Rapport de stage Damien Mostacchi

Rapport de stage au sein du laboratoire i3S, dans l'équipe Modalis.

Damien Mostacchi

IUT Nice-côte d'azur
Tuteur IUT : M. Léo DONATI
Tuteur entreprise : Mme Mireille BLAY-FORNARINO
18/06/2012

Remerciements

Je tiens à remercier ma tutrice en entreprise, Mme Mireille BLAY-FORNARINO pour son soutien et ses conseils tout au long de mon stage.

Je tiens à remercier également M. Simon URLI qui a su m'encadrer et m'orienter, ses conseils furent précieux et instructifs.

Je remercie aussi tous les membres du laboratoire i3S, qui ont su rendre mon séjour agréable et convivial.

Enfin, je remercie M. Léo DONATI pour ses conseils et pour l'intérêt qu'il a porté au bon déroulement de mon stage.

Résumé

Lors du Diplôme Universitaire de Technologie Informatique, il est obligatoire de réaliser un stage en entreprise d'une durée minimum de 11 semaines, permettant ainsi de valider le diplôme, mais aussi d'acquérir une première expérience professionnelle.

Mon stage a été effectué au laboratoire i3S à Sophia Antipolis, et plus précisément au sein de l'équipe Modalis. Mon stage s'est déroulé du 16 avril au 29 juin soit exactement 11 semaines.

Le projet dans lequel s'inscrit ce stage s'appelle YourCast. Ma part du travail se concentre essentiellement sur la partie interface graphique. Un des buts de YourCast est la diffusion d'informations dans le cadre de grands rassemblements, mais aussi dans le milieu académique, permettant donc de diffuser en continu des informations en rapport avec l'évènement auquel il est associé.

Les difficultés majeures résidaient d'une part dans le fait de produire un code clair et générique et d'autre part de résoudre des problèmes de compatibilité inter-navigateurs.

À la fin du stage, ma partie du projet est totalement implémentée, l'interface est maintenant fonctionnelle et testée.

Summary

During the DUT in computer science, it is mandatory to perform an internship for a minimum duration of 11 weeks, its serves to validate the diploma, but also to gain valuable work experience.

My internship was done at the i3S laboratory, at Sophia Antipolis, specifically in the MODALIS team. My internship took place from April 16 to June 29, exactly 11 weeks.

The project on which I worked is called YourCast. My work focuses on the GUI part. One goal of YourCast is to display informations through mass rallies, but also in academia Institute, thus allowing streaming informations related to the event which it is associated.

The major difficulties resided on the one hand the fact of producing a clear and generic code and on the other hand to solve problems of cross-browser compatibility.

At the end of the internship, my part of the project is fully implemented, the interface is now fully functional too and tested.

Table des matières

Glossaire

- Behaviors : les behaviors, comportement en français, définissent les animations utilisées, le comportement de chaque zone de l'interface. Par exemple, en fondu, de gauche à droite etc.

- Contrôleur : le contrôleur permet de faire le lien entre tous les éléments d'une zone, il la relie donc au bon behavior et aux renderers.

- JavaScript : Langage de programmation de scripts principalement utilisé dans les pages web interactives mais aussi côté serveur.

- JQuery : Bibliothèque JavaScript libre portant sur l'interaction entre JavaScript et HTML, et a pour but de simplifier des commandes JavaScript

- JSON : Sérialisation JavaScript, voir Annexe 3

- Layout : un layout est la partie graphique de l'interface et définit donc le style de la zone auquel il est relié.

- Providers : Processus informatique de récupération des informations fournies par les sources d'informations. Un provider produit des listes d'informations.

- Renderers : processus informatique de transformation d'un format JSON en un format html. Un renderer est adapté à un type d'information et à un type de zone.

- Sources : ce sont les différentes sources d'informations reçues, par exemple, des calendriers, des news, des informations internes etc.

- Zone : Partie logicielle qui supporte la diffusion de certaines informations. Un écran est composé de zones de diffusion. une zone possède un behavior, un layout et les renderers de chaque source.

I. Présentation du Laboratoire

a. Le laboratoire i3S

Le laboratoire i3S est une Unité Mixte de Recherche (UMR) entre l'Université de Nice-Sophia Antipolis (UNS) et le Centre National de la Recherche Scientifique (CNRS).
Le laboratoire est organisé en 4 pôles scientifiques regroupant 12 équipes de recherche.
Les 4 pôles sont :

- COMRED / Communications, Réseaux Systèmes Embarqués et Distribués
- GLC / Génie du Logiciel et de la Connaissance
- MDSC / Modèles Discrets pour les Systèmes Complexes
- SIS / Signal, Image, Systèmes

Il est composé d'enseignants-chercheurs de l'UNSA (71), 18 chercheurs du CNRS et 11 chercheurs INRIA.

Le pôle GLC au sein duquel j'ai fait mon stage regroupe 4 équipes :

- L'équipe KEIA (Knowledge Extraction, Integration and Algorithms)
- L'équipe WIMMICS (Web-Instrumented Man-Machine Interaction, Communities and Semantics)
- L'équipe RAINBOW (Reflexive Architecture for Interaction Network Between Objects Working together)
- L'équipe MODALIS (MODels to usAge of large scaLe InfraStructures)

b. L'équipe MODALIS

Mon stage s'est déroulé au sein du pole Génie du Logiciel et de la Connaissance (GLC) et plus précisément dans l'équipe MODALIS.

MODALIS a pour but la modélisation et l'exploitation d'infrastructures distribuées comme les grilles. Dans ce contexte, l'équipe a une forte activité autour de la métamodélisation en général et plus spécifiquement des lignes de produits pour gérer la grande complexité des logiciels produits. Les travaux de recherche portent également sur la réalisation d'architecture de services souples prenant en compte les besoins utilisateurs (déploiement distribué, qualité de service).
Afin d'optimiser les performances, l'équipe développe aussi des modèles de systèmes distribués complexes et évolutifs. Les résultats de recherche sont des outils aidant les utilisateurs à gérer des infrastructures largement distribuées pour les besoins de leurs applications. En particulier, MODALIS possède une grande expérience dans le domaine des applications distribuées d'analyses d'images médicales.

c. Les autres acteurs

Le projet YourCast regroupe aussi d'autres acteurs, autres que l'équipe MODALIS.
Ce projet regroupe l'équipe RAINBOW, aussi au laboratoire i3S à Sophia Antipolis, mais aussi, l'équipe ADAM à l'université de Lille.
Le projet vise notamment un déploiement par la société SUPRALOG qui équipe de grandes associations (fédérations sportives, mouvements de jeunesse) d'une solution de systèmes d'informations propriétaires Intr@ssoc.
Ces associations organisent des manifestations rassemblant jusqu'à plusieurs dizaines de milliers de participants : compétitions, concerts, rassemblements festifs, etc. Au cours d'un tel événement, diffuser en flux continu des informations de natures très diverses (programme d'activités horodatées et géo localisées, retransmission photos/vidéos en direct/différé, « brèves », etc.), sur différents supports (tableaux d'affichage, téléphones, etc.) et en différents lieux (zones d'activités, de passage, d'accueil) est un besoin critique pour orienter et canaliser le flot de participants.
Une partie des informations provient alors du SI lui-même (source propriétaire) et l'autre de sources « publiques » accessibles sur Internet.

II. Présentation du projet

a. Présentation de YourCast

YourCast est un projet ANR EMERGENCE, c'est à dire un projet de recherche destiné à des débouchés industriels et qui vise à une mise en production fin 2013. Il cible la production de systèmes de diffusion d'informations par génération de code.

À terme, YourCast doit permettre à un utilisateur (non informaticien) de sélectionner différents éléments constituant un système de diffusion d'informations pour ensuite supporter la génération automatique du système prêt à déployer sous la forme d'une archive.

YourCast intégrera entre autre un catalogue d'éléments permettant de construire un système de diffusion d'informations. Il sera composé d'éléments «libres» offerts par la ligne de produit YourCast et de modules spécialisés Intr@ssoc. Le moteur de génération des supports de diffusion lui-même reste libre.

Un système de diffusion d'information comprend les éléments d'architecture présentés en Figure 1. Durant le stage, je dois donc m'occuper de la partie cliente du projet, qui concerne plus spécifiquement la partie basse du diagramme.

