TP 3

La couche Vue d'un projet d'application Web

L'objectif de ce TP est de développer les différents composants de la couche Vue du projet *ourface* dans le cadre de l'architecture MVC présentée dans le TP1. Dans ce TP, vous devez comprendre et implémenter ces composants.

Il vous est, par ailleurs, demandé de fournir une application accessible en version classique et en version mobile (smartphone[1]). Une réflexion toute particulière sur la disposition des différents éléments devra donc être conduite afin d'optimiser l'accessibilité de votre application pour les deux versions. Le framework CSS Bootstrap devra être utilisé dans ce sens (voir un peu plus loin pour une présentation rapide des éléments de Bootstrap nécessaires à votre prise en main).

Description :

- L'architecture MVC présentée lors du TP1 fait intervenir plusieurs composants de la couche Vue : un *layout* unique et plusieurs *vues* chacune en lien direct avec l'action exécutée et son état après exécution *(SUCCESS, ERROR, NONE).* Comme vu dans le TP1, le dispatcher définit le nom de la *vue* et charge le *layout ;* ce dernier charge ensuite la *vue* attendue (stockée dans une variable temporaire). Ce schéma devra être modifié dans ce TP de façon à faire en sorte qu'un *layout* puisse charger plusieurs *vues* différentes, notamment pour permettre d'afficher des ensembles de contenus différents sur une seule et unique page.

  Pour ce TP, seul le couple HTML/CSS est demandé, incluant l'utilisation du framework Bootstrap pour la mise en forme. Si du Javascript est nécessaire pour le traitement des données, nous l'implémenterons dans les séances futures.

- La mise en place du *layout* et des *vues* doit être réfléchie dans un contexte de site dynamique. En d'autres termes, les informations contenues dans le *layout* et les *vues* associées sont chargées dans leur globalité une première fois (dispatcher global), leur mise à jour se fera ensuite de manière partielle par des requêtes Ajax (dispatcher Ajax).

- Il est attendu que du PHP soit utilisé à l'intérieur du *layout* ou des *vues* pour afficher des informations provenant du modèle **via le contrôleur** (pas d'accès direct au modèle par appel aux classes outils). Nous utiliserons alors une syntaxe PHP adaptée à l'écriture de vues. Un exemple utilisant une boucle *foreach* pour lister un ensemble de données est présenté ci-dessous.

  ```
  <ul> <?php foreach( $context->tableData as $data ): ?>
  <li><?php echo $data ?></li> <?php end foreach; ?>
  </ul>
  ```

  Il est rappelé que l'utilisation du PHP au niveau de la couche Vue doit se limiter **au strict minimum** touchant à la récupération d'informations à afficher.

Implémentation :

1. La fenêtre d'identification étant déjà implémentée (possibilité d'associer cette page à un *layout* différent), il faut à présent pouvoir afficher **le layout de l'application** après identification ie la page *ourface* de l'utilisateur connecté. Ce *layout* devra contenir (au minimum) différentes vues permettant :

1 Des outils de simulation sont disponibles pour tester votre application en version mobile/smartphone. Un exemple, la fonctionnalité de Firefox disponible dans le menu Outils-> Développement web-> Vue adaptative.

