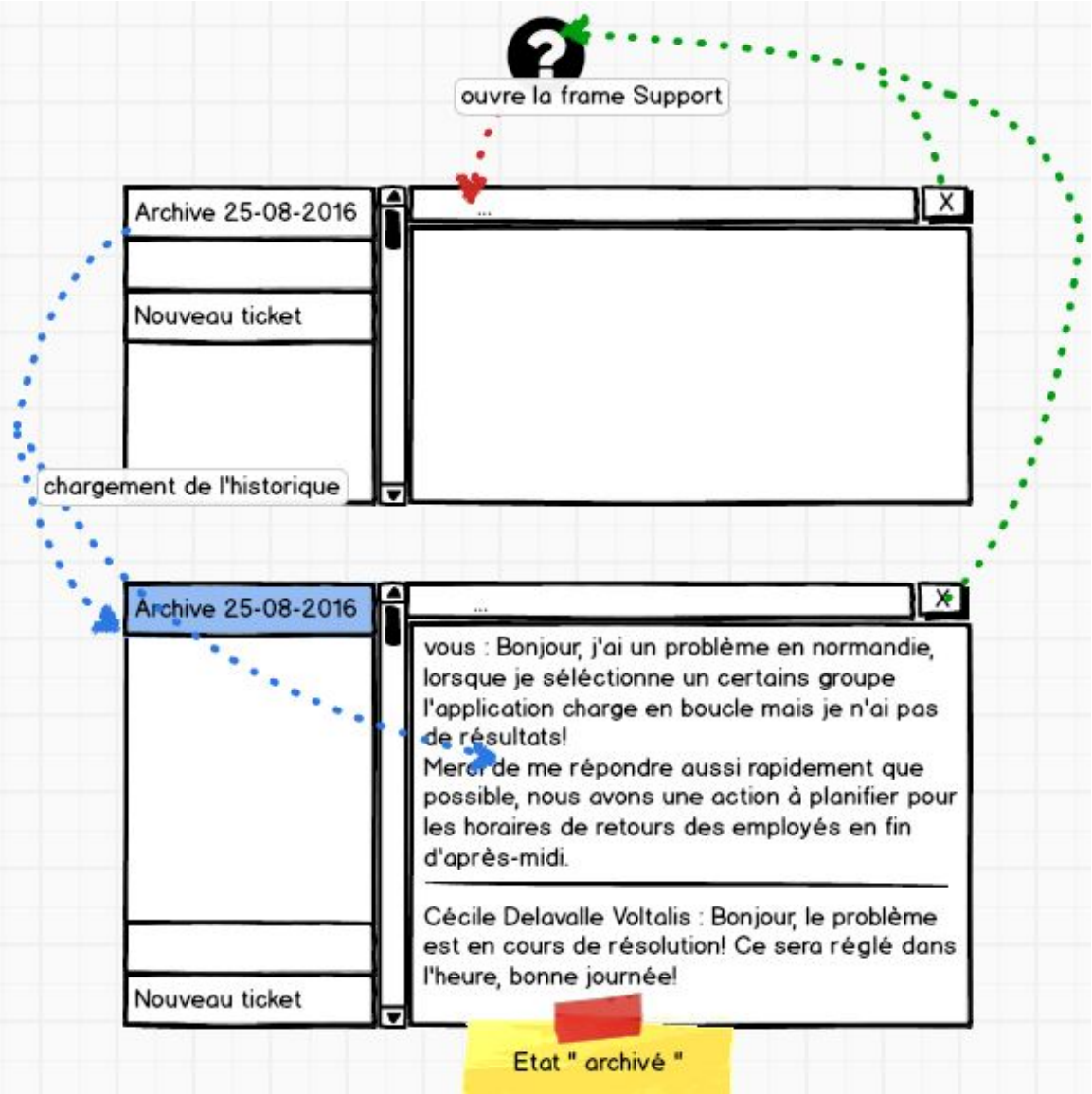
Maquette « demander de l'aide »

Cette première partie illustre les actions « créer un nouveau ticket » puis l'archiver une fois résolu.

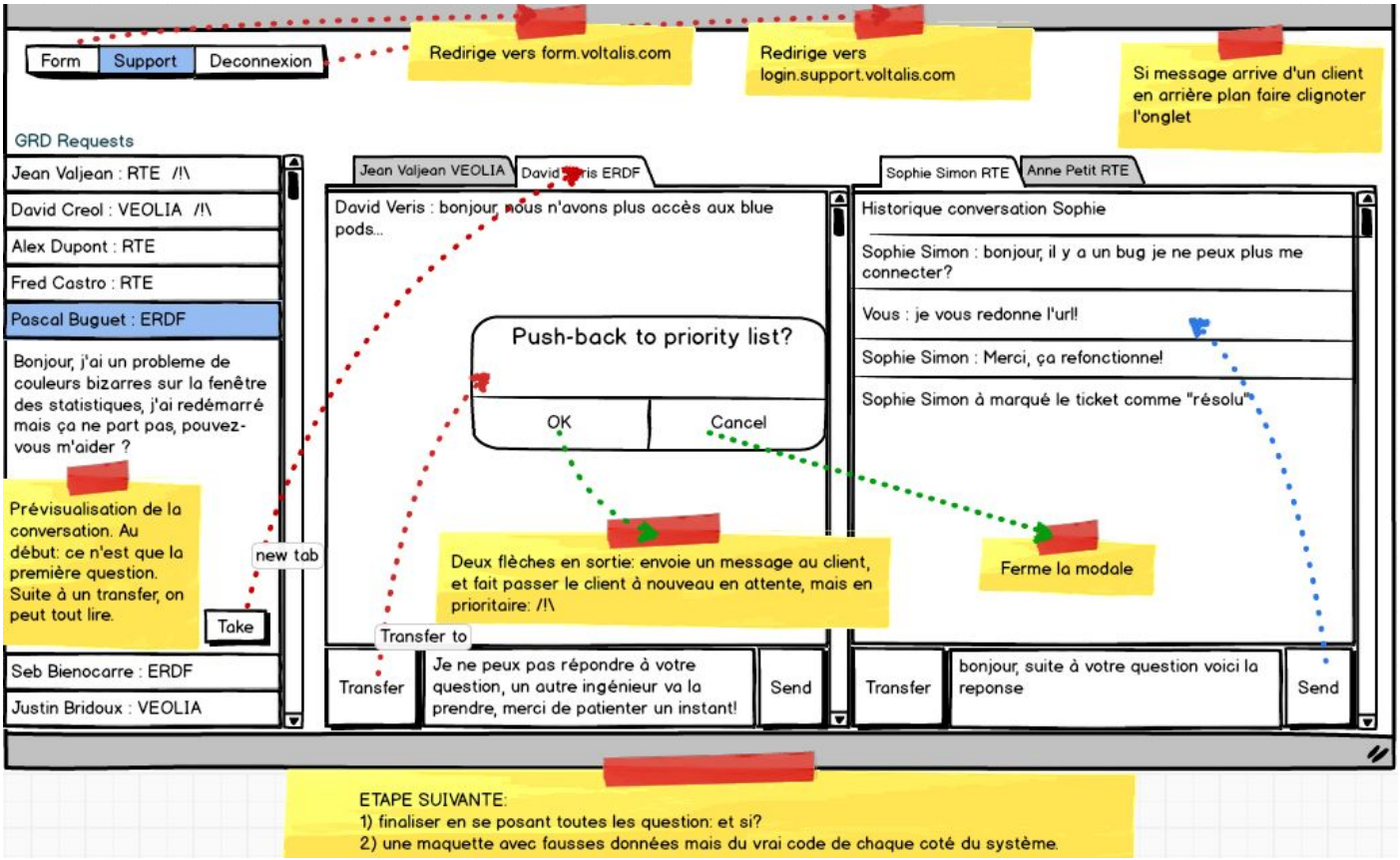


Maquette « voir une archive »

Cette deuxième partie illustre l'action « voir une archive »



Maquette « Répondre aux demandes d'aide des GRD »



Diagrammes de navigation

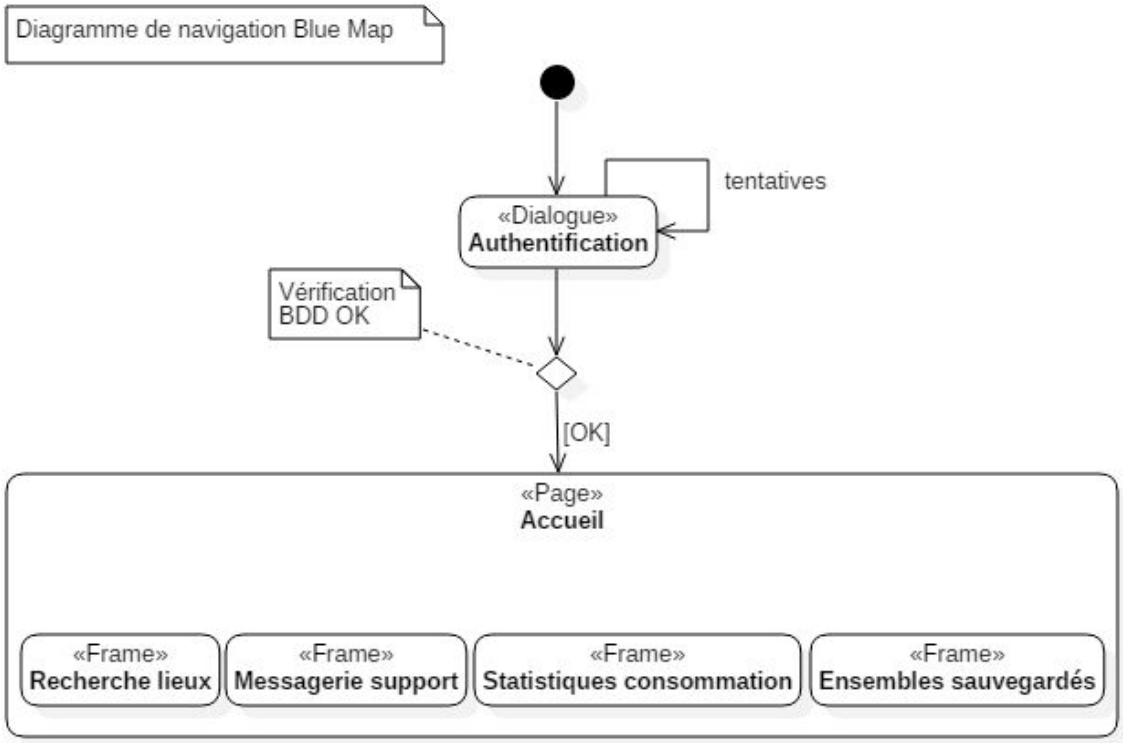
Après avoir détaillé les cas d'utilisation qui décrivent les interactions des acteurs avec le système, nous avons pu en faire découler des maquettes, censées représenter chaque «états» (le système à un instant «t»), étant dans ce contexte une page, une frame, un élément de page, une boîte de dialogue, une exception ou un connecteur.

Nous allons désormais pouvoir relier ces états entre eux via la représentation de transitions, c'est à dire des passages d'un écran à un autre, correspondants aux actions générées de l'utilisateur par ces interactions, (Et ce via des boutons, des liens etc...) pouvant être soumis à des conditions.

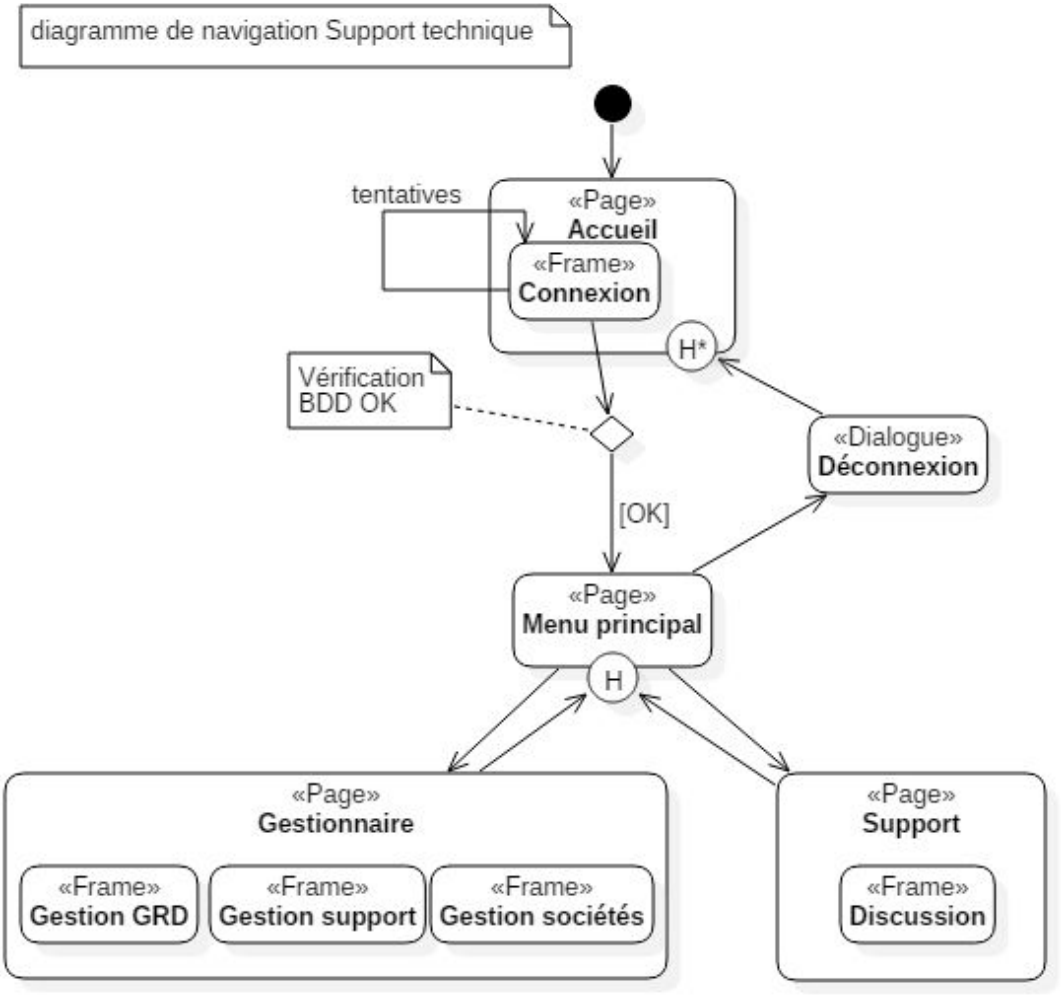
La représentation graphique et dynamique de ces «états-transitions» se fera grâce au diagramme de navigation : celui-ci sert à représenter le cheminement de l'application (où couche contrôle) entre les différents écrans (IHM).

Afin de maîtriser ce travail de modélisation, il va falloir le séparer en ensembles les plus indépendants possibles. Pour mon application deux parties sont clairement visibles, l'application Blue Map coté gestionnaire de réseau et côté support technique.

Transitions IHM gestionnaire de réseau



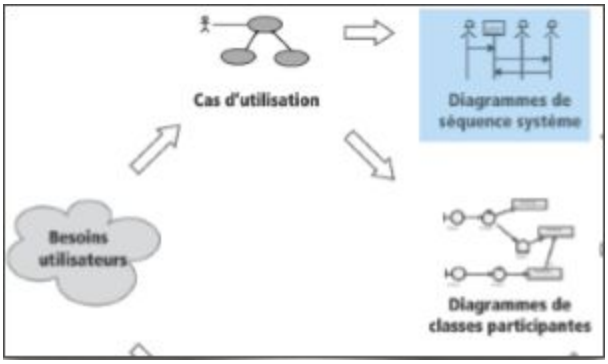
Transitions IHM support technique



Les diagrammes de séquence système

Introduction

Chaque cas d'utilisation est décrit textuellement de façon détaillée, mais donne également lieu à un **diagramme de séquence** simple représentant graphiquement la **chronologie des interactions** appelées « **messages** » entre **les acteurs** et le **système** vu comme une boîte noire, dans le cadre du scénario nominal.