Le travail à effectuer est de récupérer les données du provider qui sont sous le format JSON, JavaScript Object Notation, de créer les renderers associés, de créer des behaviors et de tout mettre en place grâce au layout et le style des zones.

Mon stage cible entre autre la production des codes permettant de visualiser les informations pendant les journées de conférences du GDR GPL qui regroupent tous les ans plus d'une centaine de participants.

Les composants que je définis aujourd'hui seront soit des éléments du catalogue, soit générés.

b. Architecture du projet

Un diffuseur d'informations YourCast diffuse des informations en continu, ces informations sont issues de différentes sources. Ces sources peuvent être internes, c'est le cas pour des annonces ou des menus de déjeuner par exemple, ou bien externes comme des tweets ou des albums photos Picasa par exemple.
Ces sources sont ensuite récupérées par un provider et condensées en un fichier JSON.

À chaque source, est associé un renderer qui lit le fichier créé. Un renderer définit l'affichage des informations, sa forme et son temps d'affichage.

L'affichage est alors décomposé en plusieurs zones, Chaque zone est associée à un provider, elle récupère alors son propre JSON et les renderers sont associés aux sources au sein d'une même zone. Chaque zone possède ses propres caractéristiques, elle comprend un comportement (behavior) et un style (layout).

Le style de la zone définit l'affichage des différentes informations générées par les renderers, mais il décrit aussi l'aspect général de cette zone, les ombres, le fond etc.

Le behavior définit lui, le comportement des informations à l'écran, si les informations clignotent, vont de droite à gauche etc.

Une zone ne possède qu'un seul comportement et un style associé. À ce style vient se rajouter le layout général de l'écran de diffusion, qui lui définit le design complet, l'emplacement des différentes zones, la couleur générale, les formes etc.

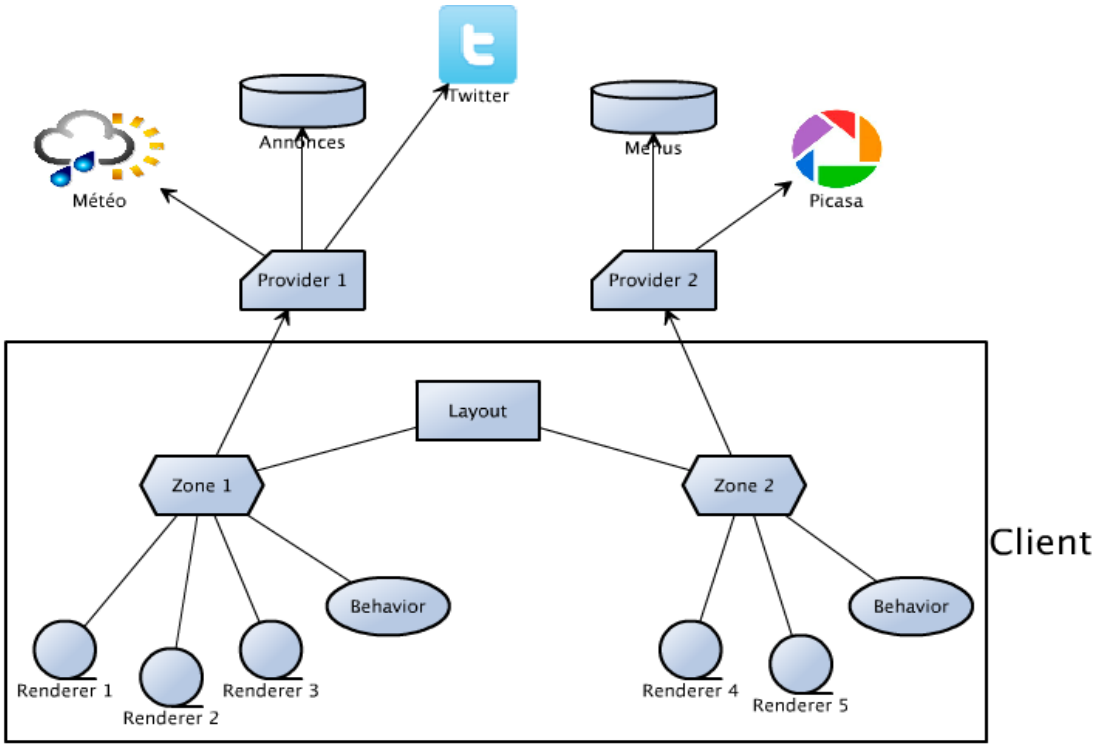


Figure 1 - Architecture du projet YourCast

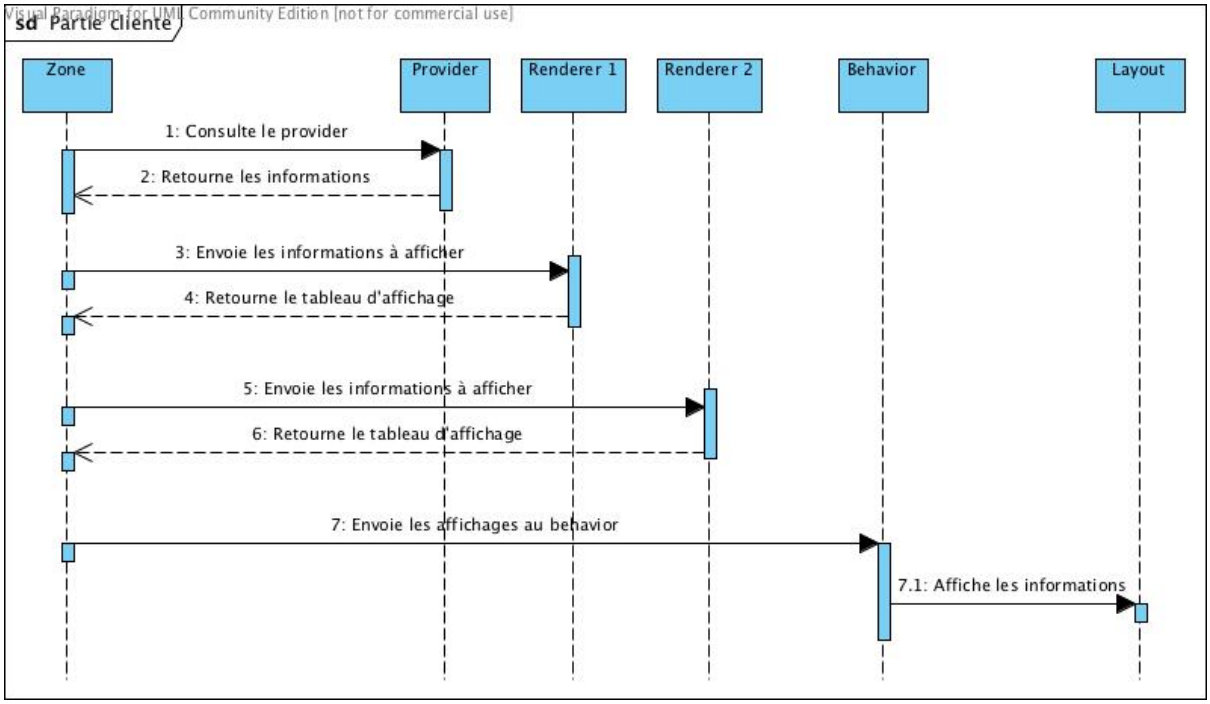


Figure 2 Diagramme de séquence partie client

c. Mon rôle dans le projet

Mon rôle consiste à réaliser la partie cliente du projet. Elle comprend la réalisation des renderers, des behaviors, des différentes zones et du layout général.
Pour le développement de l'affichage, mon principal objectif était qu'il soit adapté pour être utilisé lors des journées du GDR – GPL, les développements présentés dans cette partie sont donc en rapport avec cet objectif, le nombre de zones, les renderers etc. Le développement a été effectué de manière agile et dans une approche générique permettant plus tard une réutilisation automatique dans un but de génération.

Le layout est décomposé en deux zones, une zone principale, au centre de l'écran pour afficher des informations nécessitant de la visibilité, et une deuxième zone en bas de l'écran pour, dans mon cas, faire afficher des tweets en rapport avec les journées du GDR - GPL. Mise à part les logos, l'interface que j'ai développée ne comporte aucune image, tous les fonds, les différents affichages utilisent la propriété background de css3. On peut donc avoir des jeux de couleurs, des dégradés sans pour autant avoir d'image. Cela permet un chargement plus rapide et moins d'éléments à embarquer dans le projet qui je le rappelle, devra être totalement généré et livré sous forme d'archive. Le poids de cette archive doit donc être limité, ne pas y stocker d'images est déjà un plus dans ce sens.