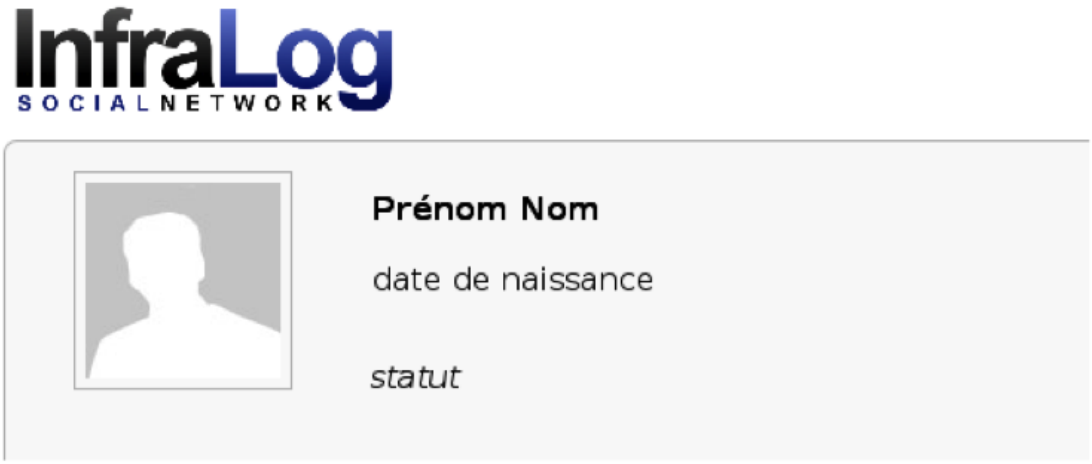

Illustration 1: Exemple de vue pour l'affichage du profil

- d'afficher le profil de l'utilisateur connecté ;
- d'afficher un bouton de déconnexion ;
- d'afficher la liste des amis ;
- d'afficher le mur ;
- d'afficher la fenêtre de chat ;
- d'afficher un bandeau de notification.

Note : Comme précisé précédemment, cela signifie que le layout devra charger non plus une seule vue stockée dans la variable $template_view mais un ensemble de vues. De même, cela signifiera que les informations dynamiques présentes dans chacune des vues devra faire l'objet de l'exécution d'une action du contrôleur (à prévoir dans les séances suivantes) permettant la récupération des données correspondantes.

2. La figure 1 fournit un aperçu possible d'une vue de profil. Celle-ci devra contenir :
   - une photo de profil (attribut avatar ; afficher une image par défaut si pas d'avatar)
   - les informations de l'utilisateur (nom, prénom, date de naissance, etc.)
   - le statut de l'utilisateur. Celui-ci pourra être modifié à tout moment à l'aide d'un formulaire pour l'utilisateur connecté uniquement. La consultation d'un profil d'un autre utilisateur que celui connecté ne permettra pas cette modification.
   - un formulaire d'envoi de messages dont le propriétaire du profil sera le destinataire.

3. Lorsque l'on clique sur un utilisateur du réseau social (présent dans la liste d'amis), il est alors possible d'afficher sa page *ourface* contenant son profil, son mur, la liste d'amis (commune à tous), … Le formulaire d'envoi de messages sera toujours à disposition sur la vue profil. Le destinataire des messages issus de ce formulaire sera alors l'utilisateur courant ie celui dont le profil est actuellement en consultation.

4. La vue du mur permettra de visualiser les messages du profil consulté (ou un sous-ensemble

restreint suivant le nombre de messages dans la base). La visualisation d'un message comprendra toutes les informations qui s'y rapportent : émetteur, destinataire, date, texte… Devront être également affichés le nombre de personnes ayant voté pour le message (champ *aime)* et des informations relatives à un éventuel partage de celui-ci. Par ailleurs, les dispositifs de vote et de partage devront être mis à disposition.

Note : Un message pourra être considéré comme une sous-vue, la vue du mur contenant alors un ensemble de sous-vues de type message à l'intérieur de la vue mur.

5. Finalement, le *layout* devra inclure une fenêtre de chat fournissant les fonctionnalités d'une messagerie instantanée, notamment l'affichage des (derniers) messages du chat. Afin d'accroître l'ergonomie de cette messagerie, la fenêtre de chat devra répondre aux exigences suivantes : (1) la fenêtre sera une zone déplaçable (type drag&drop) et redimensionnable, (2) elle sera munie d'un onglet (ou autre forme) permettant de la réduire ou de l'agrandir, (3) même réduite, la mise à jour des messages devra être effective dans la fenêtre et un élément visuel (clignotement de l'onglet, changement de couleur, …) permettra de faire savoir à l'internaute que de nouveaux messages ont été postés. Vu le nombre important de messages qui pourront être postés, nous limiterons le nombre de messages à afficher à 10^2. Il est aussi possible, depuis cet espace, d'envoyer un message de type *chat* (n'ayant pas de destinataire contrairement à un message posté sur un mur). Un formulaire d'envoi de messages devra donc être mis en place. Il s'agit ici de prévoir les différents éléments visuels, le codage des différents fonctionnalités sera prévu dans les séances à venir.

La messagerie instantanée ne sera disponible que dans la version classique de votre application.

Le framework bootstrap

Quelques informations sur le framework bootstrap que vous devez utiliser, notamment pour permettre l'accessibilité de votre application sur une version classique et mobile.