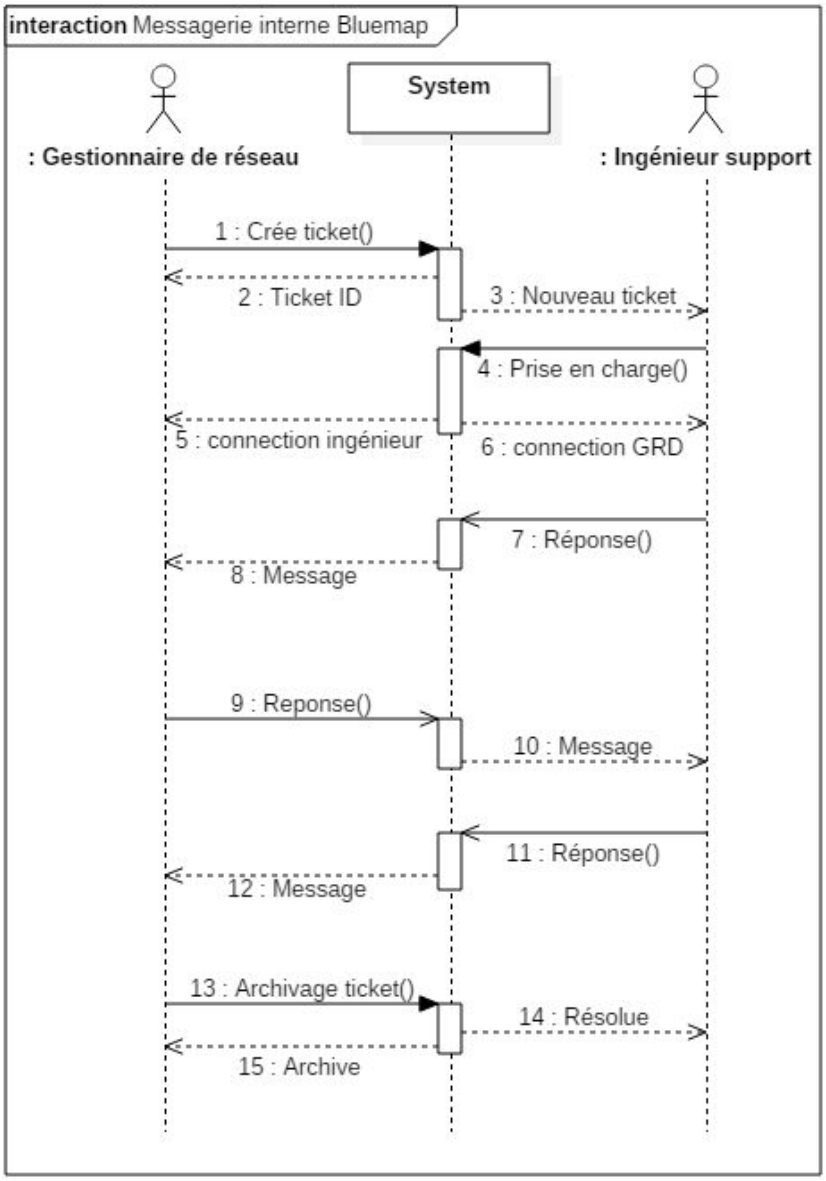


Nous appellerons ce diagramme : « diagramme de séquence système ». La dimension verticale du diagramme représente le temps, permettant de visualiser la séquence d'enchaînement des actions, et de spécifier la naissance et la mort d'objets. Les périodes d'activité des objets sont symbolisées par des rectangles, et ces objets dialoguent par le biais de messages.

C'est en isolant ces messages que nous allons pouvoir les représenter graphiquement sur des diagrammes de séquence système. Selon les cas, il est préférable de représenter chaque scénario d'un cas d'utilisation par autant de diagrammes de séquence, quoiqu'il est tout à fait possible de représenter un scénario alternatif au sein du même diagramme avec un simple message «OK» ou «KO» du système.

Diagramme de séquence Support technique temps réel

Dans notre cas d'utilisation choisi comme exemple ici, « Communiquer avec le support Voltalis via une messagerie », il est intéressant d'élargir le diagramme en y figurant les deux acteurs « gestionnaire de réseau » et « ingénieur support » :



Fonctionnalité navigation dans la carte

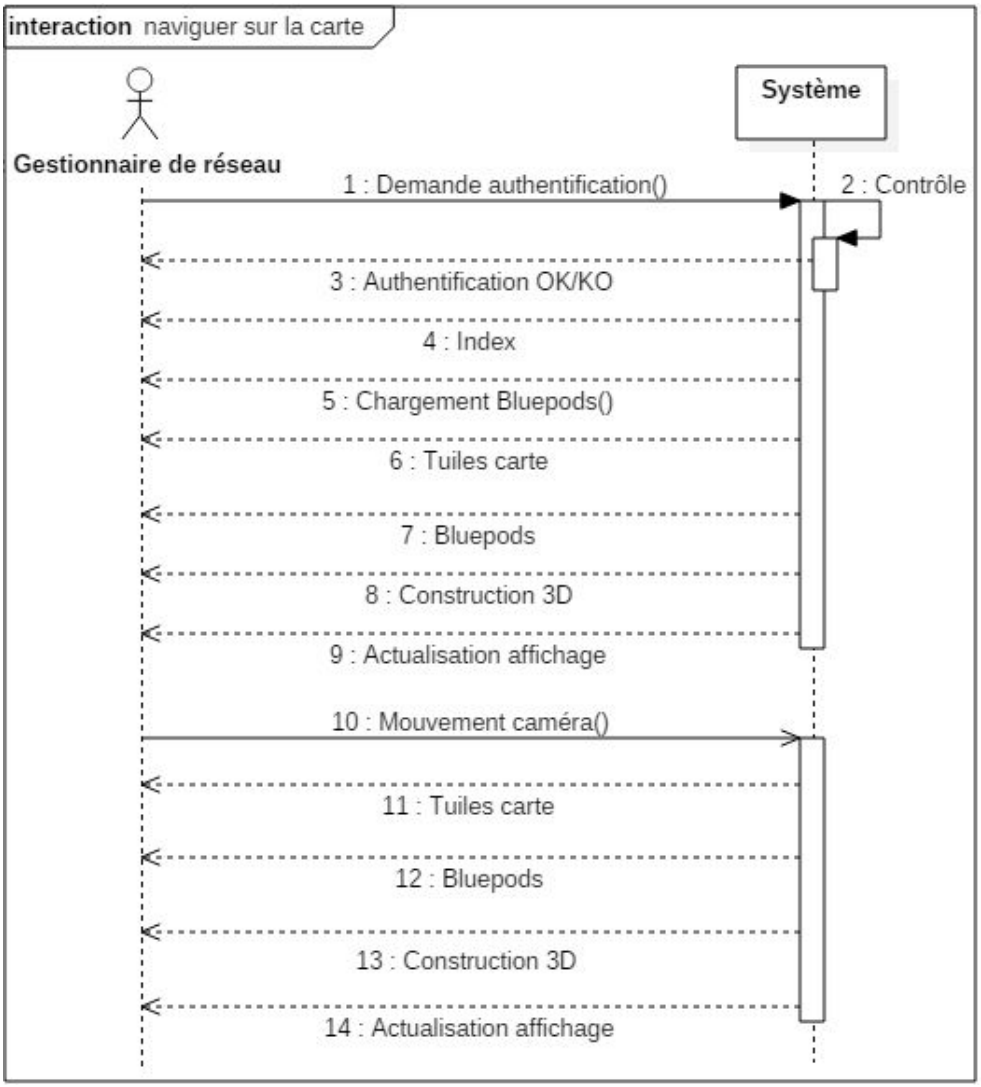
Description textuelle de ce cas d'utilisation

Deux acteurs : un utilisateur, un GRD (gestionnaire de réseau de distribution), par exemple de RTE ; le système Blue Map.

1. Le GRD se connecte à l'application
    a. Le système reçoit une requête et répond une fenêtre modale demandant nom d'utilisateur et mot de passe
        i. Le GRD renseigne ses informations d'authentification
        ii. Le système tente d'authentifier :
            1. OK → Il envoie l'index et requête l'API F4 Map avec la géolocation de départ de la caméra ;
            2. KO → Il envoie un message d'erreur et déconnecte le client.
2. Le GRD peut voir la carte et tous les Bluepods.
3. Le système actualise les tiles et la modélisation 3D de manière asynchrone lorsque la vue est réajustée par le GRD.
4. Le GRD sélectionne le bouton ré-aligner la caméra vers le nord
    a. Le système calcule la différence d'angle :
    La caméra tourne de sa position vers celle requise pour être alignée au nord,
    et ce à la vitesse prévue par le script côté front.
5. Le GRD passe d'un affichage 2D à 3D en pressant sur le bouton à cet effet.
    a. Lors du passage de 2D à 3D le serveur récupère les données topographiques et les renseignements cadastraux afin de générer la modélisation 3D.

Diagramme de séquence « naviguer sur la carte »

Ce qui va nous permettre de déterminer les messages échangés entre l'acteur et le système dans un diagramme de séquence



Les différentes opérations caméra possible;

- Suite à un mouvement, récupération des tuiles « 3D » (contenu 3D sur des tuiles ou bien informations permettant de générer de la 3D) et 2D (dessins sur la carte) du serveur de tuiles,
- Génération de la carte en 3D (faces et textures) sur le navigateur en JavaScript,
- Envoi au moteur 3D des « buffer » les contenant,
- Rendu 3D.

Fonctionnalité moteur de recherche

Les objectifs

- Le GRD tape dans une zone de saisie un nom correspondant à la ville qu'il cherche
- 10 résultats maximum forment une liste de réponses
- La carte est automatiquement déplacée sur le 1er résultat
- Le GRD peut choisir un autre résultat si le premier n'est pas pertinent
- La carte est alors déplacée sur le lieu de son choix

Recherche d'une solution technique

Pour y arriver, j'ai décidé d'utiliser l'API nominatim d'open street map (OSM). Car l'API F4 Map est basée sur celle-ci et qu'elle est libre. Etant seul sur ce projet, il aura fallu me documenter sur son fonctionnement en guise de documentation pour l'API F4 Map.

J'ai effectué une requête sur OSM et surveillé les requêtes/réponses entre le navigateur et leur serveur. Après avoir lu la documentation officielle Nominatim, il m'a semblé important de voir le fonctionnement plus concrètement depuis le débogueur du navigateur, ce qui m'a permi d'implémenter cette fonctionnalité dans Blue Map plus rapidement.

Ma démarche fût comme suit :

1. Récupérer l'URL de la requête en GET envoyée au serveur en récupérant celle-ci dans l'historique de la séquence des échanges entre le navigateur et le serveur OSM,
2. Variabiliser le paramètre de celle-ci afin de l'utiliser pour mon moteur de recherche,
3. Analyser le tableau d'array JSON renvoyé par le serveur pour renseigner la méthode de pilotage de ma carte.
4. Organiser et afficher les 10 premiers résultats dans une frame sur le site,
5. Enfin après tous les tests intermédiaires et un résultat final satisfaisant, j'ai encore effectué plusieurs tests afin de vérifier différents cas qui pourraient être sources de problèmes ou voir si des cas spécifiques à l'utilisation que pourrait en faire M. Bineau sont renseignés sur le serveur OSM.
6. Il a fallu respecter la charte graphique déjà établie donc le maquettage a été inutile, j'ai ajouté un input sur la vue via un template avec Jade qui affiche un champs de saisie et un bouton « OK » ;
7. Après validation via la touche entrée ou clic sur le bouton « OK », le formulaire est récupéré et les valeurs saisies sont envoyées via une requête AJAX[2] au serveur OSM/Nominatim.
8. OSM/Nominatim me répond la correspondance avec un JSON que je parcours pour récupérer nom, type et Bounds/Centroid...
9. ...que j'affiche pour chaque résultat dans un élément de liste les uns à la suite des autres sous le l'input de recherche.
10. Le premier résultat de la liste est envoyé au moteur de la carte qui y déplace la caméra.

[2] **AJAX** est l'acronyme de *Asynchronous JavaScript and XML*, ce qui, transcrit en français, signifie « JavaScript et XML asynchrones ».

11. Si le premier résultat n'est pas pertinent ou que le GRD souhaite en voir un autre, il le sélectionne et ses coordonnées sont envoyées à nouveau au moteur de la carte.

Il faut noter que chaque résultat **est en fait un objet javascript**, qui est **passé en paramètre à la méthode de pilotage** de la carte, cette méthode sait comment accéder **aux données membre** contenant les coordonnées.

Déroulement des échanges entre les systèmes: fiche de description textuelle

L'utilisateur doit pouvoir saisir un nom de lieu, voir les résultats de sa recherche et piloter la carte sur l'un d'eux en le sélectionnant.
Lorsqu'un résultat est trouvé, la carte est directement sur le premier de la liste.

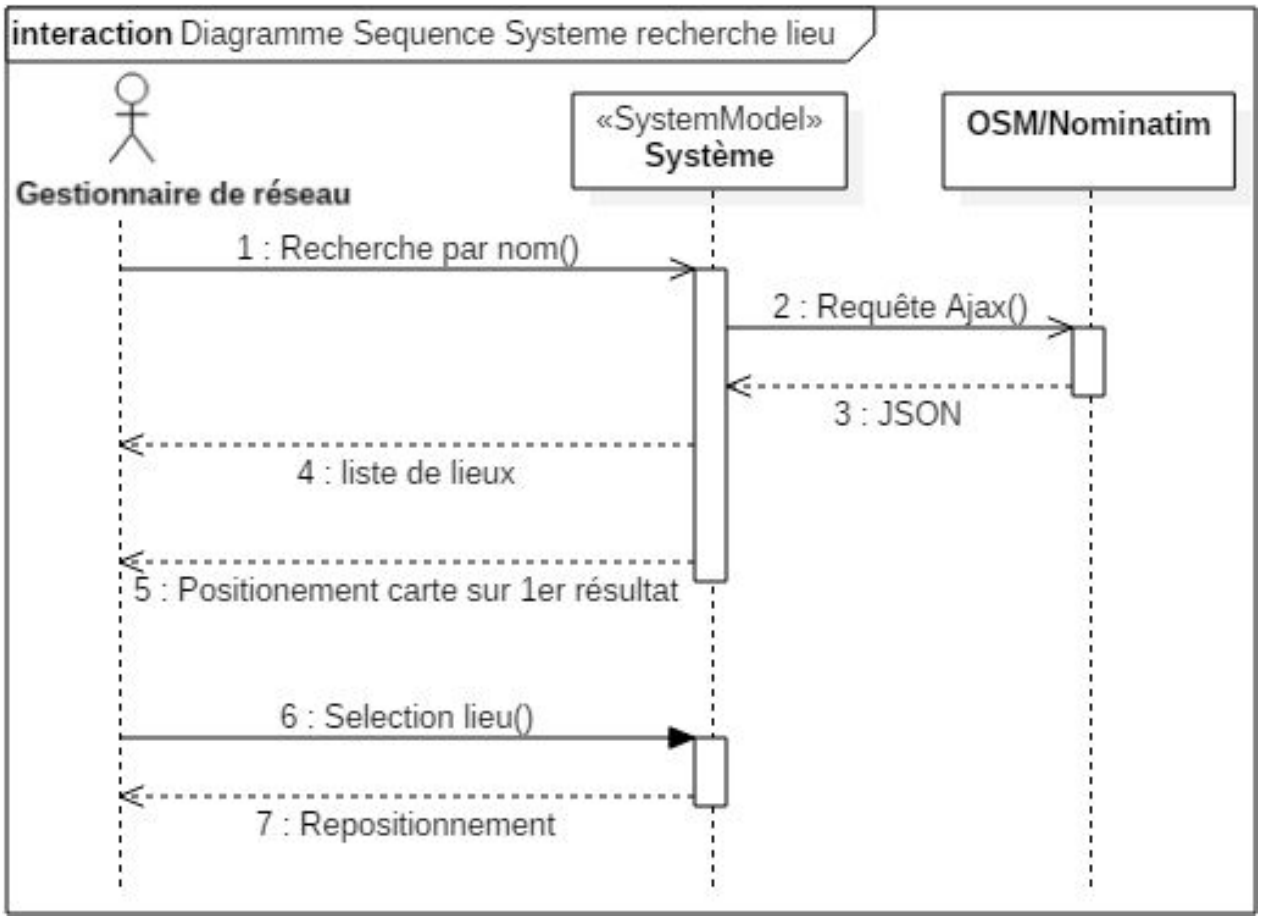
On distingue donc 3 acteurs :

1. Le GRD
2. Le système
3. L'API OSM/Nominatim

- Le GRD tape le lieu dans une input box de la carte, sur le navigateur
- Le GRD appuie sur entrée → le système envoie une requête à OSM/Nominatim
    - OSM/Nominatim fait une recherche dans sa base
    - OSM/N renvoie le résultat sous forme JSON
    - Le code coté navigateur décode le JSON : récupère les coordonnées
        - Et pilote la carte sur le premier résultat
        - Un test est effectué → le lieu sélectionné est-il un point précis sur la carte?
            - Si c'est un point précis :
                - On récupère les coordonnées « Centroid », on double les latitude et longitude auxquelles on retire 100 mètres
            - Sinon :
                - On récupère les coordonnées « Bounds »
                - La caméra est déplacée sur les coordonnées sélectionnées dans la liste de résultats par le GRD

Diagramme de Séquence Système « Recherche de lieu »

La fiche de description textuelle donne le diagramme de séquence suivant :