Une zone possède un contrôleur, qui lui permet de définir quelles informations elle va afficher, quel provider elle utilise, quelle source avec quel renderer et d'autres informations utiles sur cette dernière. Il permet de faire le lien entre le provider, les renderers et l'affichage.

Les renderers permettent de récupérer les différentes informations du provider et les afficher sur la zone. J'ai donc développé plusieurs renderers, au moins un pour chaque source, mais dans certaines sources, j'ai dû en développer plusieurs pour avoir des affichages différents qui pourront ensuite être sélectionnés par l'utilisateur et ainsi être configuré à son goût. Prenons l'exemple de la source calendrier, il y a une dizaine de renderers différents, un qui affiche chaque évènement un par un avec le détail de chacun, un en vue synthétique, un sous forme d'emploi du temps etc. Pour cela voir l'annexe 1, Renderers Calendrier.

J'ai aussi dû développer le layout général de l'écran de diffusion, définir l'emplacement des zones, mais aussi leur forme, ou encore la couleur de l'écran, l'emplacement de l'heure etc. Encore une fois, ce layout sera ensuite généré au gré de l'utilisateur à l'aide du configurateur. Il y a aussi un style pour chaque zone, ce style regroupe tous le design des renderers mais aussi les caractéristiques propres à la zone elle-même, son fond, la disposition des éléments à l'intérieur, le placement du logo et du titre.

Viennent ensuite les behaviors ou comportements en français.
Les behaviors permettent de définir la manière dont les informations vont s'afficher dans une zone, c'est à dire si les informations s'affichent simplement une par une, ou arrive de bas en haut, de gauche à droite etc. Les animations utilisées par les behaviors sont faites en CSS3, elles ont donc l'avantage d'être très légères.

Chaque zone ne possède qu'un seul behavior, par exemple la seconde zone, utilise un behavior permettant de faire défiler les informations de gauche à droite ou de bas en haut selon le choix de l'utilisateur. Alors que la zone 1 veut un simple affichage par exemple.
Il y a, comme pour les renderers, plusieurs behaviors, tout ça permettra donc ensuite à l'utilisateur final de configurer lui même son affichage en fonction de ses besoins et de ses envies.
Durant ma période de stage, il était aussi prévu de créer de nouveaux layouts, notamment celui pour l'institut IRSAM qui est l'Institut Régional des Sourds et Aveugles de Marseille. Ce layout est bien sûr totalement différent de celui des journées du GDR – GPL, mais l'objectif était bien entendu de réutiliser les renderers et behaviors déjà créés.

J'ai dû aussi durant mon stage, développer deux interfaces d'administration, pour les annonces internes et les menus (repas). Ces développements n'étais pas prévus au début du stage mais ce sont imposés par eux même quand il a fallu se servir des annonces et des menus.

d. Démarche imposée

Le but de YourCast est le déploiement automatique d'un écran de diffusion, les codes seront donc ensuite générés automatiquement et formeront une archive permettant d'être déployée automatiquement sur le poste du client. L'utilisateur n'aura plus qu'à configurer l'affichage et l'écran sera en place.
La démarche imposée était incrémentale et agile, j'ai donc développé bout par bout en rajoutant des fonctionnalités petit à petit. Tout ça en prenant en compte les différents changements que l'on a dû opérer tout au long du projet.
Les changements ont été importants, notamment sur l'architecture générale. Beaucoup de choses n'étaient pas prévues au début de mon stage, j'ai donc dû gérer tout ça, et tout ce que cela entraine derrière, notamment les modifications à opérer. Ce n'était pas un projet où tout était écrit à l'avance et où il suffisait de faire le point toutes les deux semaines : il s'agit d'un projet de recherche sur lequel nous avons travaillé en interaction avec les clients potentiels et dont les besoins changeaient d'une semaine sur l'autre.

La démarche prenait aussi en compte une forte interaction avec le client potentiel, nous jouions donc le rôle de client pour les journées du GDR – GPL, permettant de bien définir les attentes de l'utilisateur par rapport à YourCast. Il y a eu, au cours du stage plusieurs démos, permettant de bien définir ce que l'utilisateur attend de ce projet.
Durant mon stage j'ai pu voir plus précisément ce qu'était la gestion de projet, j'ai dû utiliser une forge, permettant de garder une trace écrite de mes développements et de pouvoir saisir mes temps. Cette forge permet donc à l'équipe de savoir combien de temps il faut pour faire une chose ou une autre.
Il a donc fallu expliciter les problèmes et les besoins, puis créer une tâche sur la forge. Viens ensuite le temps de réaliser cette tâche, de réaliser les tests associés, puis la coordination

avec le client potentiel pour vérifier que le résultat est correct, sinon faire les modifications en conséquence. Enfin, quand tout est correct, effectuer la fermeture de la tâche dans la forge en spécifiant le temps passé.
J'ai dû aussi apprendre à utiliser le versionning, avec l'utilisation de Git, me permettant de soumettre mon travail et ainsi pouvoir garder plusieurs versions.
La principale démarche qui m'ait été imposée, est le fait de rendre un code le plus clair et générique possible en utilisant aussi des technologies souples et légères.

e. Calendrier prévisionnel

Voici le calendrier prévisionnel comme il apparaît dans le cahier des charges.

- Phase d'analyse (1semaine) :
  - Analyse du projet

    Le projet YourCast est déjà en cours de développement, je dois donc me familiariser avec l'architecture du projet et ses différentes articulations
  - Analyse de JSeduite

    Un projet se rapprochant de YourCast avait été réalisé par l'équipe, YourCast est une sorte de JSeduite largement amélioré. L'ancien projet est lourd et n'est pas adapté au but final de YourCast, la génération automatique des codes.
  - Analyse des technologies

    Les technologies sont déjà établies, mais il va falloir analyser le besoin, ce qui est utile ou ce qui ne l'est pas, l'utilisation d'outils spécialisés ou autre.

- Phase de développement (9 semaines)
  - Phase de formation

    Durant toute la durée de la phase de développement, je prévois en parallèle une phase de formation. Les technologies utilisées ne sont pas celle que j'utilise tous les jours, je prévois donc de me former en même temps que la phase de développement, cela me permettra de me confronter au problème directement et de me former en tentant de le résoudre.
  - Développement du layout principal

    Il y aura d'abord une phase de développement du layout principal, qui servira à définir l'emplacement des différentes zones ou d'éléments externes comme la date par exemple.
  - Développement des renderers

    Il y aura donc ensuite le développement des renderers. Durant cette phase, il faudra donc récupérer et afficher les différents éléments présents dans le fichier

JSon fournit par le provider. Il y aura autant de renderers que de types d'informations transmises.

- Développement des styles de zones

  Chaque zone aura son propre style, il faudra donc mettre en place tous les éléments amenés par les renderers, les différents emplacements, mais aussi le design de la zone.

- Développement des behaviors

  Il y aura ensuite, la phase de développement des behaviors. Chaque zone possède un behavior, mais il va falloir en créer plusieurs pour permettre plus tard le choix par un manager du behavior à utiliser.

