

# APPREF - BRI

Développement d'une application serveur capable de charger et recharger des services distants, et de l'application cliente associée.

**DEVELOPPÉ PAR**  
**Clément PROST**

**Clément MAUPERON**



# DETAILS TECHNIQUES

## SERVEUR


L'application serveur correspond à la classe principale **BRILaunch** du package `bri.serveur` (projet *infrastructure*). Cette classe lance les deux applications `AppAmateur` et `AppProgrammeur` (package *bri.serveur.app*, implémentation de l'interface *IApp*) dans des threads dédiés. La méthode `main` ajoute aussi l'utilisateur par défaut `admin` (mot de passe : `admin`) et ses services (*Inversion*, *AnalyseXML* et *Messagerie*). La liste des utilisateurs est accessible en ressource patragée, via la classe `Utilisateurs`.

Les applications serveurs étendent la classe abstraite `bri.serveur.app.App`, qui s'occupe de l'acceptation des connexions clientes et l'ouverture des sessions individuelles (définies dans le package *bri.serveur.app.session*, implémentations de l'interface *ISession*). Les sessions peuvent lancer des actions (définies dans le package *bri.serveur.app.session.actions*, implémentations de l'interface *bri.serveur.app.session.IAction*). Chaque application dispose d'une classe de session dédiée telle que :

- **SessionAmateur**, qui affiche la liste des services disponibles et demande au client de choisir dans cette liste. Une instance du service est alors créée et lancée dans la session ;
- **SessionProgrammeur**, qui demande une authentification en tant que programmeur, avec 3 essais maximums par session. Si l'authentification est réussie, la session affiche la liste des actions relatives aux programmeurs et demande au client de choisir dans cette liste.

Le serveur et le client utilisent une surcouche du Socket natif de Java pour communiquer (*bri.Connexion*). Cette classe implémente différentes méthodes utilitaires, comme l'envoi d'un tableau, la demande de choix dans un tableau ou la demande d'un fichier.



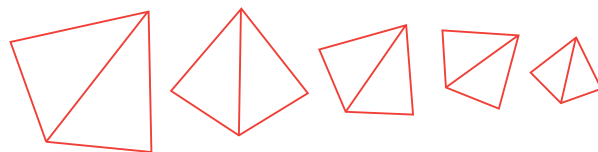


Les **services BRI** développés par les programmeurs doivent implémenter l'interface **IService** (package **bri.serveur.service**). Les classes dynamiques des services **BRI** sont gérées par la classe **Service**, qui implémente les méthodes de chargement, d'activation et de mise à jour de la classe distante depuis le serveur **FTP** du programmeur. Les services sont chargés à l'aide d'une instance propre de **ServiceClassLoader** (package **bri.serveur.service**), qui autorise le rechargement d'une classe déjà chargée. Les services peuvent également définir et accéder à des ressources partagées via la classe **Ressources** du package.

## CLIENT

L'application cliente est définie à la classe principale **Client** du package **bri.client** du même projet. Au démarrage, l'application demande quel mode doit être lancé, un mode correspondant à une application serveur (définis dans le package **bri.client.modes**, implémentations de la classe **bri.client.IMode**).

Dans son fonctionnement, le client ne fait que recevoir et afficher les demandes envoyées par le serveur.



## SERVICES

Les services **BRI** implémentés sont définis dans le projet **implémentation**. Ils sont attribués à l'utilisateur par défaut et appartiennent donc au package **admin**.

Des versions compilées des services sont disponibles dans le dossier **ftp** du projet et peuvent être directement déposés sur un serveur **FTP**.

## INVERSION

Cette implémentation est un exemple basique d'un service **BRI**.

# ANALYSE XML

Ce service propose aux **utilisateurs authentifiés** de vérifier la syntaxe d'un fichier XML fourni par l'utilisateur, et ce selon un fichier XSD que fournira également l'utilisateur. Les fichiers sont transmis par le client sur demande du serveur : *le client écrit directement le contenu du fichier sur la connexion.*

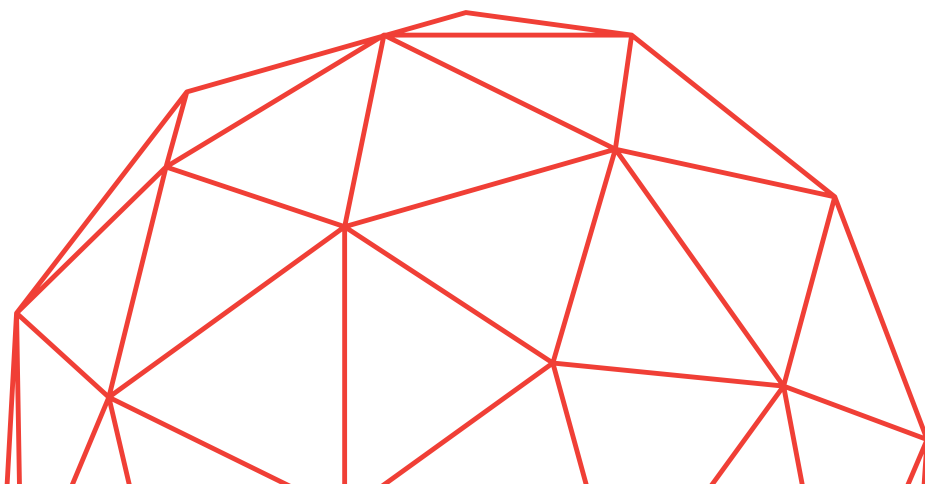
Le service présente donc notamment l'accès aux **ressources partagées** (*utilisateurs*) et le **transfert de fichier**.

**Aucun rapport mail n'est envoyé. Le service indique seulement si le fichier XML respecte le schéma XSD. Si ce n'est pas le cas, le service affiche la première erreur rencontrée.**

# MESSAGERIE

Le service de messagerie propose aux **utilisateurs authentifiés** (*utilisateurs normaux ou programmeurs*) d'envoyer des messages aux autres utilisateurs et de lire leurs messages reçus. La liste des messages est enregistrée en tant que **ressource partagée** auprès du serveur, sous la forme d'une liste de *String* représentant les **information sérialisées** des messages (*émissaire, destinataire, contenu, date d'envoi*).

**Les messages doivent en effet être sérialisés car les différentes instances du service chargent une classe Message indépendante des autres. En transformant les messages en String, on assure la cohérence des données entre ces instances, mais également entre les différentes mises à jour du service.**





# AMÉLIORATIONS

La plateforme présente en l'état actuel plusieurs axes d'améliorations :

- ajout d'une **persistance des données** pour les utilisateurs et les services, par exemple via une base de données *SQL* ;
- fonction de **hashage des mots de passes** des utilisateurs ;
- **fermeture automatique des sessions inactives**, avec une sorte de *garbage collector* consultant la liste des sessions ;
- ajout d'un **timeout** sur les méthodes de lecture de *bri.Connexion* ;
- **journalisation** des connexions et des erreurs ;
- implémentation d'un **serveur mail**, pour l'envoi des rapports d'analyse XML notamment ;
- **assurer la mise à jour complète d'un service** en fermant toutes les instances ouvertes ;

<https://github.com/tensaiji/APPREF-BRI>