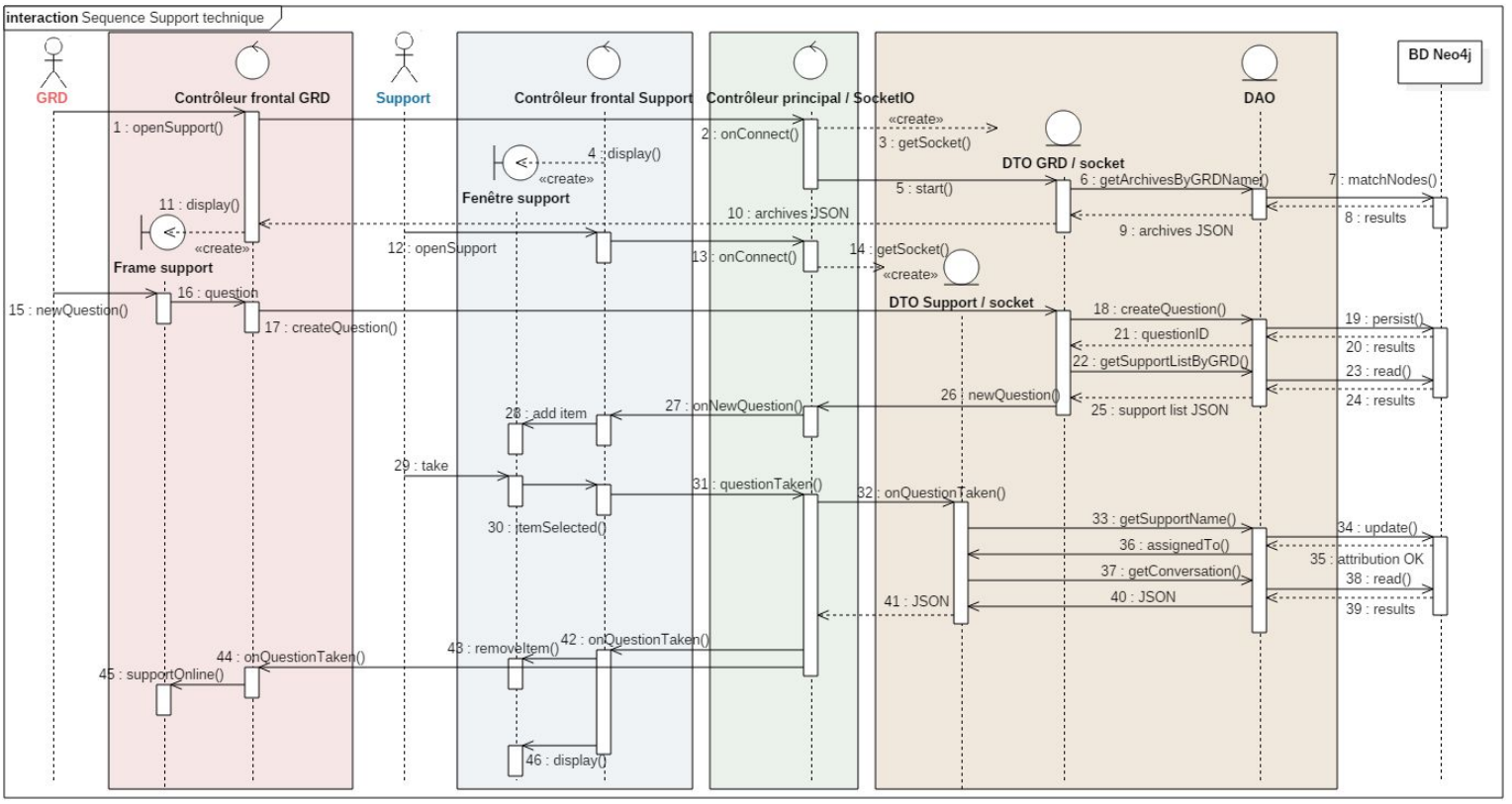
CONCEPTION DE L'APPLICATION

Diagramme de séquence détaillé

Retour vers la démarche d'analyse et de conception de Pascal Roques. Après avoir représenté les diagrammes de séquence système dans la partie «analyse» qui schématisaient le système informatique comme une simple boîte noire, nous allons nous intéresser plus en détails aux différentes couches métier de ce système, en y voyant désormais un ensemble d'objets en interaction.

Nous aurons donc des objets utilisateurs du système et des objets du système : des pages, boîtes de dialogue (view), des contrôleurs (controller) et des entités (model) interagissant entre eux.

Diagramme de séquence détaillé représentant la création d'un nouveau ticket puis la prise en charge de celui-ci par un ingénieur support, et donc les parties «READ», «UPDATE» et «CREATE» du CRUD :

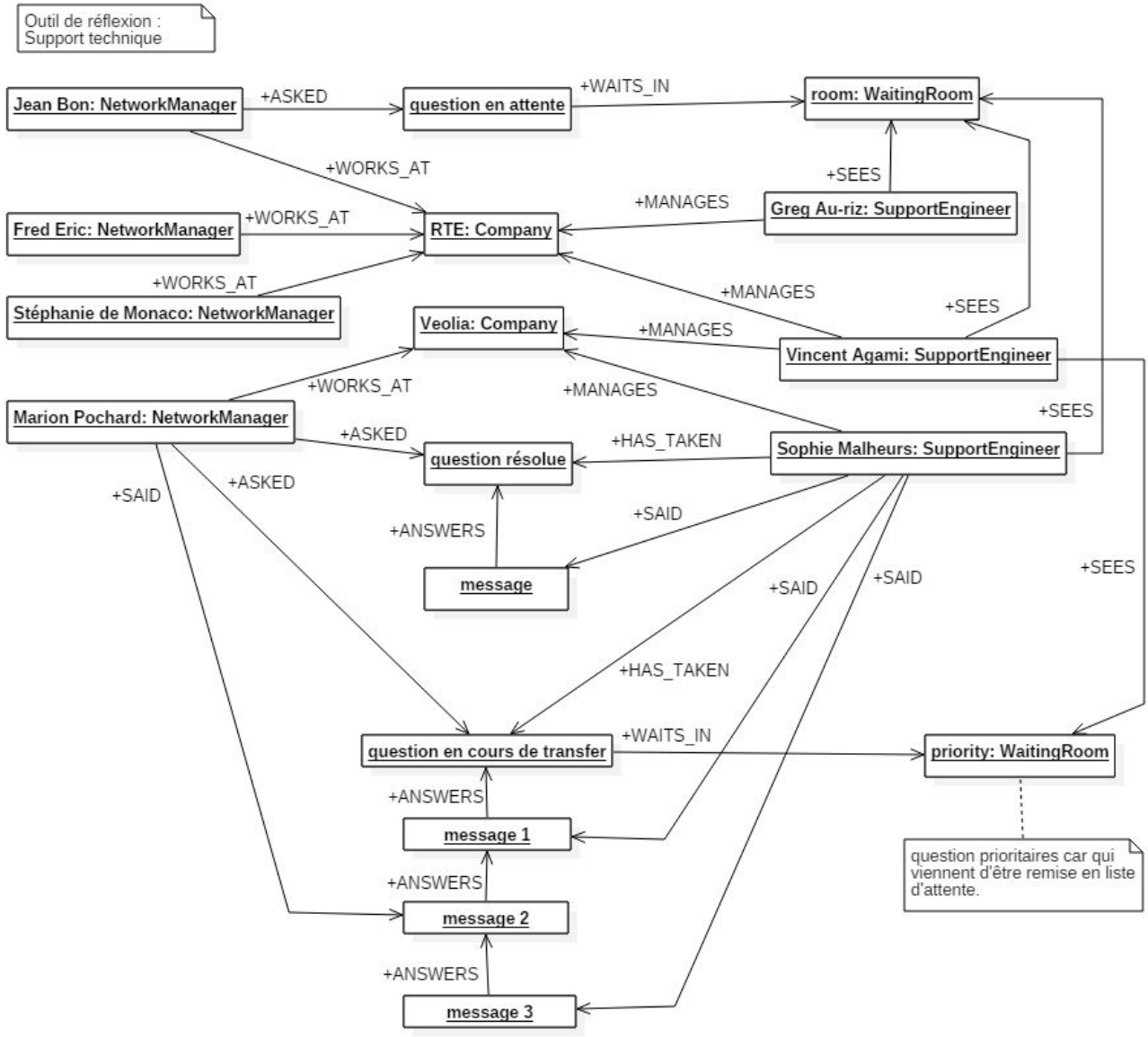


Fonctionnalité de support technique temps réel

Un outil de reflection : modèles d'instances simplifié

Avant de parler de diagramme de classes, j'ai voulu créer un modèle peuplé des différents concepts, on pourrait imaginer un diagramme d'instances mais extrêmement simplifié ! Après quelques conversations avec mon maître de stage j'ai très vite eu la matière pour créer cet outils à partir de mes différentes notes.

Il permet de survoler un pseudo-état rapidement lisible et mettre en lumière les questions de règles de gestion importantes mais peu visibles. Ce diagramme d'objet ne servant qu'à mettre en lumière un les relations entre eux, je n'ai pas inclus les attributs!



Les diagrammes de classes

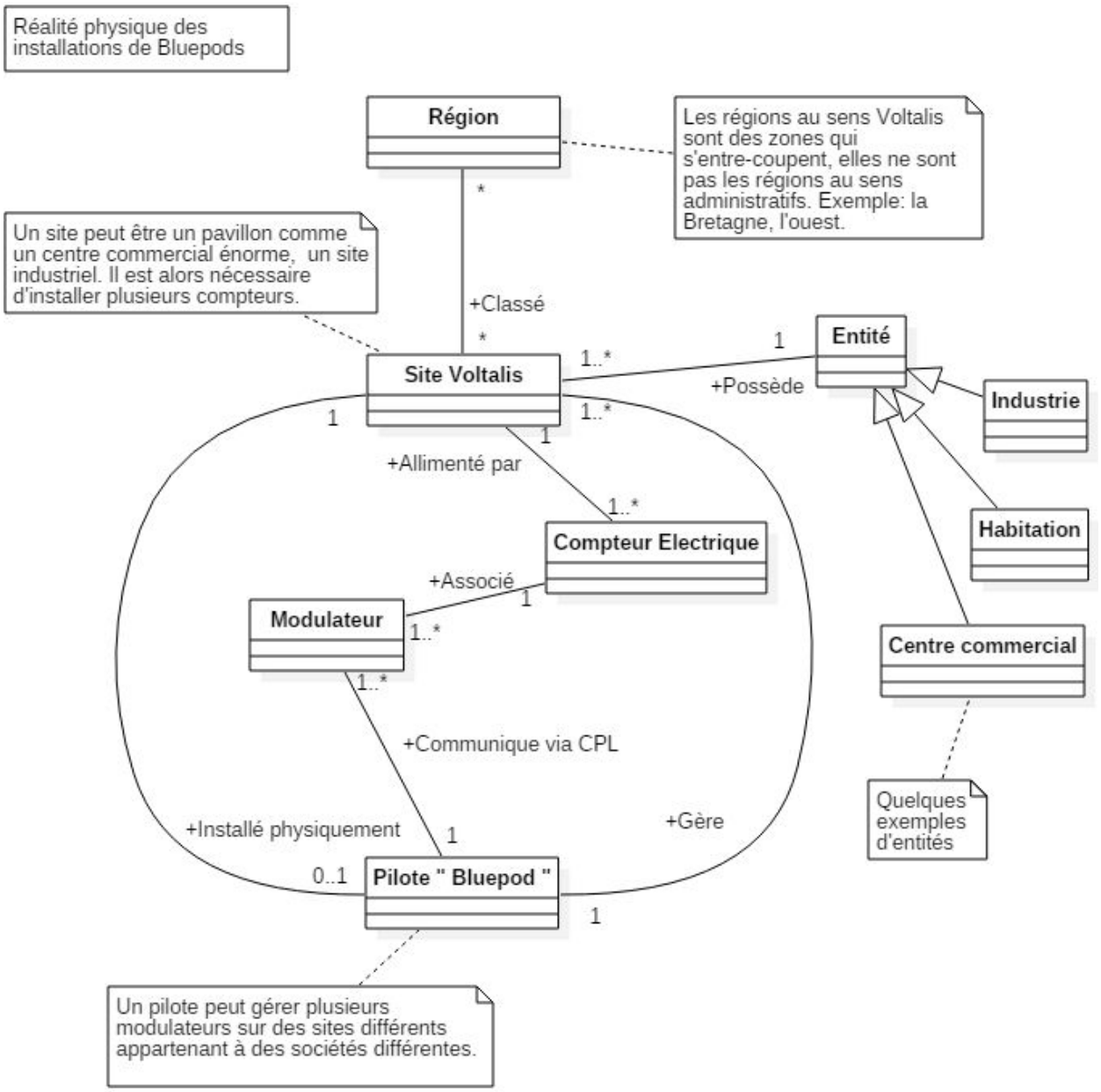
Introduction illustrée par des classes du métier de Voltalis

Identification des **concepts** du domaine L'étape typiquement orientée objet de l'analyse est la décomposition d'un domaine d'intérêt en classes conceptuelles représentant les **entités significatives** de ce domaine. Il s'agit simplement de créer une **représentation visuelle** des **objets du monde réel dans un domaine donné**. Si l'on emploie la notation UML, il s'agit d'un ensemble de diagrammes de classes dans lesquels on fait figurer les éléments suivants :

- les **classes conceptuelles** ou les **objets du domaine** ;
- les **associations** entre classes conceptuelles ;
- les **attributs** des classes conceptuelles.

La réalité des infrastructures

Ici nous allons analyser et présenter le Diagramme de Classes (DCL) sur les entités de l'application Blue Map, celles-ci conceptualisent la réalité des infrastructures qui sera représenté visuellement ci-dessous (je vulgarise volontairement pour traduire le **concret** via le **symbolisme** exprimé par la notion de Classes en langage orienté objet) :



Le pilote « Bluepod » transmet alors les relevés et interventions à une base de données en temps réel, à intervalles régulier.

Diagramme de classes : fonctionnalité support technique

A ce stade de la conception, il est utile de poser un diagramme de classes, nous ne somme pas encore dans la création de code que ce soit bien clair, mais plutôt la réalisation d'un modèle du domaine que l'on traite, la mise en relations des concepts métier liés à notre module de support technique.

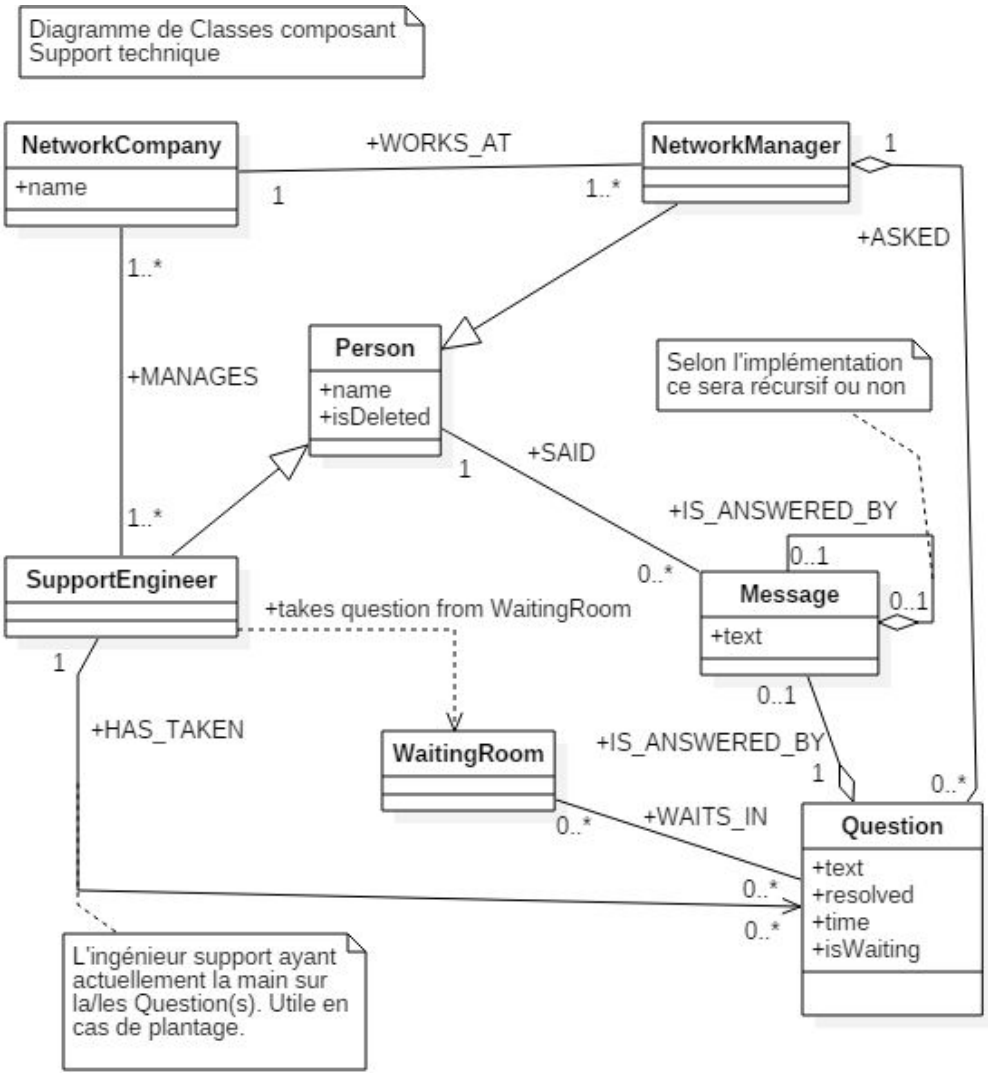
Un modèle du domaine est une sorte de carte des concepts d'un domaine. Il faut donc absolument utiliser le vocabulaire de ce dernier pour nommer les classes conceptuelles et leurs attributs.