- Phase de test (10 semaines)
  - Durée des tests

    Les tests s'étaleront sur toute la durée de développement permettant ainsi de tester au fur et à mesure.
  - Analyse des outils de tests

    Pour les outils, il faudra donc analyser ceux disponibles, pour en citer quelques uns, QUnit, JSUnit, etc. Ces outils permettent de mettre en place des tests unitaires sur les fichiers JavaScript.
  - Tests des renderers et behaviors

  Il s'agira donc de tester les renderers et les behaviors, si ils récupèrent et affichent correctement toutes les informations transmises par le provider et si tout se passe correctement durant l'animation

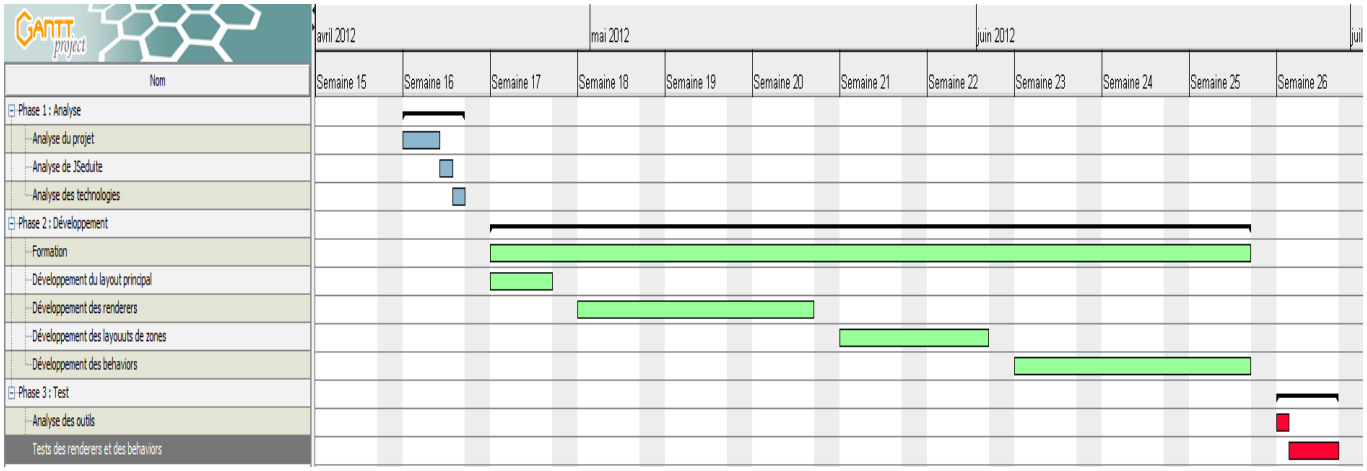


Figure 3 - Diagramme de GANTT prévisionnel

III. Mes contributions aux projets

a. Étude des solutions existantes

Un projet du même type que YourCast avait été développé par l'équipe auparavant. Ce projet, JSeduite, équipe pour l'instant Polytech Nice Sophia-Antipolis mais aussi d'autres organismes, notamment deux instituts, l'IRSAM et Clément Ader. Le problème de JSeduite, c'est qu'il ne correspond pas aux attentes de l'équipe, les codes ne sont pas clairs et loin d'être génériques, le projet n'est pas maintenable auprès des organismes équipés. Pour faire l'interface de YourCast, on m'a demandé de m'inspirer de JSeduite, notamment pour les layouts et les renderers, j'ai donc récupéré les couleurs et les dispositions.

Voir la Figure 3 qui est l'interface préalable de JSeduite et la Figure 4 qui est donc l'interface que j'ai développé pour YourCast.

Ces deux figures sont basées sur les renderers calendriers respectifs, on voit donc bien l'inspiration des couleurs, mais l'interface a totalement été refaite, JSeduite ne gérait pas par exemple, tous les débordements de contenus, les éléments n'étaient pas très bien placés et l'architecture du projet n'était pas claire et encore une fois, loin d'être générique. Il a donc fallu tout repenser et développer une nouvelle version bien en accord avec les conditions imposées pour le projet YourCast.

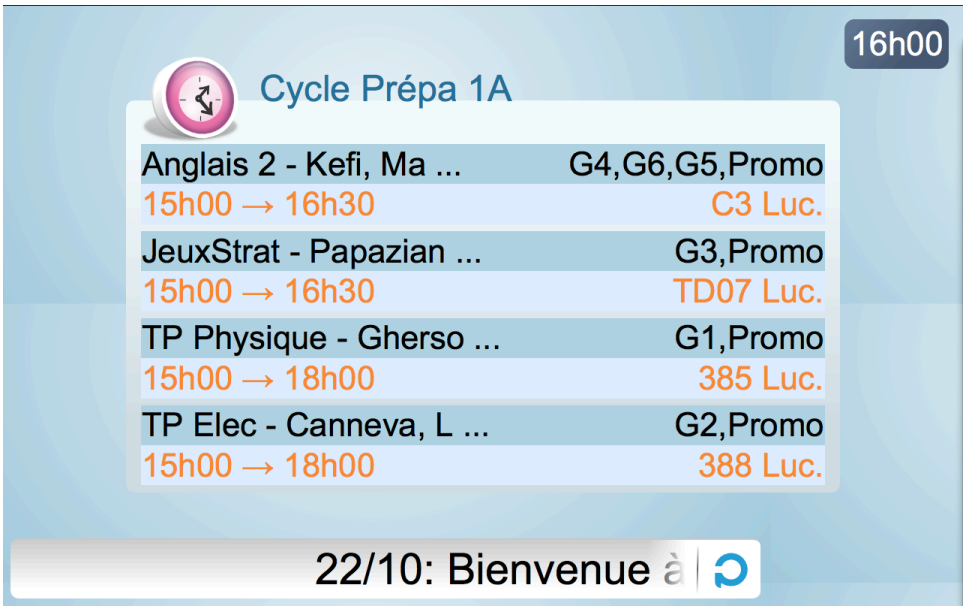


Figure 4 - Calendrier sur JSeduite

Figure 5 - Calendrier sur YourCast

b. Technologies utilisées

Les technologies utilisées sont de type Web, HTML5/CSS3 et JavaScript. Les langages web ont la particularité d'être assez souples et très légers, ce qui était le meilleur choix pour un écran de diffusion.

L'utilisation de HTML5 et css3 a impliqué de poser une restriction sur les navigateurs. L'HTML5 n'étant pas encore considéré comme un standard, tous les navigateurs ne le supportent pas encore ou ne le supportaient pas dans les versions antérieures. C'est le cas par exemple de internet explorer développé par Microsoft, qui ne supportera l'HTML5 que dans sa version 10 qui arrivera en octobre lors de la sortie de Windows 8, son nouveau système d'exploitation. Il m'a donc été imposé de ne travailler qu'avec les navigateurs les plus modernes, soit Firefox, Chrome et Safari.

Ayant déjà un peu pratiqué le JavaScript, l'intégration pour moi fût plus rapide, le plus dur a été de bien comprendre les mécanismes de YourCast et d'adapter mon développement à cela. Par contre, je ne connaissais pas vraiment l'HTML5 et le css3, j'ai donc du apprendre les différents concepts et les nouvelles normes amenées par ces deux langages. J'ai donc suivi une phase d'apprentissage, que j'ai effectuée tout au long du développement me permettant ainsi de me former en me confrontant au problème.

Pour les différents styles et le layout général, je n'ai pas utilisé du css mais un dérivé, le LESS. Le langage LESS est du css mais dynamique, on peut utiliser des variables, définir des fonctions, ou des opérations. Pour donner un exemple, on peut définir une variable « couleurDeFond » que l'on pourra alors réutiliser dans toutes nos feuilles de style. Mise à part, le langage LESS utilise la syntaxe css mais rajoute certaines améliorations qui permettent de rendre le code plus clair et plus organisé, mais bien entendu une feuille css est reconnue en LESS.

Un autre point, que je ne connaissais pas du tout, est le framework Prototype. Prototype permet donc d'apporter de nouvelles méthodes et propriétés au langage JavaScript. Nous avons pu, à l'aide de cette librairie, découper le contrôleur de zone sous forme de classe et utiliser les notions d'héritage pour implémenter ces classes. Prototype permet aussi de simplifier toutes les requêtes Ajax et apporte des méthodes, de trie par exemple, qui nous ont été utiles.

c. Les renderers

Les renderers sont développés en JavaScript, on peut en compter quasiment une trentaine, par exemple, des renderers pour la météo, pour les calendriers, pour les annonces etc. Un renderer reçoit un fichier de type JSON, qui est un dictionnaire JavaScript avec des couples nom/valeur, contenant toutes les informations à afficher. Un renderer est donc associé par le contrôleur à une source et la découpe pour ainsi créer l'affichage et renvoie alors à la zone un tableau, contenant l'affichage qu'il a créé. Un renderer définit donc cet affichage, mais il définit aussi le titre à afficher sur la zone, le logo et le temps nécessaire pour afficher l'information (qui sera ensuite utilisé par le behavior pour afficher l'information). Durant mon stage j'ai donc développé différents renderers qui pourront ensuite être sélectionnés par l'utilisateur lui même.

d. Les behaviors

Les behaviors sont eux aussi développés en JavaScript. Il existe à l'heure où j'écris ces lignes 7 behaviors différents, un qui définit le défilement de la zone de scrolling, un avec une apparence simple et un, définit pour les affichages de l'IRSAM alternant d'une couleur à une autre. Un behavior définit la manière d'affichage des informations, c'est lui qui place les informations données par le renderer sur la zone adéquate. Une zone ne peut posséder qu'un seul comportement, c'est le contrôleur de la zone qui choisit le behavior et qui l'initialise.