Bootstrap est un framework CSS incluant également des composants HTML et Javascript. L'une de ses fonctionnalités phare est de fournir une grille simple de positionnement des éléments d'une page web permettant l'adaptation des applications au type de média utilisé (et notamment la taille de leur écran et fenêtre de visualisation – *Responsive design*). Il permet également de normaliser le rendu des pages quel que soit le navigateur utilisé (feuille normalize.css) et fournit des effets de style pour les boutons, formulaires, … Différents templates sont également disponibles pour démarrer le développement d'une application.

En tant que framework CSS (ie principalement un fichier CSS), Bootstrap propose donc un ensemble de classes CSS à utiliser dans les balises HTML5 pour bénéficier de ses différentes fonctionnalités.

1. La grille bootstrap ...

Bootstrap propose une grille de 12 colonnes (max par défaut) permettant de positionner les éléments d'une page. Toutes les colonnes ont la même largeur, correspondant à un pourcentage de la largeur de la fenêtre de visualisation. Les lignes de la grille prennent quant à elles la hauteur du plus grand élément qu'elles portent.

Bootstrap considère 4 types de media : les smartphones (type xs – x-small - jusqu'à 768 pixels), les tablettes (type sm – small - jusqu'à 992 pixels), les écrans moyens (type md – medium - jusqu'à

2 En bonus, vous pourrez implémenter un système de pagination permettant d'accéder aux messages plus anciens.

1200 pixels) et les grands écrans (type lg - large - au-delà de 1200 pixels).

Il fournit la classe CSS *row* permettant de définir une ligne (à inclure dans une balise HTML de type *div)* et 12 classes * 4 types de media permettant de définir les colonnes :

- *col-xs-1 … col-xs-12*,

- *col-sm-1 … col-sm-12*,

- *col-md-1 … col-md-12*,

- *col-lg-1 … col-lg-12*.

Il est ainsi possible de définir initialement le positionnement d'un ensemble d'éléments en fonction des différents media. Une fois ces positions définies au préalable, Bootstrap va en fonction de la taille du media redimensionner la taille des éléments (en réduisant la taille des colonnes), jusqu'à finir par les empiler pour ne pas les rendre illisibles. Dans le cas où le positionnement n'est spécifié que pour un seul media (le grand écran par exemple), les règles de redimensionnement et d'empilement s'appliqueront, d'où l'intérêt de spécifier le positionnement pour des media précis pour éviter d'avoir des rendus non attendus. Il est possible, dans ce cas, de combiner les classes *col-type-nbcol (type* prenant comme valeurs *xs*, *sm*, *md* et *lg* et *nbcol* les valeurs de *1* à *12*) pour un même élément (ex : * … *). Le positionnement de l'élément dépendra alors du media rencontré.

Les classes *container* et *container-fluid* (potentiellement dans une balise *div*) permettent de centrer la grille vis-à-vis de la fenêtre de visualisation (avec des marges normalisées) avec une taille fixe pour la première classe et la largeur totale de la fenêtre pour la seconde.

Les classes de type *col-type-offset-nbcol* permettent de sauter des colonnes (colonnes vides entre des éléments).

Les classes *visible-type-display* (avec *display* prenant comme valeurs *block*, *inline*, *inline-block*) permettent de rendre visible un élément uniquement sur le media spécifié. A l'opposé, les classes *hidden-type* permettent de faire disparaître un élément pour un media visé.

Il est également possible d'imbriquer des lignes dans des lignes pour des rendus plus complexes.

2. Eléments de formes

Bootstrap propose un ensemble de classes propres aux différents éléments du langage HTML qui permettent d'améliorer leur rendu : liste, table, formulaire, boutons, … Nous vous laissons parcourir la documentation Bootstrap pour découvrir ce dont vous avez besoin pour votre application : http://getbootstrap.com/css/

Répartition du travail

Au sein de chaque binôme, le travail doit se répartir de la manière suivante.

- Une réflexion collective sur la disposition des différents éléments de l'application en version classique et en version mobile ainsi que sur la définition de la charte graphique pour le site.

- Deux parties distinctes (1 par étudiant) :
  - → gestion du mur (personnel ou amis) et de la liste d'amis.
  - → gestion du chat et du profil personnel

Tout comme pour le premier TP, des commentaires bien fournis devront expliciter l'auteur de chaque morceau de codes (sous peine d'une forte pénalité sur la note finale).