Ajout des associations et des attributs

Une fois que l'on a identifié les concepts fondamentaux, il est utile d'ajouter :

- les associations nécessaires pour prendre en compte les relations qu'il est fondamental de mémoriser ;
- les attributs nécessaires pour répondre aux besoins d'information.

Ici les entités support technique :

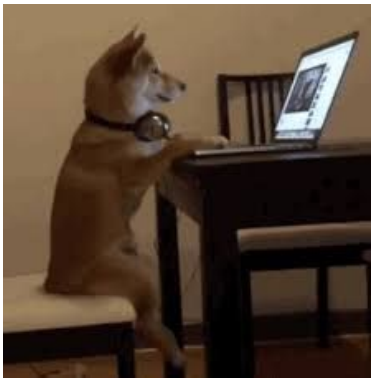


Récursivité : remarque sur le schéma

Il sera bon de noter que après l'étude ci-dessus, il y a plusieurs questions qui se posent, notamment celle de la récursivité dans le modèle. Il me semble pour le coup intéressant de voir les associations suivant un modèle de type graphe, et donc d'utiliser la convention de nommage de ces derniers qui est l'écriture majuscules en Snake case, c'est à dire chaîner les mots avec le caractère « underscore » pour l'écriture des « relations ».

En effet la récursivité est gérable en SQL, mais passé une d'arborescence de 4/5 niveaux les performances deviennent vraiment déraisonnables, et cela empire exponentiellement à chaque niveau supplémentaire.

Par contre en base de données graphe, ça entre parfaitement dans le fonctionnement normal du modèle.



C’est parti !

DEVELOPPEMENT

Technologies générales utilisées

Pourquoi utiliser Node.js ?

Node.js est un environnement d'exécution de code JavaScript qui peut (c'est souvent le cas) s'exécuter sur un serveur. Node.js profite des possibilités d'extensions du moteur JS V8 de Google pour ajouter des fonctionnalités à JavaScript et lui permettre entre autre de gérer les échanges http.

Il est donc possible et facile de créer un serveur web avec Node.js, mais son utilisation ne se limite pas aux seuls serveurs. Node.js est très modulaire ce qui permet de limiter son empreinte mémoire.

La vitesse d'exécution du serveur est très rapide lorsqu'il n'est pas surchargé mais au contraire que l'on maîtrise la nécessité de ce qui doit être implémenté côté serveur et ce qui peut être délégué au client.

Un modèle événementiel sans blocage I/O

Node.js utilise un modèle événementiel, sans blocage I/O qui le rend léger et efficace, parfait pour des applications en temps réel avec un volume de données important qui fonctionne sur les appareils distribués. L'exemple suivant va illustrer cela :

```
database.query("SELECT * FROM hugetable",
      function(rows) { // Code de traitement du résultat quand il arrivera
            var result = rows;
      });
console.log("Hello World");
```

Ici, contrairement à d'autres langages où la méthode query n'admettrait qu'un argument, avec Node.js, le deuxième argument est une fonction dite de callback qui sera invoquée lorsque la requête aura terminé son traitement. Ici le callback est une fonction anonyme, ce qui est une pratique très courante en JavaScript.

Si l'on considère que database.query() fait partie d'une bibliothèque asynchrone de Node.js, au lieu d'avoir un long temps d'attente avant de voir le message du logger affichant « Hello World » et qui plus est bloquant pour l'ensemble des clients connectés au serveur, car Node.js s'exécute sur un thread unique globalement, Node.js va envoyer la requête au serveur de la base de données et continuer l'exécution des tâches programmées.

Il retournera en attente d'un événement dans la boucle qui écoute le retour du serveur de base de données à chaques fois qu'il n'a plus d'autres tâches à effectuer. Lorsque le serveur de base de données retournera un résultat, un événement appellera alors la fonction en callback.

Même langage de programmation pour le serveur et le client: JavaScript

Justement parmi ses grandes qualités est celle que le langage serveur est le même que celui côté client!

Ainsi l'on peut très aisément déplacer des tâches vers le client afin de soulager le serveur, où si une évolution des besoins amène une tâche à demander plus de sécurité et donc être exécutée par le serveur. Car, oui, tout étant en javascript, une partie de l'application sera envoyée au client et cela pourrait poser des problèmes de sécurité ! D'où le besoin de maîtriser la répartition des couches en connaissance de causes.

Énorme bibliothèque de composants développés par la communauté

Une autre grande qualité est l'énorme bibliothèque de composants développés par la communauté, grâce à NPM (Node Package Manager) il est possible de démarrer très rapidement le développement de l'application elle-même et ne pas devoir tout recoder « from scratch ».

L'arrivée de Node.js a permis une meilleure interaction entre développeurs back-end et front-end.

Création d'un serveur web avec Node.js en 5 lignes !

Il est également très simple à mettre en route et seulement quelques lignes de code suffisent à avoir un serveur HTTP fonctionnel !

```
var http = require('http');
http.createServer(onRequest).listen(8080);
Function onRequest(req, res) {
      res.writeHead(200);
      res.end('<h1>Hello World !</h1>');
}
```

Un outil puissant pour mieux gérer les callbacks : Async.js

Puissant...

Async.JS est une librairie Javascript permettant au programme de contrôler le flux d'exécution asynchrone de ses fonctions. Très pratique lorsque l'on gère plusieurs requêtes vers des bases de données. Ses qualités ne s'arrêtent pas là, car elle fournit un lot de fonctions utilitaires.

Ces fonctions offrent au programmeur des alternatives à ne pas négliger lorsqu'il se retrouve empêtré dans sa propre boue, à devoir traiter des tableaux et des objets tout en étant embarqué dans un enfer rempli de callbacks imbriqués les uns dans les autres.

Async.JS est né pour combattre ce qu'on appelle le Callback Hell. Ce terme est parfaitement illustré sur le site callbackhell.com.

...mais dangereux!

Async permet une gestion en cascade des opérations asynchrones dans Node.js mais il est tout à fait possible de l'utiliser dans le navigateur. Il y a toutefois des règles qu'il faut veiller à respecter, notamment veiller à ce que les fonctions utilisées soit bien asynchrones ou passer un paramètre de type «setImmediate» qui lui indiquera qu'elles ne le sont pas. Les risques sont soit de provoquer un «stack overflow» soit de faire crasher l'application et ne même pas s'en rendre compte ce qui provoquera des bugs très difficiles à diagnostiquer plus tard!

Un exemple issu de la documentation officielle

```
async.parallel([
    function(callback){ ... },
    function(callback){ ... }
], function(err, results) {
    // optional callback
};
```

Un exemple d'utilisation, en admettant «`callback`» une fonction définie plus haut dans le script, l'appel des callbacks sera fait dans un bloc plus simple à lire, le traitement des erreurs l'ensemble de celui-ci effectué à la fin, mais pas représenté ici. Par contre en cas d'erreur, l'exécution est stoppée.

Technologies : web html5 JQuery Twitter BootStrap etc.

HTML

J'ai utilisé l'hypertext markup language, généralement abrégé HTML, qui est le format de données conçu pour représenter les pages web. Ce langage a été conçu avec XML et est donc de type «balisé». HTML permet également de structurer sémantiquement et de mettre en forme le contenu des pages, d'inclure des ressources multimédias dont des images, des formulaires de saisie...

jQuery

jQuery est une bibliothèque Javascript open-source permettant de faciliter l'écriture des scripts. Elle permet de par une utilisation front-end, et tous comme Javascript, de manipuler le DOM (document object model), de créer des événements, des effets visuels, la manipulation des feuilles CSS etc...

Twitter Bootstrap

Twitter Bootstrap est une bibliothèque contenant des codes HTML et CSS, ainsi que des extensions Javascript optionnelles. C'est un des framework front-end les plus utilisés. Il permet de facilement styliser sa page web en lui donnant des rendus épurés. avec Twitter BootStrap pour profiter de l'organisation de ma page sous forme de grille et ainsi l'organiser aisément.

CSS

Les feuilles de style en cascade (cascading style sheet) ou plus généralement abrégées CSS, forment un langage informatique qui décrit la présentation visuelle des documents écrit dans un langage balisé (HTML , XML...). Il devient de plus en plus évolué en terme de dynamisme, notamment grâce à ses «webkit-transitions», permettant un changement dynamique des valeurs de propriétés, le tout sur une période de temps donnée.

Git, Tortoise, Putty

Git

Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. J'en ai eu besoin afin de gérer les version de mon projet, en cas d'erreur de pouvoir retrouver la dernière version valide en une ligne de commande.

TortoiseGit

TortoiseGit est un client Git.

PuTTY

PuTTY est un émulateur de terminal doublé d'un client pour les protocoles SSH, Telnet, TCP brut...

Jade et Stylus, moteur de template et CSS pre-processor

Jade Template

Jade est un moteur de template pour Node.js qui permet d'écrire du code HTML sans avoir à placer les balises. On utilise plutôt les ids et les classes des objets du DOM que l'on a plus souvent défini en CSS. C'est l'indentation, un peu comme en Python qui permet de définir les relations parents-enfants.

Stylus

Fonctionne très bien avec Stylus, un pré-processeur CSS fonctionnant de la même manière, l'indentation! Ils permettent tous les deux d'utiliser des fonctions «mixin» qui permettent de faire un modèle réutilisable dans le script. Stylus permet de variabiliser des éléments redondants comme les couleurs par exemple.

Socket.io

Socket.IO est une bibliothèque JavaScript pour des applications web temps-réel. Il permet une communication du temps-réel bidirectionnel entre clients et serveurs. Il possède deux parties: une bibliothèque côté client qui fonctionne dans le navigateur, et une bibliothèque côté serveur pour Node.js. Ces deux composants ont des API quasi identiques. Tout comme Node.js, son fonctionnement est événementiel.

Apache Maven

Apache Maven est un outil pour la gestion et l'automatisation de production des projets logiciels Java en général et Java EE en particulier. L'objectif recherché est comparable au système Make sous Unix : produire un logiciel à partir de ses sources, en optimisant les tâches réalisées à cette fin et en garantissant le bon ordre de fabrication.

Maven utilise un paradigme connu sous le nom de Project Object Model (POM) afin de décrire un projet logiciel, ses dépendances avec des modules externes et l'ordre à suivre pour sa production.

J'ai dû l'utiliser pour compiler les procédures stockées dans Neo4j, appelées «user procédures», qui sont sous la forme d'un fichier .jar, il m'a fallu pour cela copier les bibliothèques de développement de Neo4j dans le répertoire du projet. Elles sont open sources.

Moteur de recherche

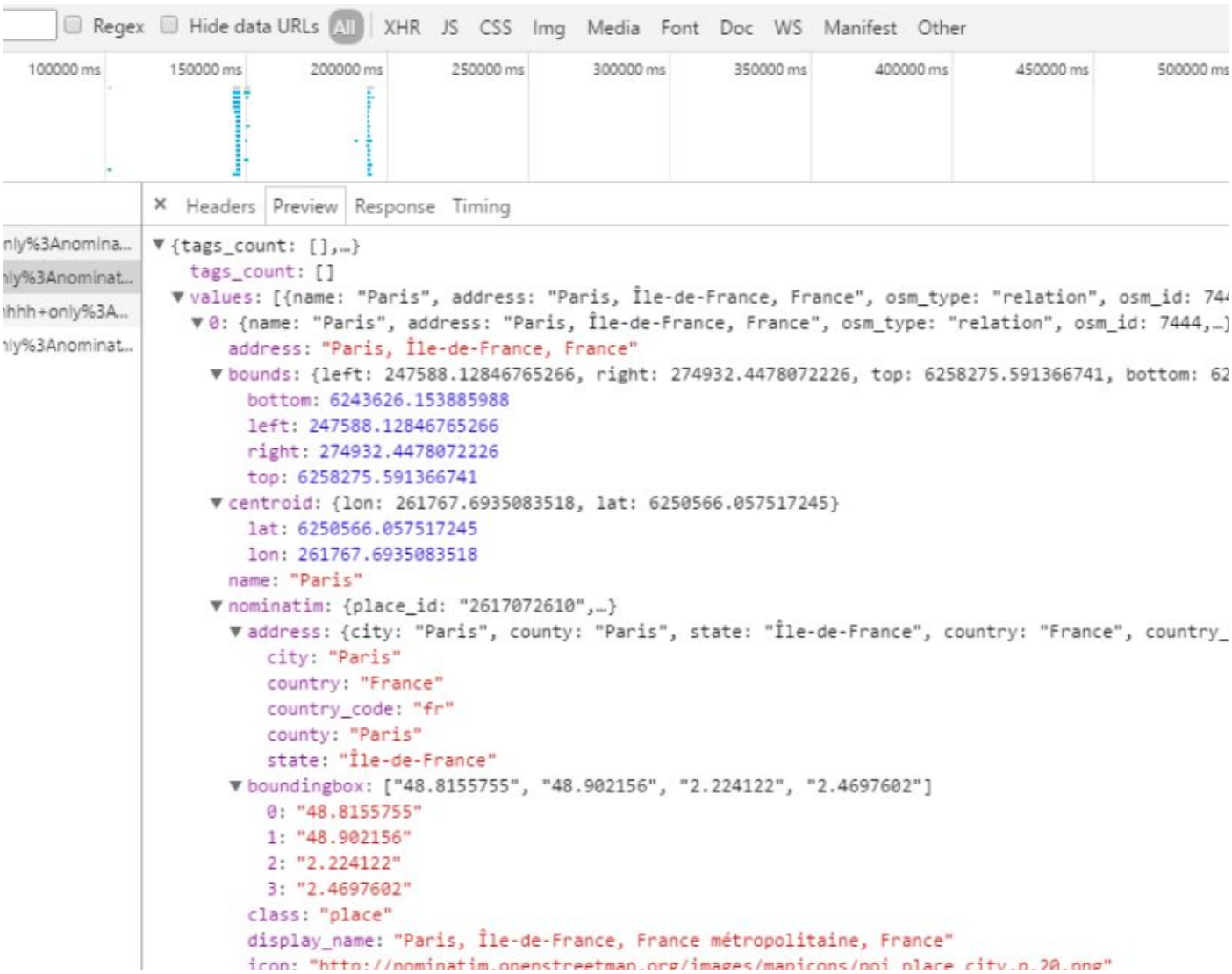
Difficulté : code existant de Blue Map non documenté

Il m'a fallu en premier lieu un moyen de me référer à un moteur de recherche fonctionnant de la même manière que F4 Map, OpenStreetMap est donc l'endroit idéal car l'API OSM a été le modèle lors de la création de l'API F4 Map.

Donc après avoir consulté la documentation officielle de OSM/Nominatim j'ai fait une recherche sur leur API et récupéré la requête envoyée à leurs serveur sur laquelle j'ai remplacé la chaîne de caractères que j'ai envoyé par une variable, j'ai en fait « variabilisé » la requête pour la rendre réutilisable pour n'importe quelle recherche afin de l'implémenter dans mon moteur de recherche.

Retro-engineering d'une requête sur OSM/Nominatim

L'étape suivante, il m'a fallu voir et décomposer la réponse, au format JSON, pour pouvoir créer mes conditions permettant d'utiliser les informations utiles pour piloter ma caméra sur la carte de Blue Map.



Implémentation de la requête OSM/N et pilotage de la carte

J'ai ensuite pu créer les éléments rendant disponible le moteur de recherche pour les utilisateurs, ici pas de maquette car j'ai dû me cadrer sur la charte graphique existante.
Voici le code de création des éléments du moteur de recherche.

Écoute d'événements sur la zone de saisie de recherche de lieux

Récupère le noeud texte de l'élément lié à l'id search :

```
var $searchLocation = $("#search");
```

Permet de détecter tout changement sur la valeur de la variable $searchLocation et la passer en argument de la fonction « updateResults() » :

```
$searchLocation.on('change',
                   function () { updateResults($searchLocation.val()); });
```

Au dessus, ce qui est très intéressant, c'est qu'en une ligne de code à la fois je surveille tout événement de modification sur la valeur d'une variable et lorsque cela arrive, j'appelle une fonction de callback qui va alors exécuter la fonction « updateResult() » avec la nouvelle valeur!