Le behavior de la seconde zone, celle qui affiche les tweets dans notre cas, utilise les propriétés animation introduites en CSS3. Cela permet d'avoir une animation assez fluide et légère contrairement à ce que l'on peut avoir en JavaScript par exemple. Le behavior de la zone principale affiche les informations une à une, pour cela, elle utilise le temps d'affichage donné par les renderers.

e. Layout pour l'IRSAM

L'IRSAM (Institut Régional des Sourds et Aveugles de Marseille), est un institut pour les déficients sensoriels qui dispose d'un centre à Nice.
Ce centre est équipé d'un écran de diffusion JSeduite qui est tombé en panne. JSeduite étant assez compliqué à maintenir, il leur a été proposé que l'on installe à la place une nouvelle version, issu de YourCast.
L'affichage des informations est, bien sûr, totalement différent de l'affichage que l'on peut trouver pour les journées du GDR (voir Figure 4). L'affichage se fait alternativement avec un fond noir puis un fond jaune, ce qui permet aux résidants de bien voir l'écran. Il a donc fallu totalement changer le style mais tout ça bien entendu en réutilisant les renderers préexistants. L'écran pour L'Irsam embarque donc les mêmes renderers, seules les zones changent, avec plus qu'une seule zone et les informations beaucoup plus grosses.
Il a fallu aussi développer un nouveau behavior pour gérer ce changement alternatif de couleur pour chaque information. Les Figures 5 et 6 ci-dessous montrent l'affichage du calendrier en noir puis en jaune.

Figure 6 Calendrier Irsam noir

Figure 7 Calendrier Irsam jaune

f. Interfaces d'administration

Durant mon stage, j'ai dû aussi développer une interface d'administration pour les annonces. Les annonces sont des messages entrés par l'utilisateur qui sont ensuite envoyés au provider et enfin affichés sur l'écran. Pour cela, il existait une interface de type REST permettant de faire le lien entre le provider et cette administration d'annonce. J'ai donc dû créer une interface web permettant de supprimer, ajouter, éditer ou trier des annonces. Pour cela j'ai utilisé plusieurs librairies, comme bootstrap qui est un Framework créé par Twitter permettant d'avoir des éléments déjà stylisés, mais aussi JQuery.

L'interface d'administration est totalement implémentée et fonctionnelle.

Dans le même style que les annonces, j'ai aussi dû développer un service d'administration de menus. L'utilisateur doit donc pouvoir ajouter des menus, des plats, mais aussi les supprimer, les trier ou les éditer.

Voir les figures 7 et 8 ci dessous présentant quelques captures de ces services.

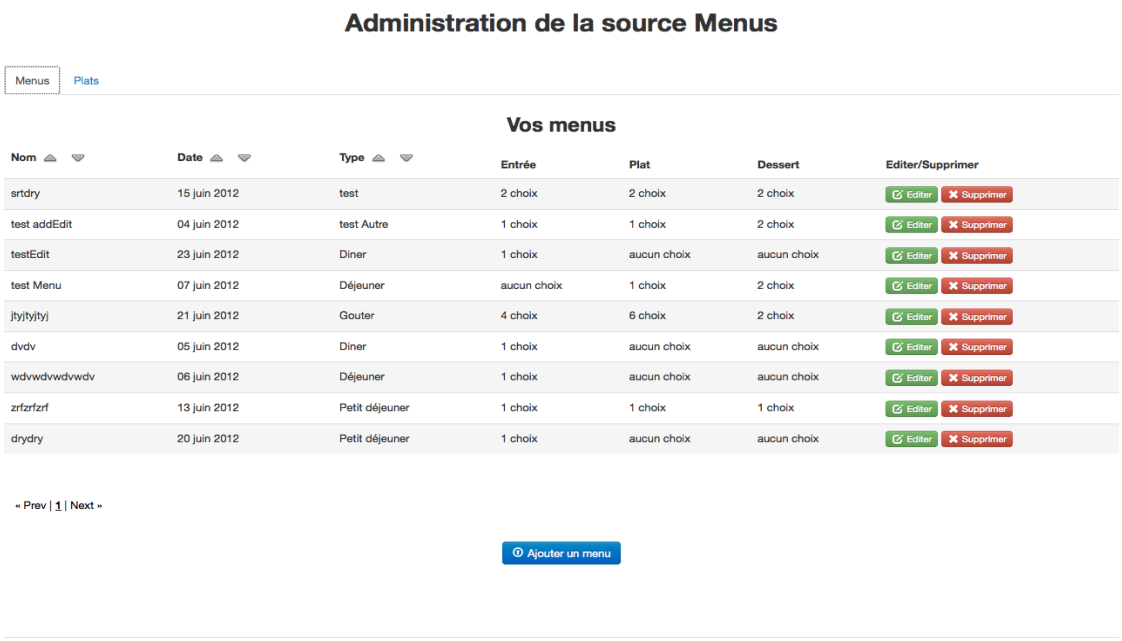


Figure 8 Administration source Menus

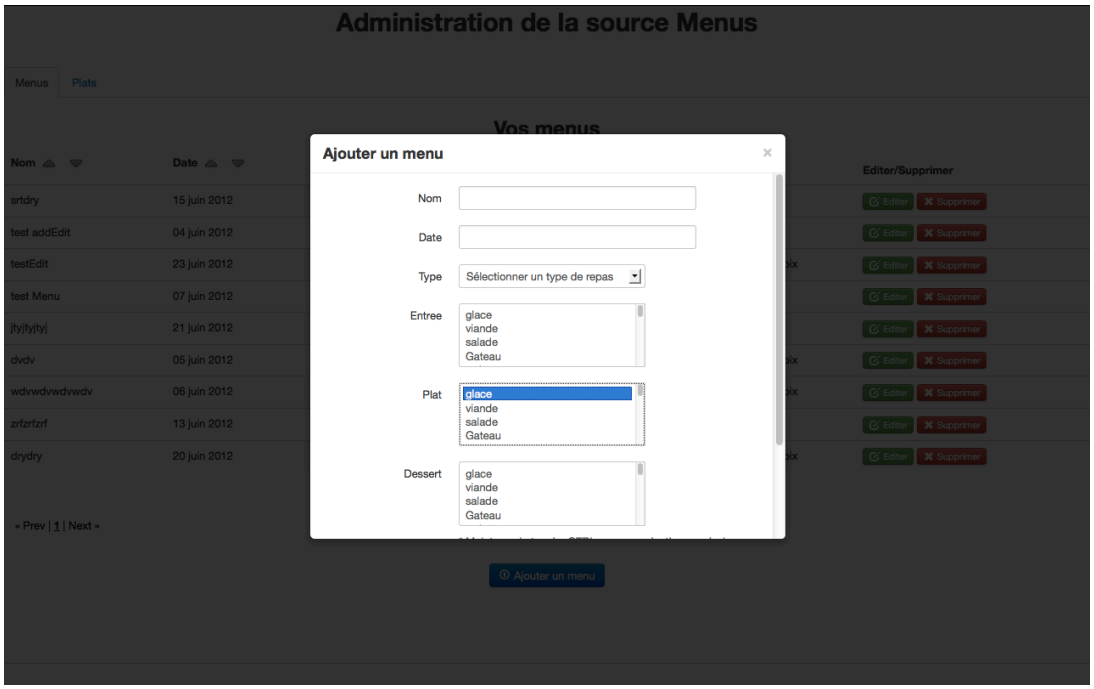


Figure 9 Ajout d'un menu

g. Les alertes

Un autre concept a dû aussi être implémenté, les alertes. Les alertes permettent d'arrêter totalement l'affichage en n'affichant qu'une seule information, cela peut être une alerte incendie et d'évacuation, mais aussi un enfant perdu par exemple.
Le problème pour les alertes est qu'elles possèdent leur propre provider, et doivent être au dessus de toutes les autres zones, elles doivent avoir un impact sur tout le système en arrêtant les mécanismes des autres zones.
Pour cela, il a fallu créer une nouvelle classe en JavaScript héritant de celle des zones, un concept d'héritage que je ne connaissais pas du tout en JavaScript.
Il y a donc sur l'écran de diffusion une zone d'alerte prenant tout l'écran, qui est donc invisible en temps normal et affichée seulement quand le provider possède une information.
La zone d'alerte a donc le contrôle sur toutes les autres zones et arrêtent celles-ci quand l'alerte est affichée.
La zone d'alerte intègre donc de nouveaux concepts qu'il a donc fallu adapter aux contrôleurs et autres éléments déjà créés.