Mise à jour des résultats de recherche dans une liste interactive

Ici on a une fonction appelée « updateResults() » à laquelle on passe en argument la variable $searchLocation qui va être utilisée dans une requête Ajax vers OSM/N et dont le résultat sera récupéré en callback et envoyé au template « resultsTemplate » via un objet. On recherche la classe « .headerSearchResultContainer » que l'on vide, pour ajouter « resultsTemplate » dynamiquement peuplé avec un « each » directement par le moteur de Jade.

Encore une fois on utilise l'écoute d'un événement qui sera le click sur un des éléments de la liste contenu dans « resultsTemplate », celui-ci déclenchera la fonction de callback qui va envoyer le bon objet selon son id au moteur de pilotage de la carte :

```
function updateResults(searchLocation) {
	$.get(	'http://search.f4map.com/search?q=' + searchLocation
		+ 'only%3Anominatim'
		+ '&rawNominatim=true&withWay=true&sort=-score&language=fr-FR',
	function (data, status) {
		console.log(status);
		var $resultsTemplate = $(resultsTemplate({results: data}));
		$body.find(".headerSearchResultContainer").empty()
		    .append($resultsTemplate);
		$resultsTemplate.find(".searchResultItem")
		    .on("click",function (event) {
		        var osm_id = $(event.currentTarget).data("id");
		        var item = findItemBoundsByOsmId({
		            datas: data.values,
		            osm_id: osm_id
		        });
		        moveCameraToItemResultPosition(item);
		    });
		    if (data.values.length > 0)
		        moveCameraToItemResultPosition(data.values[0]);
		}
	);
}
```

Pilotage de la caméra sur un emplacement de la carte

Ici, la fonction « moveCameraToItemResultPosition() » envoie l'objet qu'elle prends en argument à un bloc conditionnel qui détermine :

- Si l'objet contient des coordonnées en « bounds » c'est à dire un quadrillage de latitudes et longitudes déterminants un carré au dessus duquel la caméra sera positionnée
- Si l'objet ne contient pas de « bounds », on ira chercher les coordonnées « centroid » qui pointent vers unsimple croisement latitude/longitude que l'on va transformer en carré de 100 m^2 pour positionner la caméra de manière plus rapprochée

```
function moveCameraToItemResultPosition(item) {
        var bounds = item.bounds;
        if (bounds) {
               bounds = new api.LatLngBounds(
                       api.LatLng.from900913(bounds.bottom, bounds.left),
                       api.LatLng.from900913(bounds.top, bounds.right)
               );
        } else {
              bounds = new api.LatLngBounds(
                      api.LatLng.from900913(item.centroid.lat, item.centroid.lon - 100),
                      api.LatLng.from900913(item.centroid.lat, item.centroid.lon - 100)
              );
         }
         map.panToBounds(bounds);
}
```

Ici, la fonction permet d'instancier « item » avec un objet en fonction de son ID dans un tableau d'objets :

```
function findItemBoundsByOsmId(options) {
       var item = _.find(options.datas, function (data) {
       return data.osm_id == options.osm_id
       });
       return item;
}
```

Comme dit dans le commentaire de code, détecte un click sur le bouton de recherche et passer la valeur du noeud de l'élément « #search » à la variable « $searchLocation » et la passer en argument de la fonction
« updateResults() » :

```
// detection du click sur le bouton de recherche
$searchBtn.on("click", function () {
       var searchLocation = $("#search").val();
       updateResults(searchLocation);
});
```

Requête vers la base PostGIS pour créer le fichier Sites.json référençant les Bluepods

On va pouvoir récupérer les données nécessaires au positionnement des Bluepods sur la carte avec une requête vers la base PostGIS, qui est une base de données PostgreSQL avec plugin pour requêtes **SIG** trigramme pour Sciences de l'Information Géographique.

```
SELECT osm_id, ST_AsGeoJSON(ST_Transform(ST_PointOnSurface(way), 4326))
       AS centroid
FROM planet_osm_polygon
WHERE building IS NOT NULL
AND building NOT IN ('bridge', 'no', 'false', 'proposed')
AND ST_DWithin(way, ST_Transform(ST_SetSRID(ST_MakePoint($1, $2), 4326),
900913), 500)
AND ((NOT tags ? 'wall') OR tags->'wall' != 'no')
ORDER BY ST_Distance(way, ST_Transform(ST_SetSRID(ST_MakePoint($1, $2),
4326), 900913))
LIMIT 1
```

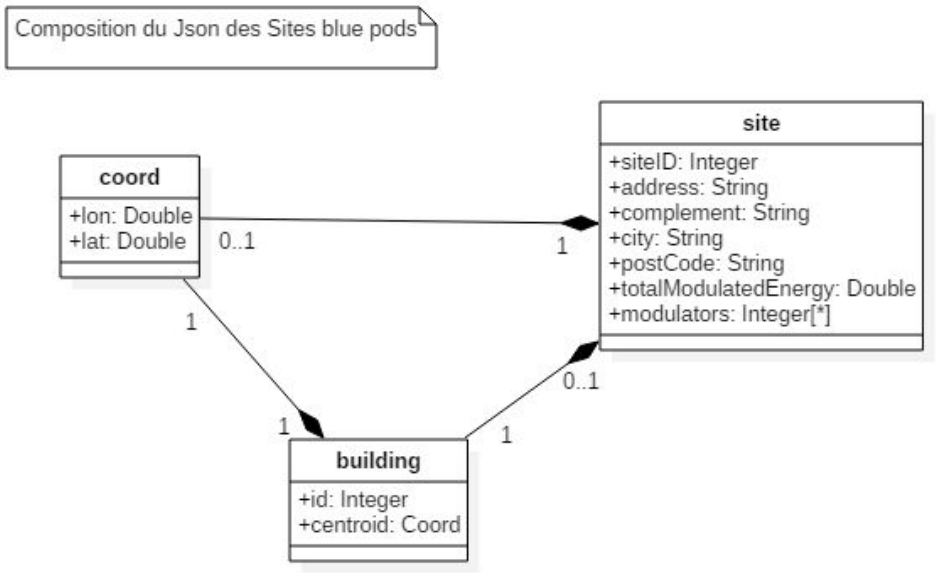
En demandant, parmi les résultats non nuls, qui ne sont pas des ponts, des murs, dans une zone de 500m autour de points envoyés via les variables ($1 et $2) et en filtrant les bâtiments sans toit, le tout en précisant à la base que l'on utilise une projection de Mercator[3] (4326), que l'on souhaite un format type Google Maps (900913), un résultat par itération et le tout au format JSON.

L'application Blue Map en l'état actuel lis les coordonnées des sites Blue Pods dans des fichiers JSON. Le diagramme de classes ci-dessus démontre la **composition**[4] des attributs représentés dans le fichier « sites.json ».

[3] La projection de Mercator est une projection conforme, c'est-à-dire qu'elle **conserve les angles**. Elle a cependant pour effet des **déformations sur les distances et les aires**. En effet, une distorsion s'accroît au fur et à mesure de l'éloignement de l'équateur vers les pôles. Car on transforme une sphère en cylindre.

[4] Une **composition est une agrégation forte** impliquant que : 1/ une coordonnée et un bâtiment ne peuvent appartenir qu'à un seul site et les coordonnées d'un bâtiment lui sont propres 2/ La destruction du site entraîne la destruction de toutes ses coordonnées (le **composite est responsable du cycle de vie des parties**).

Les données seront représentés en JSON sous la forme suivante



Extrait du fichier Sites.json obtenu après composition


```
[
	{
		"siteId":677793,
		"address":" rue robespierre",
		"complement":"rdc droit",
		"city":"BREST",
		"postCode":"29200",
		"modulators":
			[
				"1867193"
			],
		"totalModulatedEnergy":1407.5
	},
	{
		"siteId":621719,
		"address":"1 allee jacque brel",
		"city":"BREST",
		"postCode":"29200",
		"modulators":
			[
				"1286761",
				"1286757"
			],
		"coord":
			{
				"lon":-4.51399,
				"lat":48.3938
			},
		"totalModulatedEnergy":9635.566666666666,
		"building":
			{
				"id":42570597,
				"Centroid": {
					"lon":-4.51412444111349,
					"lat":48.3937906303365
				}
			}
	}
]
```

Interface du moteur de recherche

J'ai utilisé la charte graphique déjà existante dans Blue Map, il m'a simplement fallu créer un nouveau template avec Jade, l'inclure dans le code Javascript et utiliser Stylus, un préprocesseur CSS.

Extrait de code de l'index en Jade

Ce morceau de code représente le champs de saisie utilisé par le moteur de recherche et l'élément dans lequel je l'ai inclus :

```
Header
    .heaterTitle
        img(src=resourceUrl("/images/logo.png"))
        .pitch
            |Load management Map
    .headerLegend
    .headerSearch
        input#search(type='text', placeholder='Rechercher')
        button#searchBtn GO
    .headerSearchResultContainer
```

Extrait de code, le template Jade contenant les résultats

Il est bon de noter la fonction « mixin » qui définit un modèle et permet de le réutiliser par exemple pour des itérations dynamique depuis Javascript :

```
mixin searchResultItem(result)
   .searchResultItem(data-id=result.osm_id)
       i.firstIcon.fa.fa-map-marker
       span.searchResultItem_title= result.name + ", " + result.nominatim.type
       i.lastIcon.fa.fa-angle-right

.headerSearchResultContent
   each result in results.values
       +searchResultItem(result)
```

Extrait de code Stylus, le CSS de la liste de résultats

```
.headerSearchResultContainer
 padding 5px 5px 0 5px
 .searchResultItem
   font-size 13px
   display table
   width 100%
   margin-bottom 5px
   background rgba(255,255,255,0.1)
   cursor pointer
   transition all ease 0.3s
   &:hover
     background rgba(255,255,255,0.3)
   .firstIcon
     width 30px
     text-align center
     display table-cell
   span.searchResultItem_title
     display table-cell
     padding 5px
   .lastIcon
     width 30px
     text-align center
     display table-cell
```

Moteur de recherche terminé

Après avoir terminé mon travail sur la fonctionnalité de moteur de recherche, j'ai utiliser Git une fois les derniers tests validés, pour rendre cette itération du développement de Blue Map disponible dans le repository de générale de l'entreprise.

Git et IntelliJ

En clair lorsque j'ai pris en main les sources de Blue Map, j'ai effectué un «fetch and rebase» (alt + t dans IntelliJ) afin de récupérer toutes modifications éventuelles par un autre développeur jusque là, puis après avoir terminé une parcelle fonctionnelle, j'ai «commit» et ce plusieurs fois afin de garantir le travail déjà achevé.

À la fin du développement du moteur de recherche, tous tests conclus et validé positivement par le client, j'ai «push» le dernier commit sauvegardé dans le repository de mon projet, et donc mon moteur de recherche pleinement fonctionnel.

Choix d'une base de données adaptée

Ici je vais faire l'inventaire de différents modèles de bases de données ainsi que leurs caractéristiques et d'éventuels défauts ou qualités, on fera un choix dans la conclusion de celle que l'on va utiliser comme moteur de notre module de support technique.

Relational (SQL) Databases: SQLite, MySQL, PostgreSQL

Leurs principales qualités sont l'intégrité référentielle, les transactions, le typage des données (règles de gestion), la possibilité de créer des *prepared statements*, qui préparent une seule fois un plan d'exécution pour une requête, mais permettent son application répétée avec des valeurs différentes, les procédures stockés ; les **propriétés ACID** (**atomicité, cohérence, isolation et durabilité**) sont un ensemble de propriétés qui **garantissent** qu'une **transaction informatique** est exécutée de façon **fiable**, sont la raison même pour laquelle les bases Relationnelles sont encore autant d'actualité !

Un plugin PostgreSQL (PostGIS) permet des requêtes SIG (**système d'information géographique,** parfois utilisé pour définir les **« sciences de l'information géographiques »**).

Adapté parce que

Les bases relationnelles sont adaptées à ce type d'outils, en décomposant le problème du support technique par messagerie ont peut facilement y voir des relations d'entités et donc l'utilisation d'un SGBD-R.

Par contre si les besoin auxquels l'outils répond devaient s'élargir avec des fonctionnalitées demandant une modification du modèle, il faudrait alors casser celui-ci afin de mettre en place le nouveau, la gestion de la récursivité est très difficile passé 4 niveaux d'arborescence et les performances deviennent désastreuse, extrêmement coûteux, à prendre en compte.

Key/Value Store (Redis, Riak, etcd_node)

Leurs intérêt peut-être de stocker des configurations destinée à des machines virtuelles pour le cas de etcd par exemple.

Redis possède également un système de requêtes SIG.

Pas adapté parce que

Toute la **partie logique** serait à implémenter par le développeur, cela reviendrait à comparer l'utilisation d'un fichier CSV à MySQL par exemple !

Columnar Store (Cassandra, HBase)

Les SGBD-R (bases relationnelles) fonctionnent ligne par ligne. Si on veut faire du **data warehouse** : on va vouloir agréger des millions de lignes, sur une ou deux colonnes, par sur toutes, sur des tables qui ont souvent plusieurs dizaines de colonnes (puisqu'on dénormalise beaucoup pour améliorer les performance et la facilité d'utilisation par les analystes de données).

Donc les bases relationnelle en ligne ne sont pas adapté car pour recueillir l'information souhaité il va falloir tout charger! Dans le cas présent, la solution: une base en colonne, c'est toujours des tables, mais les données sont stockées en colonnes, la requête ne va chercher que les colonnes nécessaires. Donc rapide. Cassandra permet la création de prepared statements.

Pas adapté parce que

L'intérêt d'une base en colonne est de pouvoir accéder à des enregistrements de données (lignes provenant d'un SGBD-R) **qui ne bougent plus** ! Mais pour des requêtes sur de si grandes quantitées de données que les bases SQL peuvent avoir des problèmes à répondre.
Elles n'ont donc aucun rapport avec une application de support technique (sauf pour l'analyse des performances d'une équipe de support technique, mais ce n'est pas notre sujet).

Document Store (CouchBase, MongoDB)

L'intérêt principal des bases documents comme MongoDB est le traitement qu'elle permettent façon « map-Deploy » et « map-Reduce » ce qui revient à collecter un nombre de données et de faire un traitement statistique ou de très rapidement isoler certains documents, ce que cela signifie c'est que ces bases de données servent principalement à faire de l'**agrégation[5] de documents**. Très pratique dans le monde des études et statistiques. MongoDB possède un moyens de faire des requêtes SIG d'après les formations proposées par la « MongoDB University ».

MongoDB permet aussi l'instanciation horizontale (le **sharding**) de plusieurs bases réduisant les temps d'accès à la base en période où de grands nombre d'utilisateurs sont connectés à un instant « t », mais réduisant les temps d'accès aux données car celles-ci sont éclatées sur plusieurs serveurs, contrairement à la **réplication**. La récupération des données peut alors demander la collecte des documents sur plusieurs serveurs lors d'un mapReduce, les différents serveur devant faire leurs propre plans d'exécutions puis parcourir toute leur structure, la requête devient immédiatement plus coûteuse, il faut donc bien analyser les besoins et faire le bon compromis !

[5] Une agrégation est un **cas particulier d'association non symétrique** exprimant une **relation de contenance**. Les agrégations n'ont pas besoin d'être nommées sur les diagrammes : implicitement elles signifient « contient », « est composé de ».

Il est également possible d'instancier plusieurs serveurs en « Réplication », ils copieront l'ensemble des données présentes sur le serveur primaire, qui servira de point d'entrée et de réponse au driver sur l'application cliente, en cas de défaut sur le serveur principal un vote sera produit entre les autres serveurs qui éliront un autre serveur primaire et à sa réintroduction dans la chaîne le serveur précédemment hors-ligne prendra un rôle « secondaire » dans le réseau.

La fiabilité est alors accrue, les données sont pérenne et restent accessibles même en cas de panne sur un serveur. Par contre on perd en temps d'accès à la base de données car une modification = 3 modifications (ou tentatives par serveurs).

Il est intéressant de parler du théorème CAP ou CDP, aussi connu sous le nom de **théorème de Brewer** dit qu'il est **impossible sur un système informatique de calcul distribué de garantir en même temps (c'est-à-dire de manière synchrone) les trois contraintes suivantes** :

- Cohérence des données (Consistency en anglais) : tous les nœuds du système voient exactement les mêmes données au même moment ;
- Disponibilité (Availability en anglais) : garantie que toutes les requêtes reçoivent une réponse ;
- Tolérance au partitionnement (Partition Tolerance en anglais) : aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement (ou encore : en cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome).

D'après ce théorème, un système de calcul distribué ne peut garantir à un instant « t » que deux de ces contraintes mais pas les trois.