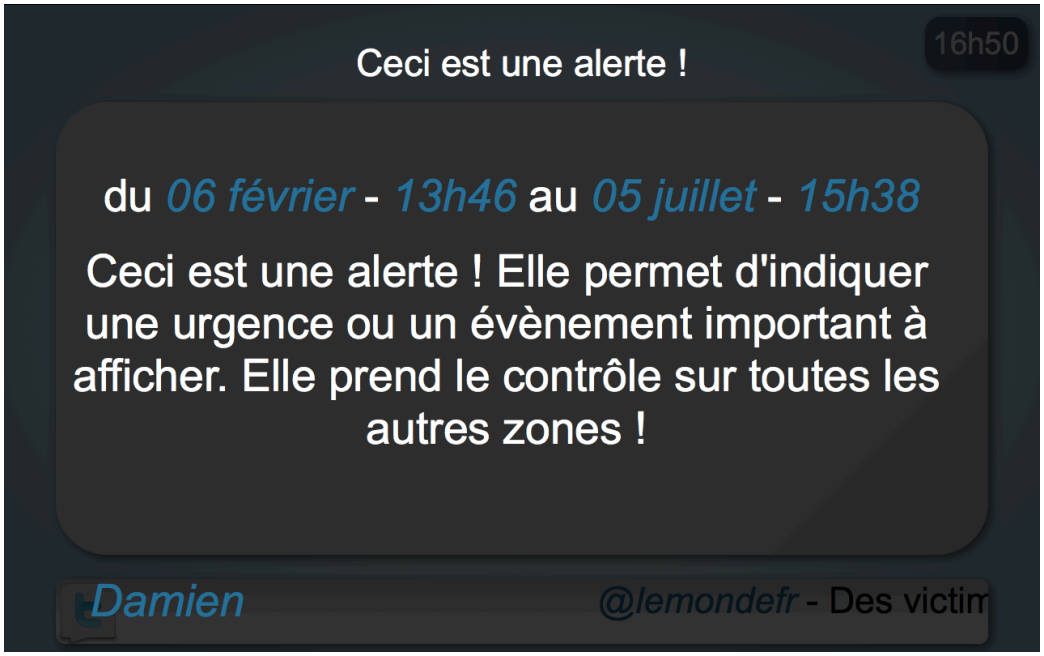


Figure 10 Capture d'une alerte

h. Problèmes rencontrés et solutions choisies

Les principaux problèmes rencontrés sont liés au fait que les informations sont dynamiques, nous ne savons pas à l'avance la taille du texte que nous recevons. De plus, à cette difficulté se rajoutent les problèmes de résolution d'écran. Les résolutions d'écrans sont un problème récurant en programmation Web et ne pas connaître la taille du texte crée beaucoup de débordements et d'affichages inattendus.

Pour pallier ce problème, j'ai utilisé les propriétés MediaQueries, introduites en css3, elles permettent de définir certaines propriétés en fonction des résolutions. J'ai donc dû travailler seulement en pourcentage, mais il faut alors savoir que le pourcentage d'une police s'apparente à la taille de police de son parent. Il a donc fallu définir une taille de police dans le Body comme une référence pour les autres tailles, c'est que l'on appelle, le point. Grâce aux MediaQueries, en fonction de la résolution de l'écran, j'ai pu adapter le point, cela a permis d'avoir un affichage adapté quelque soit la résolution.

Le fait de ne pas connaître la taille du texte que nous allons afficher a posé des problèmes plus importants, notamment avec les animations css3. Ces animations se basent sur une méthode de keyframe : c'est à dire un endroit où elle démarre, un endroit où elle s'arrête et au milieu des « keyframes » lui disant où aller. Le problème c'est que comme la taille nous est inconnue, il n'est pas possible de savoir où commencer et où s'arrêter.

Pour pallier ce problème il a fallu, en JavaScript, calculer la taille et envoyer dynamiquement l'animation dans le css.

Un autre problème rencontré est la différence d'interprétation des navigateurs, que ce soit du point de vue JavaScript ou css. Malgré le fait que nous n'ayons gardé que les navigateurs les plus modernes, ces problèmes inter-navigateurs sont quand même présents, notamment par exemple, des interprétations différentes au niveau des heures en JavaScript, ou au niveau des animations css3, ou il faut créer une animation par navigateur.

Nous avons aussi rencontré un problème quant à l'utilisation de JQuery, notamment, dans les interfaces d'administration des annonces ou menus. En effet, nous avons décidé d'utiliser JQuery, mais nous utilisions aussi Prototype, le problème est, que l'utilisation de ces deux librairies en parallèle est source de conflits. En effet, JQuery et Prototype utilisent tous deux, le symbole « $ » pour leurs appels. Pour gérer cela, nous avons donc du surcharger la librairie JQuery et utiliser le symbole « $j » pour faire des appels avec cette dernière.

Nous avons du faire face à un autre problème, du coté des renderers cette fois, avec la source photo. Pour les photos, nous avons voulu avoir un renderer affichant la photo en plein écran et un autre affichant les photos sous forme de mosaïque. Le problème de l'affichage en mosaïque est, d'une part le fait que nous ne savons pas à l'avance le nombre de photos à afficher, il fallait donc gérer ce cas, mais d'autre part un problème de taille et de forme d'image. Le problème étant donc le fait que si on fixait une taille par défaut, les images qui n'étaient pas en accord avec la taille choisie (verticale ou horizontale) étaient complètements déformées. Nous avons donc pour cela vérifié les photos une par une et les avons classées sous forme de tableaux, en fonction de leur forme, verticale ou horizontale. Les images sont donc ensuite affichées par groupe, les images de forme verticale sont affichées sur deux lignes, alors que les autres ne sont que sur une seule ligne.

Mais ce problème de forme d'images nous a confrontés à un autre problème encore, le chargement des images. Quand l'écran est lancé, les images peuvent ne pas être affichées si elles ne sont pas encore chargées. Pour cela, nous avons dû mettre en place un écran de chargement. À chaque fois qu'une image est insérée dans le projet, c'est le cas des photos provenant de l'extérieur mais aussi des logos ou images propres au projet, elle est tout d'abord chargée en mémoire et stockée dans le cache. La zone de chargement attend que toutes les images soient chargées avant de lancer l'affichage et permet alors d'éviter ces problèmes de chargements.

IV. Résultats

a. Planning effectif

Suite à ces 11 semaines de stage, on peut établir un planning réel. Le planning initial n'a pas été totalement respecté, le travail s'est déroulé d'une manière plutôt agile.
Les phases d'analyses et de tests sont conformes au calendrier précédemment établis, seule la phase de développement diffère. Dans le calendrier prévisionnel, il était prévu un certain ordre de développement, d'abord le layout principal, puis les renderers, puis les layouts de zone et enfin les behaviors. Mais le développement a plus été orienté sur une méthode de type agile et surtout adapté aux changements fréquents, apportés pour correspondre aux attentes du client potentiel.

b. Quantification

La quantification des codes peut être découpée en plusieurs parties :

- Tout d'abord les fichiers HTML/CSS définissant les layouts comptent environ 800 lignes de codes.
- Un renderer possède en moyenne 50 lignes de codes purs et 10% de commentaires. La totalité des renderers représente environ 1700 lignes de codes purs et 10% de commentaires.
- Le développement des interfaces d'administration, menus et annonces réunis représente environ 1400 lignes de codes purs et 5% de commentaires.
- Les behaviors représentent eux, 350 lignes de codes purs et 10% de commentaires.
- Les fonctions externes développées pour les renderers et behaviors représentent 650 lignes de codes et 10% de commentaires
- Les tests unitaires, environ 650 lignes.
- La totalité du projet s'élève donc a environ 5500 lignes de codes.

c. Tests Unitaires

Les tests unitaires permettent de tester le bon fonctionnement du programme, de ses différentes composantes. Les tests sont exécutés en JavaScript à l'aide du script JsTestDriver. Ce script permet de définir des tests, en JavaScript donc, sur plusieurs navigateurs, mais propose aussi un plugin éclipse, permettant d'effectuer et lancer les tests automatiquement depuis Eclipse et de visionner les résultats directement dans l'espace de travail. Voir Figure 10, Tests unitaires sur Eclipse avec JsTestDriver. Les tests sont donc découpés par navigateurs et permettent donc d'éviter au mieux les problèmes d'inter-compatibilité des navigateurs.

Les tests sont effectués à l'aide d'assertions, il en existe une multitude permettant de comparer un résultat attendu et celui obtenu, ou bien si le résultat est nul ou indéfini etc. Tous ces tests ont permis de découvrir certaines « failles », certains renderers ont par exemple été modifiés, conformément aux attentes de l'utilisateur, et grâce aux tests nous avons pu voir que certains n'avaient pas été modifiés. Les tests sont donc très importants, c'est pourquoi, nous avons décidé de les placer tout au long du projet permettant de les exécuter au fur et à mesure du développement

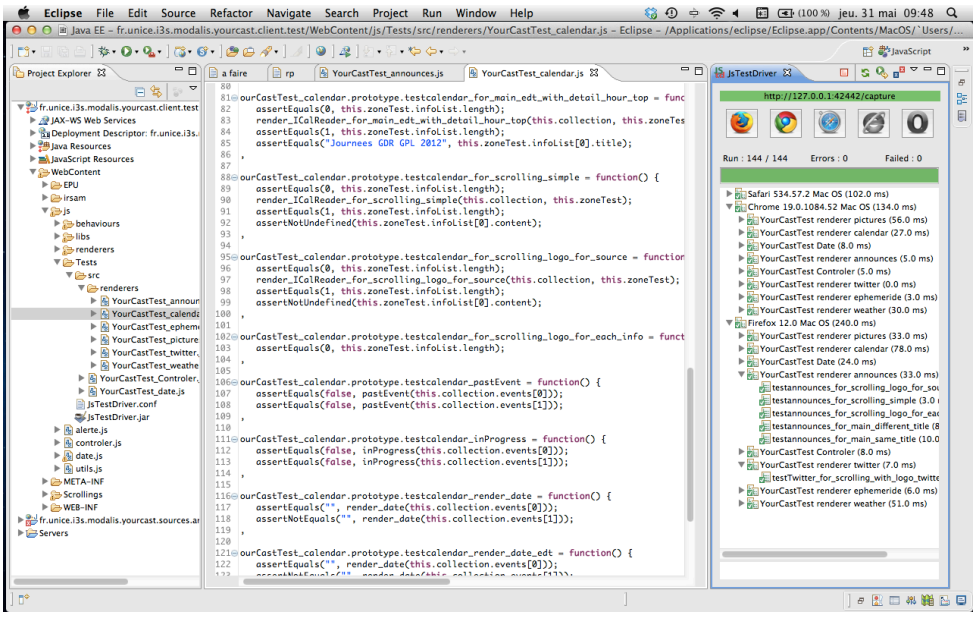


Figure 11 Tests unitaires sur éclipse avec JsTestDriver

V. Perspectives

a. Perspectives d'amélioration

À l'heure où j'écris ce rapport, mon stage n'est pas terminé et je continue donc après ma soutenance, ce qui me permettra d'améliorer certaines choses et d'en commencer de nouvelles.
Dans les perspectives d'avenir, il y a la création de nouveaux renderers et behaviors mais aussi de nouveaux usages, notamment la création d'une version pour l'institut Clément Ader, qui est un institut d'éducation sensorielle. Il faut donc créer de nouveaux renderers permettant de répondre à leurs usages mais aussi de nouveaux behaviors.
Le projet ne s'arrête bien sûr pas après mon stage et ma partie sera donc récupérée et modifiée pour d'autres usages, il faut donc que je continue les développements génériques et que je rajoute quelques commentaires pour bien expliquer mes codes.
Les tests sont aussi une perspective, il est possible d'en rajouter, de les améliorer et de continuer à les produire tout au long des développements futurs.

VI. Conclusion

a. Objectifs atteints

Aujourd'hui, l'interface est totalement implémentée, fonctionnelle et testée. Les objectifs imposés dans le cahier des charges ont été atteints et même certains développements annexes ont pu être implémentés.
L'objectif principal du stage est d'apprendre de nouvelles choses et en même temps de découvrir le monde professionnel, ces deux objectifs ont clairement été atteints et ce stage m'a été très bénéfique pour bien envisager la suite de mes études et pour le déroulement de ma future vie professionnelle.

b. Acquis

Durant le stage, j'ai dû utiliser un gestionnaire de version et une forge. Ces éléments m'ont permis de voir et mieux comprendre comment se déroule un projet et la manière de le gérer entre ses différentes parties et les personnes travaillant dessus. J'ai aussi pu en apprendre beaucoup sur certains concepts de développement et acquis certaines méthodes de travail et une rigueur que je n'avais pas auparavant.

Ce stage m'a énormément appris, que ce soit sur le plan des compétences, mais aussi sur le plan moral. Grâce à ce stage j'ai donc pu me faire une idée de ce que pourrait être ma future vie professionnelle, les contraintes comme les avantages.

Ce stage est donc une bonne expérience, très enrichissante et vraiment utile pour bien se faire une idée sur le monde professionnel et mettre en pratique tout ce que l'on a étudié lors de notre DUT.

Au point de vue développement, j'ai pu apprendre de nouveaux langages, concepts que je ne connaissais pas. Notamment l'utilisation d'outils de gestion de projet, mais aussi des serveurs web comme Tomcat, les tests unitaires en JavaScript, des librairies tel que prototype et bien d'autres encore..

Pour conclure, ces 11 semaines furent une très bonne expérience, très enrichissante, les personnes rencontrées ont été très sympas et on su me faire une place très rapidement.

Bibliographie

I3S – Présentation générale du laboratoire :
- http://www.i3s.unice.fr/I3S/

Modalis – Présentation générale de l'équipe :
- http://modalis.i3s.unice.fr/start

JsTestDriver – outil pour les tests unitaires :
- http://code.google.com/p/js-test-driver/

Prototype – doc prototype, script JavaScript :
- http://www.prototypejs.org/learn

StackOverFlow – forum d'aide informatique :
- http://stackoverflow.com/

RedMine YourCast – forge du projet :
- http://yourcast.unice.fr/redmine/projects/yourcast