Pas adapté parce que

Toute la **partie logique** serait à la **charge du développeur** ! Les bases documents sont intéressantes pour traiter des collections de documents ; au plus on pourrait l'utiliser afin de stocker un historique des échanges mais il faudrait alors la coupler avec un SGBD-R ou un SGBD-G ou gérer la logique en code, ce qui reste coûteux pour le développeur.

Graph Store (Neo4j, Titan DB)

Les systèmes de gestion de bases de données graphes (SGBD-G) référencent des noeuds. Chaque noeud possède des attributs sous la forme de paires clef/valeur. Il est possible de connecter les noeuds entre eux avec des « arcs » qui possèdes un identifiant et peuvent également posséder des attributs (clef/valeur et de créer ainsi une structure de données arborescente. Il est alors possible de « parcourir » la base de donnée.

Par exemple (Cypher Query Language/Neo4j)

```
MATCH p=(a)-[*1..3]->(b)
WHERE a.name='Alice' AND b.name='Daniel' AND ALL (x IN nodes(p) WHERE x.age > 30)
RETURN p
```

Se traduit: parcours la base suivant le model que défini « p »,

-pour tout cas « a », à 3 niveaux (de relations), en liens vers « b », où :

« A » nommé ‘Alice’ ET « b » nommé ‘Daniel’ ET tous les noeuds où l'âge est supérieur à 30.

En français : on récupère tous les liens entre Alice et Daniel à 3 niveau de connaissance (par exemple) où les personnes sont âgées de plus de 30 ans.

Autre exemple où l’on va requêter une base pour lister l’ensemble des relations

```
// Ce qui est connecté et comment!
// Sample le graphe, retourne les patterns des labels connectés,
// avec min, max, avg degrés et les noeuds associés, leurs relations et les
propriétés.
MATCH (n)-[r]->(m)
WITH n, type(r) as via, m
RETURN labels(n) as from,
   reduce(keys = [], keys_n in collect(keys(n)) | keys + filter(k in keys_n
WHERE NOT k IN keys)) as props_from,
   via,
   labels(m) as to,
   reduce(keys = [], keys_m in collect(keys(m)) | keys + filter(k in keys_m
WHERE NOT k IN keys)) as props_to,
   count(*) as freq
```

OrientDB possède un langage de query très proche de SQL, ce qui peut être appréciable! Il est également gratuit, Neo4J offre une version gratuite mais passé certaines conditions devient payant.

Il est possible d’utiliser plusieurs langages pour interroger ces bases selon les cas: « Cypher » (par defaut Neo4J), « GREMLIN » (TinkerPop), « SQL » (avec des modifications), ...

Adapté parce que

Les bases de données graphes permettent quasiment la totalité des opérations que les bases de données relationnelles permettent. Ici, ce qui est intéressant, c’est que les problèmes de récursivité ne le sont plus.

Une base graphe va permettre un parcours en profondeur dans l'arborescence de la base très simplement.

Comparaison base relationnelle et base de graphes

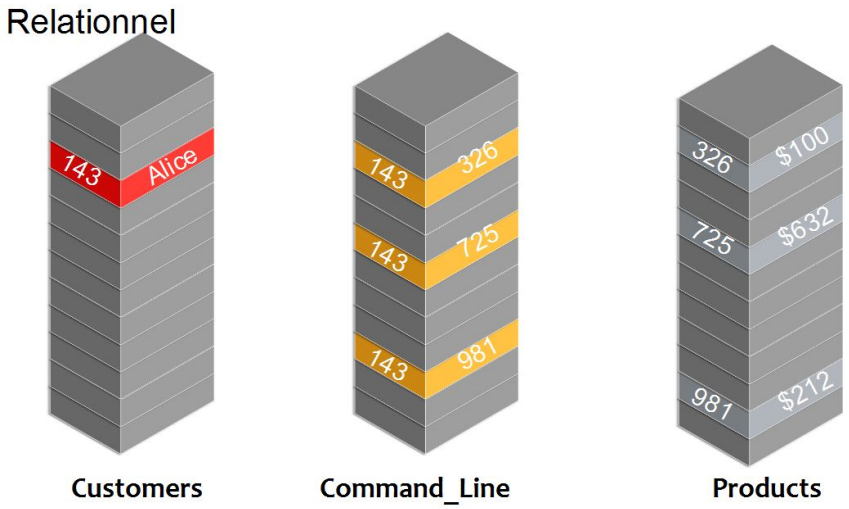
Les deux sont adaptées, mais il semble que quelques points pourraient être source de problèmes pour les règles que j'ai choisi d'appliquer lors de la conception du support technique. D'abord voyons quelques points de comparaison entre les deux.

Les cardinalités en relationnel représente une jointure d'éléments à un niveau horizontal par rapport au point d'origine de la requête dans la base.

Cela demande de passer par une jointure, il faut, dans le cas de cardinalités « *..* » par exemple, faire une requête compliquée où l'on demande les correspondances depuis une ligne d'une table via une table de jointure référençant la clé primaire de la ligne de notre table servant de point de départ de la requête ainsi que les clés primaires des éléments à joindre, les clés étrangères forment alors à elles toutes une clé primaire pour chaque lien qu'elles référencent.

Modèle physique de données en base relationnelle

Prenons un groupe de tables comme suit :



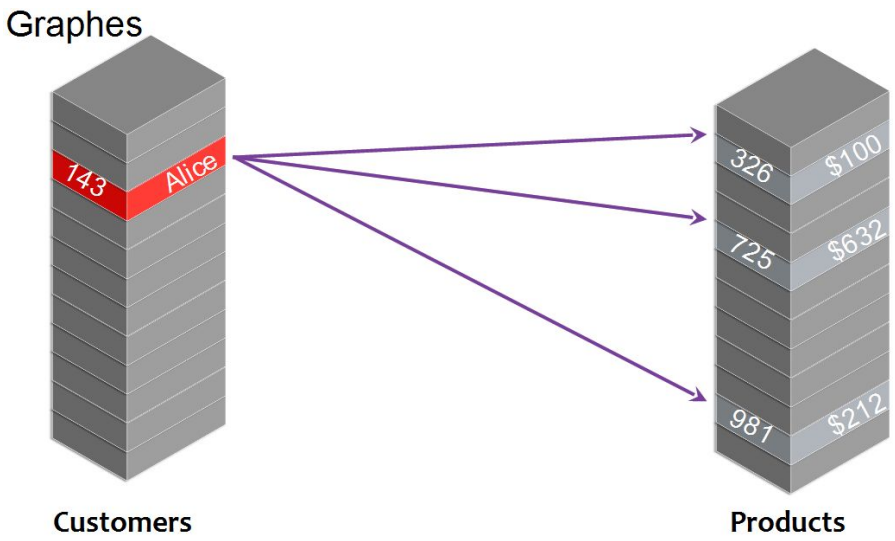
Dans cette figure, nous apercevons l'élément « Alice » dans la table « Customers » à gauche et la liste de ses achats dans la table « Products » à droite, le seul moyen de les relier est par l'intermédiaire d'une table de jointure dans notre cas. La table « Command_Line » fait donc le liens entre les deux autres.
Cette opération demande déjà des ressources au système, bien sûr les clés étrangères peuvent-être indexés ce qui accélèrera le parcours des jointures par le moteur de la base. Les cardinalités seront telles que l'on dira :

« il y a 'n' achats par utilisateurs ou pas » qui se traduit par les cardinalités « 0..n »

Modèle physique de données en base de graphes

En graphe les liaisons horizontales des jointures de tables en bases relationnelles sont déjà renseignées lors d'une requête classique, car il s'agit simplement d'un « cluster » . En effet les relations d'un noeud à un groupe de noeuds sont stockées physiquement par le moteur car s'est sa raison d'être. Une base comme Neo4j peut parcourir des millions de relations par seconde.

Lorsque nous parlons de cardinalités en graphes, il s'agit d'arborescence dans le niveau des relations. Ce que la figure en dessous va illustrer :



Ici les relations entre Alice et ses achats sont faites sans ordres de jointure, elles sont représentés physiquement dans la base et sont accessibles par la plus simple des requête, qui pourrait être comme suit :

```
MATCH (a:Person)-[:BAUGHT]->(p:Products)
WHERE a.name = "Alice"
RETURN p AS achats
```

Et cela serait extrêmement rapide.

Un grand nombre de benchmarks ont été réalisés au fil des années et les résultats sont là, le parcours de relations est plus aisé via une base graphe.

Des atouts qui ne retirent en rien les grandes qualités des SGBD-R

Tout de même cela ne retire en rien les grandes qualités de bases relationnelles, qui grâce à des règles de création de schémas tels que :

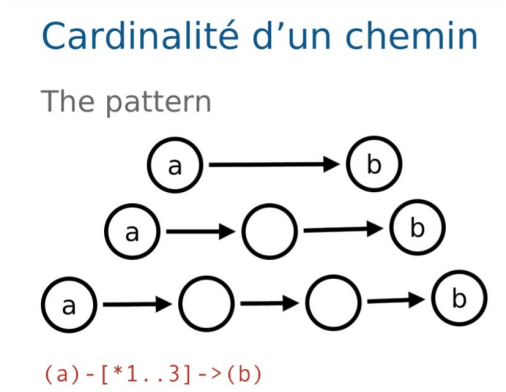
- Non-nul
- Clé indexée
- Unique

Qui permettent de garantir l'intégrité référentielle. Et assurent une forme de contrôle que le développeur n'aura pas à produire lui-même! Règles qui font que depuis leur création dans les années soixante-dix, les SGBD-R sont toujours utilisés et sont loin d'être dépassés.

Pour exemple : schéma illustrant les cardinalités en graphe

Dans le monde des graphes, les cardinalités représentent plutôt la profondeur de parcour que l'on souhaite appliquer à un pattern lors d'une requête. Pour expliquer l'illustration du dessous, et plus particulièrement lorsque l'on utilise le langage Cypher conjointement avec notre base de graphe, nous interrogeons la base via « pattern matching ».

On indique le départ de notre pattern, la relation qui lie un noeud suivant ou final, on peut omettre toutes précisions ce qui entraînera, dans la majorité des cas, le parcour de toute la base, donc à éviter! On préférera ancrer nos requêtes à un noeud précis et faire notre pattern matching depuis ce point dans la base.



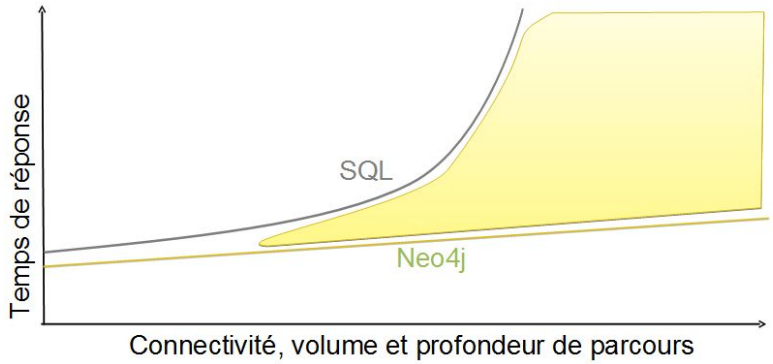
En explication, on demande à la base de parcourir tous les patterns partant d'un noeud identifié comme (a) et relié par tout type de relations à un autre noeud identifié (b).

Ces identifiants ne pointent pas vers un noeud unique, en fait ils sont anonyme et ces identifiants ne vont servir que pour initialiser des variables qui seront alors récupérées dans une instruction RETURN a, b et tout noeud unique ayant alors N relations à d'autres peut alors être transmit N fois et ce soit dans (a), soit dans (b).

Mais lors d'une seule requête une relation ne sera parcourue qu'une seule fois, et cela est une règle fondamentale du moteur sans laquelle le fonctionnement de celui-ci serait totalement erratique.

Rapport des performance temps de réponse/connectivité, volume et profondeur de parcours

Le graphique du dessous illustre le rapport Temps de réponse / Connectivité, volume et profondeur de parcours :



Comparaison de complexité d'une même requête entre le langage déclaratif Cypher et SQL

Exemple: Identifier tous les managers et compter leurs subordonnés sur 3 niveaux de profondeur :

Cypher

```
MATCH (sub)-[:REPORTS_TO*0..3]->(boss),
      (report)-[:REPORTS_TO*1..3]->(sub)
WHERE boss.name = "John Doe"
RETURN sub.name AS Subordinate,
  count(report) AS Total
```

SQL

```
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.pid AS directReportees, 0 AS count
  FROM person_reportee manager
  WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
UNION
  SELECT manager.pid AS directReportees, count(manager.directly_manages) AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT manager.pid AS directReportees, count(reportee.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT manager.pid AS directReportees, count(L2Reportees.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
UNION
SELECT reportee.pid AS directReportees, count(reportee.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION

SELECT depth1Reportees.pid AS directReportees,
count(depth2Reportees.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
  FROM(
  SELECT reportee.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT L2Reportees.pid AS directReportees, count(L2Reportees.directly_manages)
AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT L2Reportees.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
)
```

Conclusion

J'ai fait le choix d'une base de graphe parce que plus adapté pour les raisons suivantes

La simplicité d'accès à des niveaux de profondeurs différents avec le langage déclaratif Cypher étant un point très fort avec la base de graphe Neo4j, ses performances de parcours sur des grandes quantités de données très connectées font que l'accès à une chaîne récursive est simplifié avec cette technologie.

Mon maître de stage me fait remarquer que cette caractéristique sera utile pour la fonctionnalité de gestion des droits avec héritage et délégation des ingénieurs support et des GRD, réduite à son strict minimum aujourd'hui.

De plus les bases de données NOSQL se prête plutôt bien à un environnement Javascript. Mais je dois avouer que le challenge que représente l'apprentissage d'un nouveau type de persistance des données n'y a pas été pour rien !

Les technologies dans notre métier avancent tellement vite qu'il faut savoir prendre le train en marche et garder l'oeil ouvert.

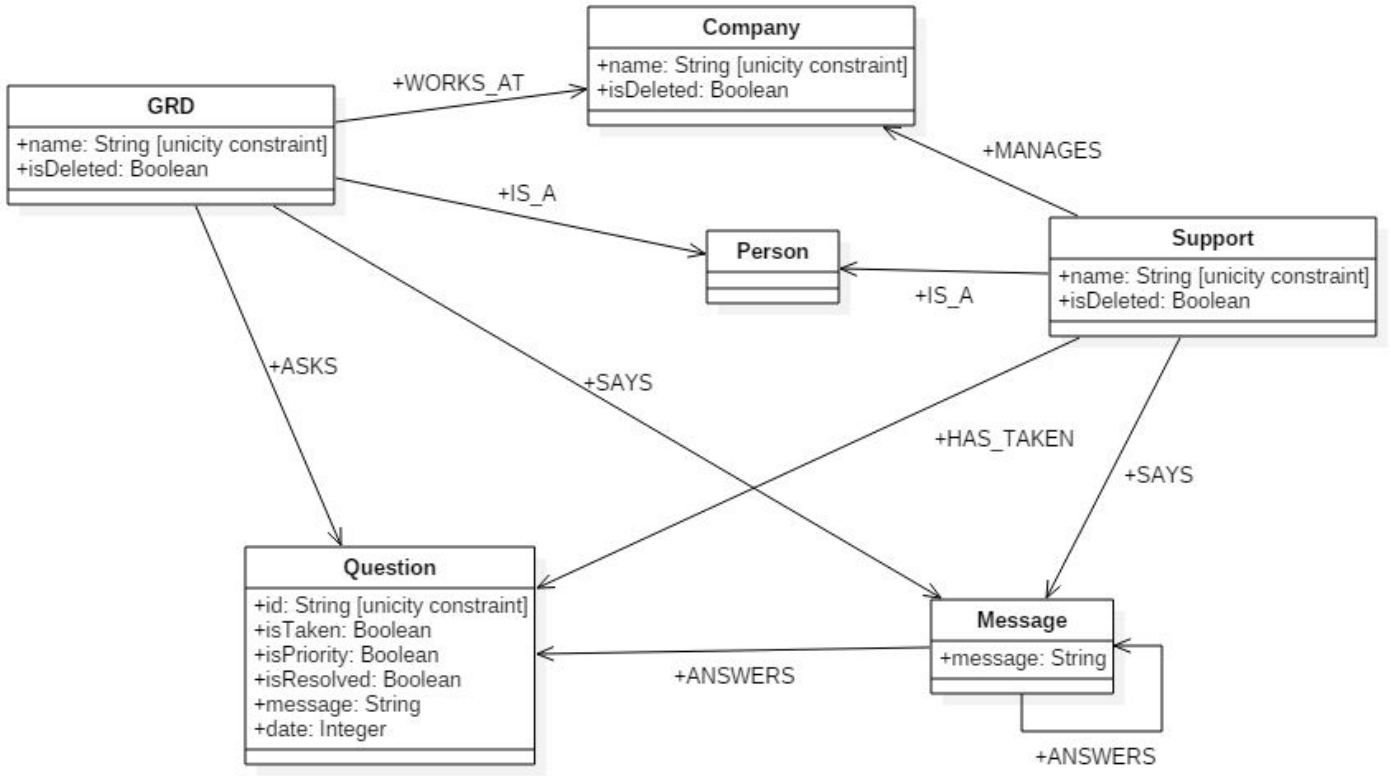
Modèle Physique de Données (MPD)

Explications

Un modèle physique de données est lié à une implémentation spécifique de base de données. En base de graphe je n'ai pas de « CREATE TABLE », le modèle physique n'est absolument pas lié à un schéma particulier et c'est au niveau implémentation logicielle que tout est défini.

Ce qui permet de pouvoir adapter mon schéma en un temps record si des modifications devaient être appliquées tant dans la phase de développement que celle de production. J'ai également été confronté à des modifications de mon schéma pendant le développement et les changement ne m'ont pris que des minutes à mettre en place !

En fait un modèle physique de données en base de graphe est identique au modèle conceptuel de données avec pour seule différence que l'on va ajouter des labels, des relations dirigées mais pas de clés étrangères ni tables de jointure.



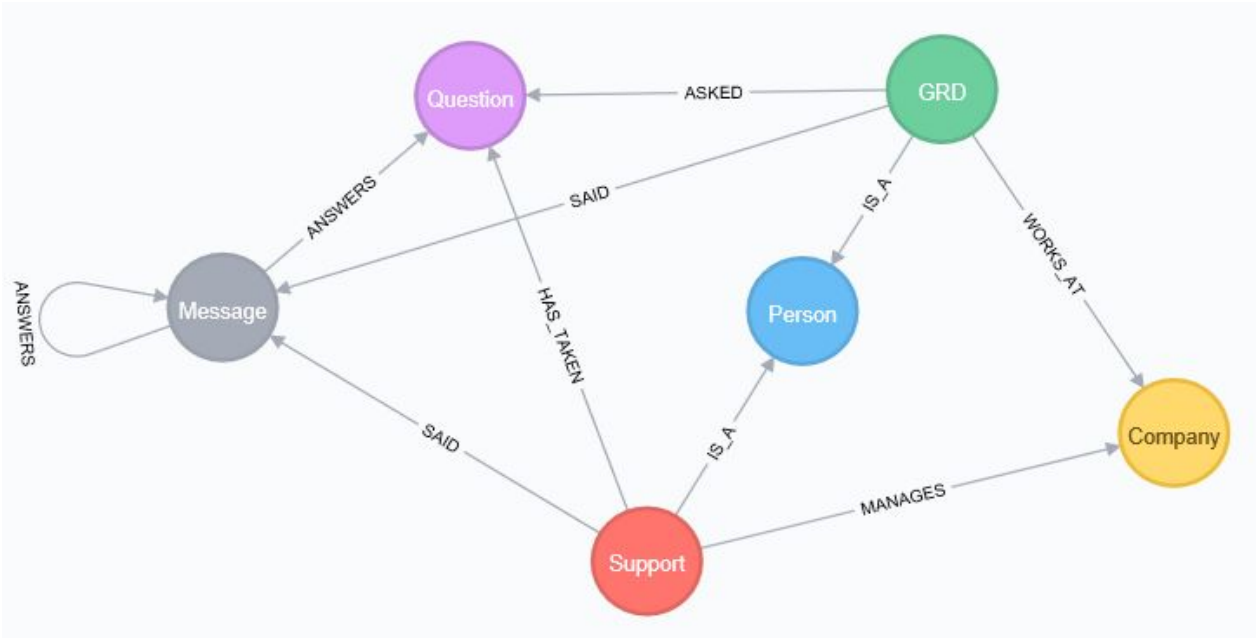
Ici chaque entité est représentée par un noeud, chaque noeud peut être groupé par ensemble qui seront défini par les labels attribués aux noeuds. Et, chaque noeud peut posséder un nombre d'attributs représentés par des ensembles clé/valeurs.

Les « jointures » entre les noeuds sont appelées des relations, chaques relation possède une direction, un noeud de départ et un noeud d'arrivée. Les relations sont nommés, en majuscules/Snake case par convention et peuvent posséder des attributs représentés par des ensembles clé/valeurs.

Le résultat tel qu'il est représenté dans la base

L'illustration ci-dessous a été obtenue grâce à une procédure stockée listant de manière unique les type de relations et les noeuds regroupés par labels, le tout alors attaché pour chaques cas physique dans la base. Elle permet un aperçu rapide du modèle.

La «**waiting room**» est **gérée de manière dynamique en javascript grâce à deux attributs du noeud «question»** qui sont «**isResolved**», «**isPriority**» , «**isPriority**» **et** «**isTaken**»



Gestion de la base de données

Création

Une instance de Neo4j = un serveur avec une Java Virtual Machine (JVM), et donc un espace de travail, il n'y a pas besoin de créer physiquement le schéma dans la base de données car il suffit de la peupler depuis l'applicatif.

Code javascript qui appelle Neo4J avec les commandes ci-dessous, elles sont présentées exécutées depuis javascript.

Exemples d'insertion de données depuis Javascript

- Création d'une entreprise RTE, par exemple

```
CypherQuery(
  "CREATE (company:Company { name:{name}, isDeleted: False }) RETURN company", {
    name: 'RTE'
  }, function (err, resp) {
    if (err) { console.log(err); } else { console.log(resp); }
  });
```

- Création d'un GRD "Jean Bon" de l'entreprise RTE

```
CypherQuery(
  "CREATE (p:GRD { name: {name}, isDeleted: false }) WITH p match (c:Company) where
c.name= {company} WITH p, c MERGE (p)-[:WORKS_AT]->(c)  RETURN p AS GRD", {
    name: 'Jean Bon',
    Company: 'RTE'
  }, function (err, resp) { if (err) { console.log(err);
    } else { console.log(resp);}
  });
```

- Création d'un ingénieur support, gérant une entreprise

```
CypherQuery(
  "CREATE (s:Support { name: {name}, isDeleted: False }) WITH s MATCH (c:Company {name:
  {company} }) MERGE (p)-[:MANAGES]->(c)  RETURN s AS Support", {
    name: 'RTE',
  }, function (err, resp) {
    if (err) { console.log(err); } else { console.log(resp); }
  });
```

- Ajout d'une question par un GRD

```
CypherQuery(
  "CALL custom.proc.uuid() YIELD uuid MATCH (p) where p.name = {name} WITH p, uuid merge
(p)-[:ASKED {date:timestamp()}]->(q:Question {id: uuid, message: {message}, resolved:
false, isTaken: false, priority: false }) RETURN q.id AS ID", {
    name: 'Jean Bon',
    message: 'Bonjour, les Bluepods ne s'affichent plus!'
  }, function (err, resp) {
    if (err) { console.log(err); } else { console.log(resp); }
  });
```

- Réponse à une question par un message

Un test doit être effectué pour savoir si aucun message ne répond déjà, si il y a un message, on chaînera les suivants au dernier message envoyé, sinon à la question ;

```
CypherQuery("MATCH (p) WHERE p.name = {who} WITH p MERGE (p)-[:SAID
{date:timestamp()}]->(a:Answer {message:{message}}) WITH p, a MATCH (q) where q.id = {id}
WITH a, q merge (a)-[:ANSWERS {date:timestamp()}]->(q) RETURN a", {
    who: "Greg Jeftic",
    message: "Bonjour, je vais me renseigner.!",
    id: '57acfacb-34f6-4e0f-94d5-c72601541667'
  }, function (err, resp) {
    if (err) {console.log(err);} else {console.log(resp)}
      });
```

- Réponse à un message par un autre message

  Un test doit être effectué pour savoir si aucun message ne répond déjà, si il y a un message, on chaînera les suivants au dernier message envoyé, sinon à la question :

```
CypherQuery(
      "MATCH (p) WHERE p.name = {who} WITH p MERGE (p)-[:SAID
date:timestamp()}]->(a:Message {message: {message} }) WITH p, a match
(m:Message)-[:ANSWERS*0..]->(q:Question) where q.id = {id} WITH a, last(collect(m)) as
last MERGE (a)-[:ANSWERS]-(last) RETURN last", {
      who: "Greg Jeftic",
      message: "Bonjour, je vais me renseigner.!",
      Id: '57acfacb-34f6-4e0f-94d5-c72601541667'
       }, function (err, resp) {
          if (err) {console.log(err);} else {console.log(resp)}
      });
```

- Fermeture d'une question

```
CypherQuery("MATCH (q) where q.id = {id} SET q.resolved=true RETURN 'Résolu et archivé'," {
id: '57acfacb-34f6-4e0f-94d5-c72601541667'
  }, function (err, resp) {
    if (err) {console.log(err);} else {console.log(resp)}
});
```

- Remise en attente d'une question qui devient alors prioritaire :

```
CypherQuery(
  "MATCH (q:Question {id: {id}, resolved: false, isTaken: {taken}, priority: {priority} })
RETURN q.id AS ID", {
    id: '57acfacb-34f6-4e0f-94d5-c72601541667'
    taken: false,
    Priority: true
  }, function (err, resp) {
    if (err) { console.log(err); } else { console.log(resp); }
  });
```

Testées indépendamment de la maquette, ces procédures seront transformée en méthodes dont les paramètres alimenteront les variables utilisées pour l'interrogation de la base et les créations ou modifications.

Elles ont été testées dans un script Javascript exécuté par Node.js, j'ai visualisé les résultats via l'interface graphique de la base, après plusieurs tests concluant, j'ai constitué les requêtes qui seront utilisées dans le module de support technique.

J'ai utilisé le driver 'philippkueng' qui est un driver Javascript pour Neo4j de bonne qualité.

Exemple de Data Transfer Object (DTO)

Ici, un objet sera créé à l'exécution d'une méthode du socket auquel le gestionnaire de réseau par exemple, se sera connecté. Pas besoin de getters et setters car l'objet sera créé de manière «littérale», et les attributs initialisés sont ceux attendus lors de l'appel de cette méthode. Mais il est possible de définir un DTO particulier auquel on passera la méthode toJSON permettant de sérialiser les attributs nécessaires.

```
var GRD = {
   toJSON: function(obj){
       return {
       name: obj.name,
       newName: obj.newName,
       company: obj.company,
       isDeleted: obj.isDeleted};
   }
};
```

L'appel de la méthode getGRDByName de l'objet d'accès aux données DAO renvoie un résultat sous forme JSON.

```
GRD = DAO.getGRDByName("Jean Bon");
```

On pourra alors le sérialiser afin de le transférer entre serveur et client.

```
JSON.stringify(GRD);
```

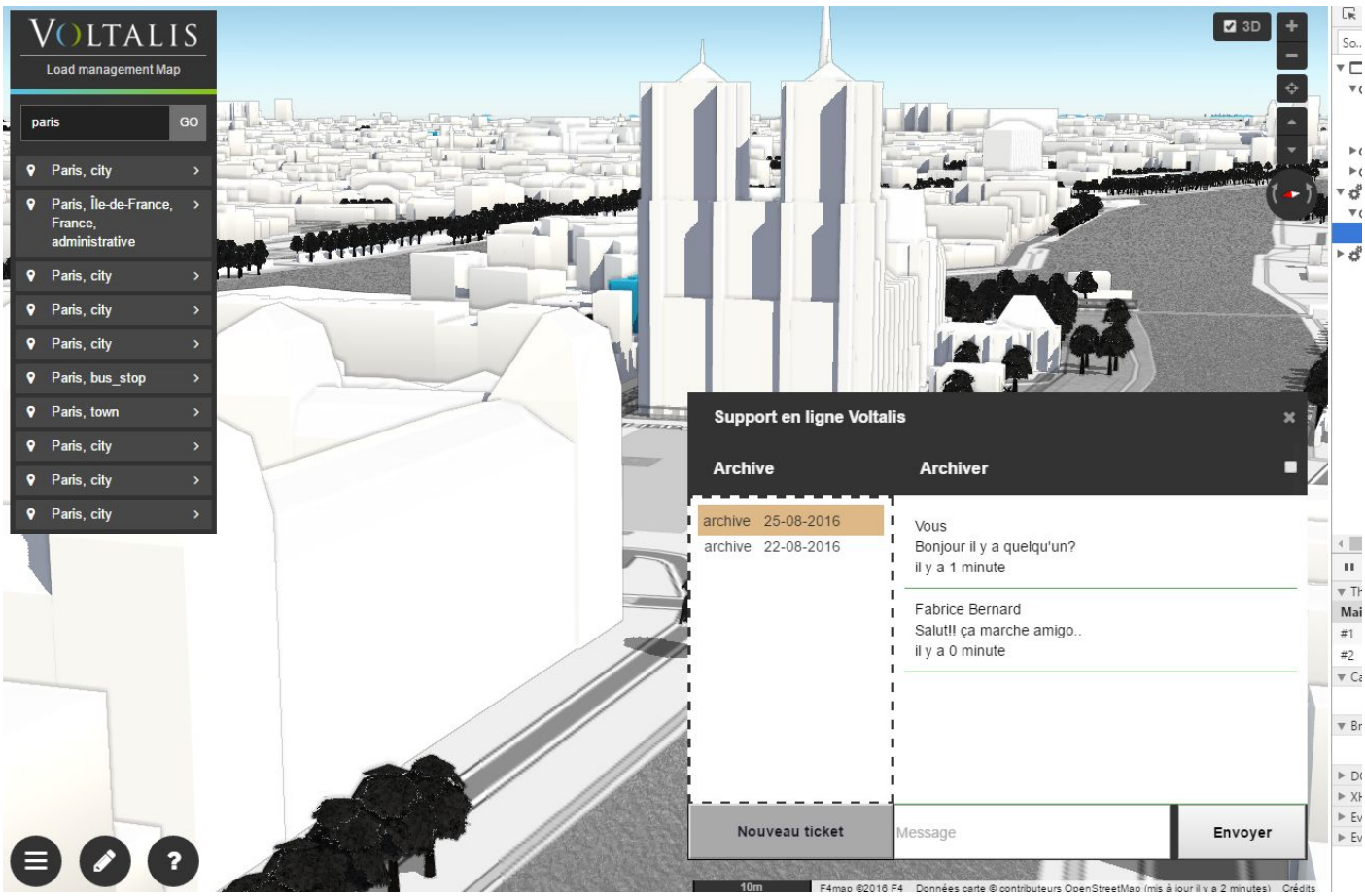
Exemple de Data Access Object (DAO)

```
// Require the Neo4J module drivers
var neo4j = require('node-neo4j');

// Create a db object. We will using this object to work on the DB.
var db = new neo4j('http://username:password@localhost:7474');
var DAO = new Object();


DAO = {
 getMessageById: function (id){
   db.cypherQuery(
   "MATCH (m)-[:ANSWERS]->(q:Question) where q.id = {questionID} return m AS message",
   {
     questionID: id
   }, function (err, result) {
     if (err) {
       return console.log(err);
     }
     return result.data; // delivers an array of query results
   });
 },


 createCompany: function (name){
   db.cypherQuery(
     "CREATE (company:Company { name: {name}, isDeleted: False }) RETURN company", {
       name: name
     }, function (err, result) {
       if (err) {
         return console.log(err);
       }
       return result.data;
     });
   }
}
```

Interface web

Interface coté GRD

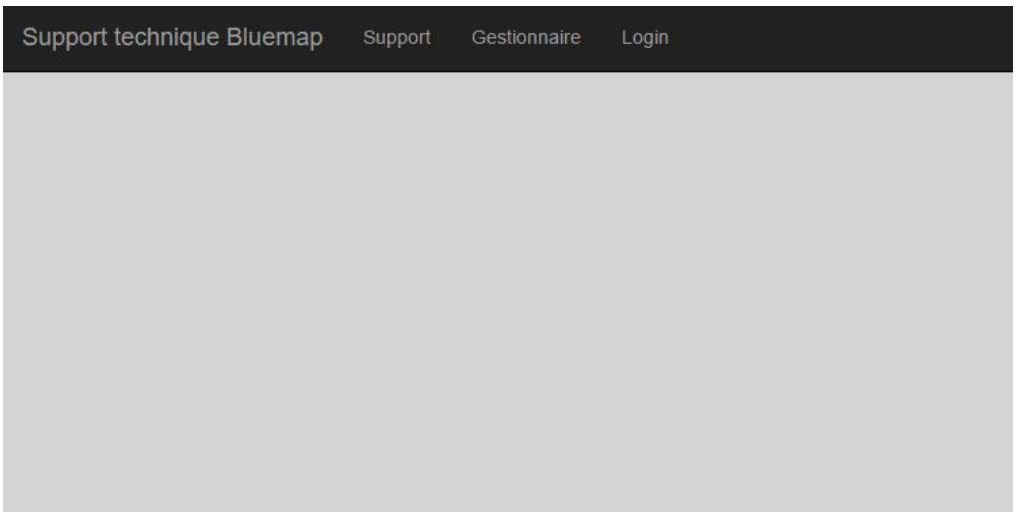
Voici les éléments inclus à l'application Blue Map côté GRD, créés avec Jade template et Stylus :



Interfaces pour la partie ingénieur support

- J'ai commencé par créer les interfaces depuis le débogueur du navigateur
- Lorsque j'obtenais un résultat satisfaisant, je récupérais les structures HTML et CSS
- Après finalisation de mes interfaces, et accord du représentant du client.
- Je les ai fait fonctionner avec des données en dur pour simulations.
- Lorsque les simulations furent abouties, j'ai inclus les contrôles dynamiques.