YourCast – site du projet :
- http://yourcast.unice.fr

JSeduite – site du projet :
- http://www.jseduite.org

Table des illustrations

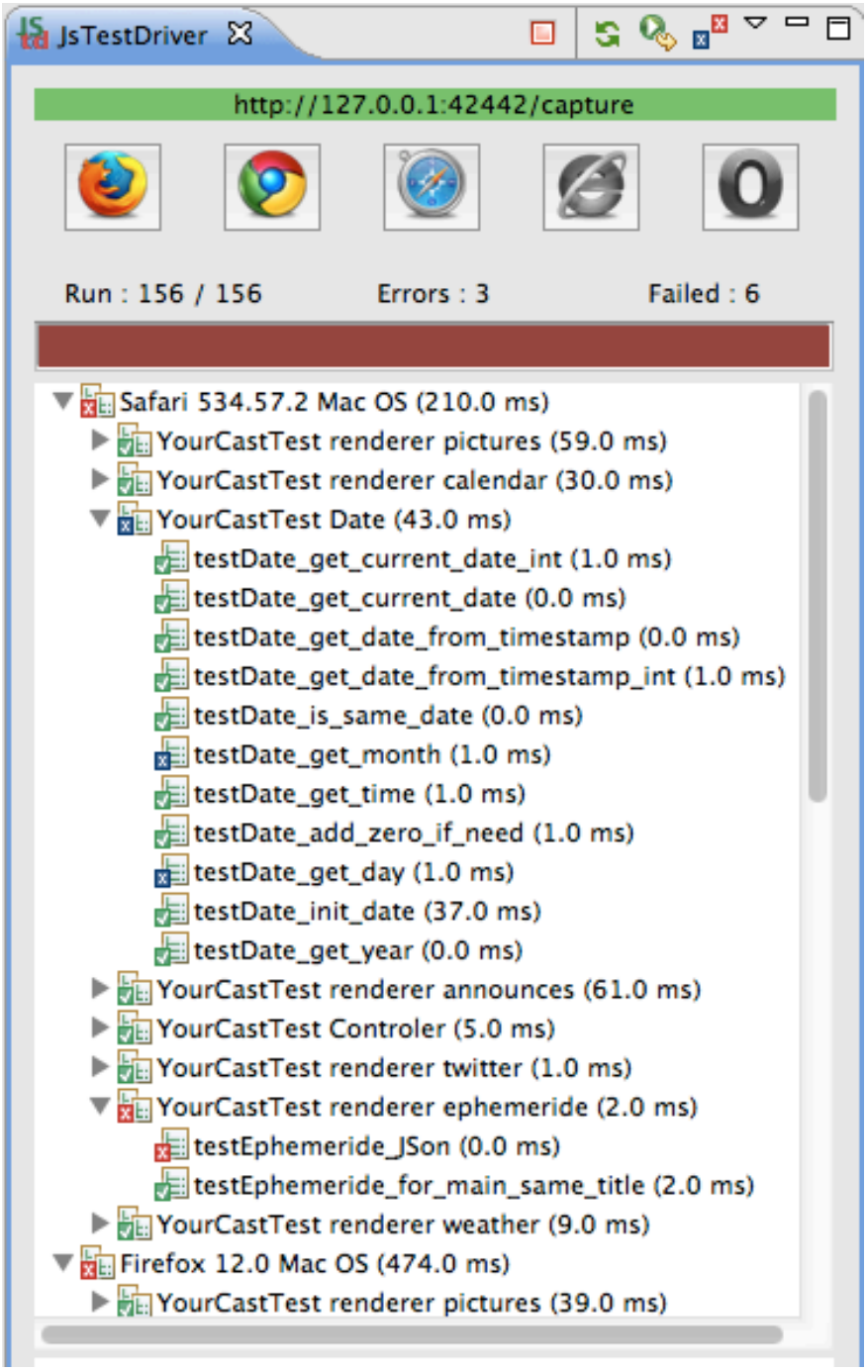
Annexes

Table des annexes

Annexe1

```
/*
 * work on weather source
 * display the current weather on the zone1 (main)
 * the logo and the content are modified, the title is always the same (Meteo)
 */
function _render_Weather_current_for_main_same_title(collection, zone, ville) {

    var logo = '<img src="img/logos/weather.png"/>';
    zone.loadImage("img/logos/weather.png");

    var title = "Météo";
    var time = 5000;

    var currentDate = new Date();
    var tableau = collection.weather.curren_weather;

    for (var indice = 0; indice < tableau.length; indice++) {
        elements = tableau[indice];
        currentCode = dicoMeteo[elements.weather_code];

        zone.loadImage(currentCode.day_icon);
        zone.loadImage(currentCode.night_icon);

        temp = elements.temp;
        temp_unit = elements.temp_unit;

        speed = elements.wind[0].speed;
        speed_unit = elements.wind[0].wind_unit;

        humidity = elements.humidity;

        if (temp == 0) {temp = "n/a";};
        if (temp_unit == "c") {temp_unit = "C";};
        if (speed == 0) {speed = "n/a"; speed_unit = "";};
        if (speed_unit == "kph") {speed_unit = "km/h";};
        if (humidity == 0) {humidity = "n/a";}
        else {humidity = humidity+" %";}

        content = "<div id='Weather_current' class='main_div_zone1'>";
                content += "<div class='currentWeather_city'><b> "+ville+" </b></div>";
                if (currentDate.getHours() <= 17) {
                    content += "<div class='currentWeather'><img src='"+currentCode.day_icon+"'/><p>"+temp+"°"+temp_unit+"</p></div>";
                }else{
                    content += "<div class='currentWeather'><img src='"+currentCode.night_icon+"'/><p> "+temp+"°"+temp_unit+"</p></div>";
                }
        content += "<div class='currentWeather_wind_rain'>";
            content += "<div class='currentWeather_wind'><img src='img/vent.png'/>  "+speed+" "+speed_unit+"</div>";
            content += "<div class='currentWeather_rain'><img src='img/pluie.png'/>  "+humidity+"</div>";
        content += "</div>";
        content += "</div>";

        var dico = {"content": content, "logo": logo, "title": title, "time" : time};
        zone.pushInfo(dico);
    }
}

```

Annexe 2


JsTestDriver
http://127.0.0.1:42442/capture
Run : 156 / 156
Errors : 3
Failed : 6
Safari 534.57.2 Mac OS (210.0 ms)
YourCastTest renderer pictures (59.0 ms)
YourCastTest renderer calendar (30.0 ms)
YourCastTest Date (43.0 ms)
testDate_get_current_date_int (1.0 ms)
testDate_get_current_date (0.0 ms)
testDate_get_date_from_timestamp (0.0 ms)
testDate_get_date_from_timestamp_int (1.0 ms)
testDate_is_same_date (0.0 ms)
testDate_get_month (1.0 ms)
testDate_get_time (1.0 ms)
testDate_add_zero_if_need (1.0 ms)
testDate_get_day (1.0 ms)
testDate_init_date (37.0 ms)
testDate_get_year (0.0 ms)
YourCastTest renderer announces (61.0 ms)
YourCastTest Controler (5.0 ms)
YourCastTest renderer twitter (1.0 ms)
YourCastTest renderer ephemeride (2.0 ms)
testEphemeride_JSon (0.0 ms)
testEphemeride_for_main_same_title (2.0 ms)
YourCastTest renderer weather (9.0 ms)
Firefox 12.0 Mac OS (474.0 ms)
YourCastTest renderer pictures (39.0 ms)

Annexe 3


```
{
  - informations: [
     - {
        - announces: [
           - {
                id: 1103405643,
                title: "une alerte !",
                author: "Damien",
                content: "Test d'alert !!! ",
                img: "",
                start: 1328532406540,
                end: 1341495496800,
                target: "all"
           }
        ]
     },
     - {
        - icalreader1: {
             name: "Journées GDR GPL 2012",
             date: 1340182800000,
             description: "Programme des journées GDR GPL 2012",
             url: https://www.google.com/calendar/ical/vpd9psgi66nuoadlh586s77ffg%40group.calendar.google.com/private-14b83e20d01a54d7d43eb1de278b6c1f/basic.ics,
           - events: [
              - {
                   summary: "CompilationA : Défis des architectures à venir, quelle compilation pour demain ? ",
                   description: "Erven Rohou (INRIA/IRISA)",
                   location: "salle A",
                   start: 1340182800000,
                   end: 1340184600000
              },
              - {
                   summary: "CompilationB : Défis des architectures à venir, quelle compilation pour demain ? ",
                   description: "Erven Rohou (INRIA/IRISA)",
                   location: "salle B",
                   start: 1340182800000,
                   end: 1340184600000
              },
              - {
                   summary: "CompilationC : Défis des architectures à venir, quelle compilation pour demain ? ",
                   description: "Erven Rohou (INRIA/IRISA)",
                   location: "salle c",
                   start: 1340182800000,
                   end: 1340184600000
              }
           ]
        }
     }
  ]
}
```

Annexe 4

```
/*
 * this is a behavior for the zone1 (main)
 * the behavior is a simple appearance of each info
 */
function simple_appearance_timeout(zone, indice) {
    if(zone.behaviour_running){
        var info = "";
        if (zone.timeoutBehav)
            clearTimeout(zone.timeoutBehav);
        info = zone.infoList[indice];
        zone.changeContent(info.content);
        document.getElementById(zone.id+"_logo").innerHTML = info.logo;
        document.getElementById(zone.id+"_title").innerHTML = info.title;
        indice = (indice+1) % zone.counterInfo;

        zone.timeoutBehav = setTimeout(function() { simple_appearance_timeout(zone, indice); }, info.time);
    }else {
        zone.timeoutBehav = setTimeout(function() { simple_appearance_timeout(zone, indice); }, 1000);
    }
}
```