La page servant de point d'entrée dans l'application pour les ingénieurs support :



La demande d'authentification est faite en cliquant sur «Login» et le formulaire est généré :

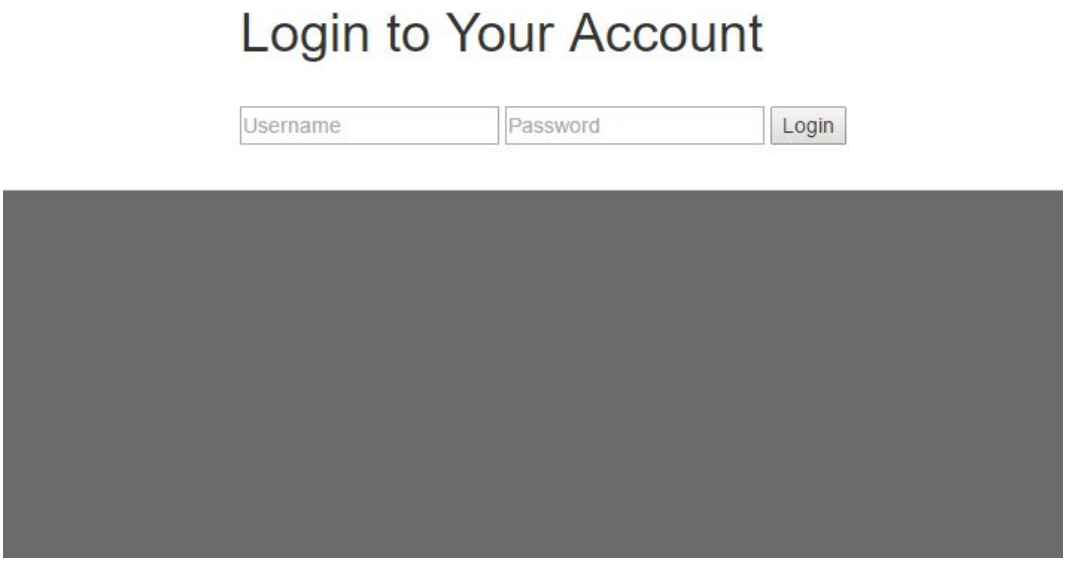
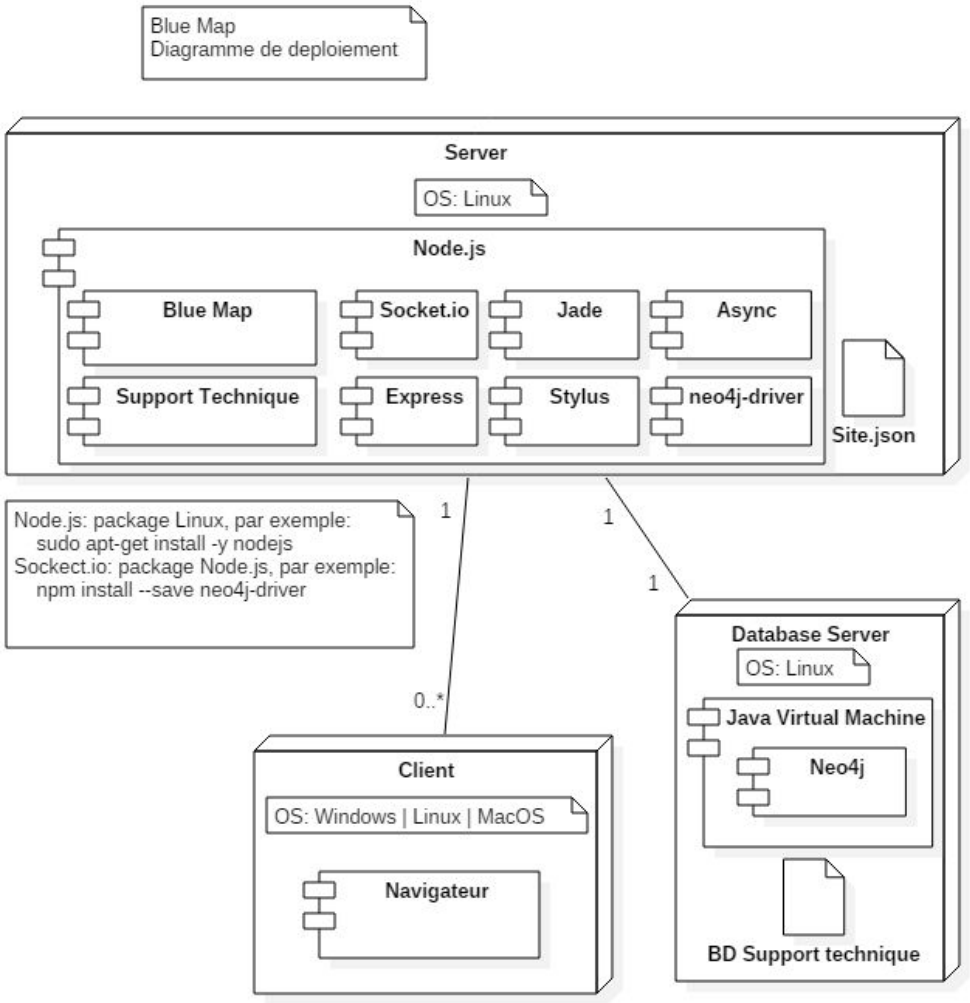


Diagramme de déploiement

Le **diagramme de déploiement** sert à représenter l'utilisation de l'**infrastructure physique par le système**, la manière dont les **composants** de ce système sont répartis ainsi que les relations qui les lient.



Les concepts utilisés dans ce diagramme sont :

→ Les **nœuds**, représentés par des polygones en trois dimensions, et symbolisants les composants matériels de l'infrastructure (par exemple : un ordinateur, un serveur, etc...)

→ Les **composants**, représentés par des rectangles avec deux «connecteurs» à gauche, symbolisent les différentes parties dynamiques du système étudié

→ Les **associations**, représentées par des lignes droites avec indicateurs de multiplicités, symbolisent les liens de communication entre les différents composants du système

→ Les **artéfacts**, dans ce contexte, représentés par une «feuille de papier», sont une manière de définir un fichier ou une base de données d'un projet.

Conclusion

Durant toute cette formation, j'ai découvert ce qu'était vraiment le métier de concepteur développeur, j'ai pu voir l'évolution d'un rêve d'enfant devenir un outils d'adulte qui fait désormais partie de moi.

Cette formation a été trop courte à mon goût et il reste tellement de choses que je ne sais pas ou que je ne maîtrise pas. Mais j'ai appris à trouver les bonnes informations et comment les interpréter.

La période de stage, également trop courte et semée d'embûches, m'a poussée dans mes limites et m'a forcé à les repousser.

Mon envie de toujours en savoir plus, de découvrir tout ce qui est possible et continuer à chercher et essayer de comprendre…

Tout ça m'a aidé à tenir le coup, quand c'était difficile, et aller en avant toute quand l'ampoule dans mon cerveau s'allumait…

J'ai découvert à quel point ce vieu rêve est devenu une passion et je vais continuer sur cette voie.

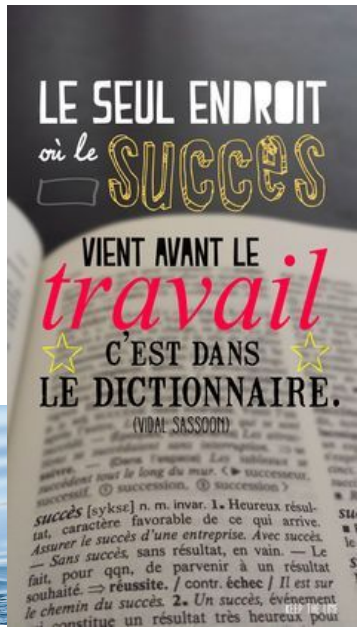
Honnêtement, je ne partais pas gagnant en découvrant le travail à fournir, j'ai tellement douté de moi! Mais j'ai fini par oublier mais doutes tout simplement et emporté par le rythme de la molette de M. Pascal Buguet, j'ai juste suivi sans plus réfléchir!

Me voilà aujourd'hui, et même si j'aurais souhaité pouvoir en faire plus et alimenter ce rapport avec tout ce que j'ai fait, je pense qu'il reflète tout de même la charge de travail que je n'ai pas fini d'accomplir et qui ne s'arrêtera plus d'empiler à mesure que je la désempile.

Je sens que je suis ressorti grandi de cette aventure, et plus mature, quelqu'un de nouveau, comme reforgé, les yeux ouvert sur une dimension plus vaste du monde. Je ne m'arrêterais plus et les ambitions que je voyais s'en aller finalement, je les aperçois de nouveau à l'horizon!

Comptez sur moi pour changer le monde tel que vous le connaissez.

Merci encore à ceux qui ont cru en moi ou qui m'ont simplement soutenus.



ANNEXES

Code de peuplement de la base

La base à un état prototype exportée via une procédure stockée, chaque groupe d'instructions sont encadrées dans des blocs de transaction afin de garantir le bon déroulement de la restauration de la base, et ne pas surcharger la Java virtual machine, ce qui entraînerait de fâcheuses et coûteuses interventions du garbage collector!

```
begin
        CREATE (:`Support`:`UNIQUE IMPORT LABEL` {`name`:"Greg Jeftic", `role`:"support", `UNIQUE IMPORT ID`:0});
        CREATE (:`Support`:`UNIQUE IMPORT LABEL` {`name`:"Ludo Peper", `role`:"support", `UNIQUE IMPORT ID`:1});
        CREATE (:`Support`:`UNIQUE IMPORT LABEL` {`name`:"Fab Beber", `role`:"support", `UNIQUE IMPORT ID`:2});
        CREATE (:`UNIQUE IMPORT LABEL` {`name`:"Mathieu Bineau", `UNIQUE IMPORT ID`:3});
        CREATE (:`Company` {`name`:"Voltalis"});
        CREATE (:`Company` {`name`:"RTE"});
        CREATE (:`GRD`:`UNIQUE IMPORT LABEL` {`name`:"Pascal Bugu", `UNIQUE IMPORT ID`:7});
        CREATE (:`Question` {`id`:"57acfacb-34f6-4e0f-94d5-c72601541667", `message`:"rien ne s'affiche.", `resolved`:false});
        CREATE (:`Message`:`UNIQUE IMPORT LABEL` {`message`:"je regarde tout de suite!", `UNIQUE IMPORT ID`:9});
        CREATE (:`Person`:`UNIQUE IMPORT LABEL` {`is`:"human", `UNIQUE IMPORT ID`:10});
        CREATE (:`Message`:`UNIQUE IMPORT LABEL` {`message`:"Merci bande de shnoufs!", `UNIQUE IMPORT ID`:11});
        CREATE (:`Message`:`UNIQUE IMPORT LABEL` {`message`:"De rien TonTon!", `UNIQUE IMPORT ID`:12});
        CREATE (:`Company` {`name`:"ERDF"});
        CREATE (:`GRD`:`UNIQUE IMPORT LABEL` {`name`:"Jean Bon", `UNIQUE IMPORT ID`:24});
commit
begin
        CREATE CONSTRAINT ON (node:`Question`) ASSERT node.`id` IS UNIQUE;
        CREATE CONSTRAINT ON (node:`Person`) ASSERT node.`name` IS UNIQUE;
        CREATE CONSTRAINT ON (node:`Company`) ASSERT node.`name` IS UNIQUE;
        CREATE CONSTRAINT ON (node:`Pool`) ASSERT node.`access` IS UNIQUE;
        CREATE CONSTRAINT ON (node:`UNIQUE IMPORT LABEL`) ASSERT node.`UNIQUE IMPORT ID` IS UNIQUE;
commit
schema await
begin
        MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:3}), (n2:`Company`{`name`:"Voltalis"}) CREATE (n1)-[:`WORKS_AT`]->(n2);
        MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:7}), (n2:`Company`{`name`:"RTE"}) CREATE (n1)-[:`WORKS_AT`]->(n2);
        MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:7}), (n2:`Question`{`id`:"57acfacb-34f6-4e0f-94d5-c72601541667"}) CREATE
        (n1)-[:`ASKED` {`date`:1475069322618}]->(n2);
        MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:0}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:9}) CREATE
        (n1)-[:`SAID` {`date`:1475070593324}]->(n2);
        MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:9}), (n2:`Question`{`id`:"57acfacb-34f6-4e0f-94d5-c72601541667"}) CREATE
        (n1)-[:`ANSWERS` {`date`:1475070593324}]->(n2);
        MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:1}), (n2:`Company`{`name`:"RTE"}) CREATE (n1)-[:`MANAGES`]->(n2);
        MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:7}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:11}) CREATE
        (n1)-[:`SAID` {`date`:1475770481223}]->(n2);
        MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:11}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:9}) CREATE
        (n1)-[:`ANSWERS`]->(n2);
        MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:0}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:12}) CREATE
        (n1)-[:`SAID` {`date`:1475770696896}]->(n2);
        MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:24}), (n2:`Company`{`name`:"ERDF"}) CREATE (n1)-[:`WORKS_AT`]->(n2);
```

```
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:0}), (n2:`Company`{`name`:"ERDF"}) CREATE (n1)-[:`MANAGES`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:1}), (n2:`Company`{`name`:"ERDF"}) CREATE (n1)-[:`MANAGES`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:2}), (n2:`Company`{`name`:"ERDF"}) CREATE (n1)-[:`MANAGES`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:0}), (n2:`Company`{`name`:"RTE"}) CREATE (n1)-[:`MANAGES`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:12}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:11}) CREATE
(n1)-[:`ANSWERS`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:7}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:10}) CREATE
(n1)-[:`IS_A`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:24}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:10}) CREATE
(n1)-[:`IS_A`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:0}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:10}) CREATE
(n1)-[:`IS_A`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:1}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:10}) CREATE
(n1)-[:`IS_A`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:2}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:10}) CREATE
(n1)-[:`IS_A`]->(n2);
commit
begin
MATCH (n:`UNIQUE IMPORT LABEL`)  WITH n LIMIT 20000 REMOVE n:`UNIQUE IMPORT LABEL` REMOVE n.`UNIQUE IMPORT ID`;
commit
begin
DROP CONSTRAINT ON (node:`UNIQUE IMPORT LABEL`) ASSERT node.`UNIQUE IMPORT ID` IS UNIQUE;
commit
```

Bibliographie

UML 2 - Modéliser une application web 4ème édition
Par Pascal Roques
Édition Eyrolles

Graph Databases 2ème Edition
by Ian Robinson, Jim Webber, and Emil Eifrem
Published by O'Reilly Media

Don't make me think 2ème Edition
by Steve Krug
Published by New Riders

Pro Javascript Design Patterns
by Ross Harmes and Dustin Diaz
Published by Apress

Webographie

http://www.w3schools.com/

http://stackoverflow.com/

http://www.supinfo.com/

http://www.adultswim.com/etcetera/soup/

https://developer.mozilla.org/fr/

http://underscorejs.org/#

https://hyperdev.com/

https://scotch.io/

http://socket.io/docs/

http://fontawesome.io/icons/

http://caolan.github.io/async/

https://neo4j.com/developer/

https://lodash.com/

http://csv.adaltas.com/transform/

http://www.postgis.fr/wiki/Documentations

https://neo4j-users.slack.com/messages/help/

Définitions

Effacement diffus

L'**« effacement diffus »** ou *« lissage de la courbe de charge par le pilotage de la demande »* consiste, en cas de déséquilibre offre/demande d'électricité à provisoirement (en période de pointe journalière et/ou saisonnière de consommation électrique) réduire la consommation physique d'un site donné ou d'un groupe d'acteurs (par rapport à sa consommation « normale »), l'effacement étant déclenché par une stimulation extérieure.

L'effacement peut constituer un substitut économique à l'installation de nouveaux moyens de production et une réponse à la difficulté de stocker de l'énergie pour l'utiliser plus tard, notamment afin de contribuer à l'équilibrage du réseau lors d'une baisse de production, d'une hausse de consommation, ou de compenser l'intermittence de la production à partir d'énergies renouvelables (solaire, éolien, etc.), sans sacrifier le confort des ménages et la production des entreprises.