

Qualoo: le générateur d'applications J2EE



ST Informatique
SERVICES

QUALOO

- Présentation
- Architecture de l'application
- Code source
- Les avantages
- L'application obtenue
- La customisation: le métier, la présentation
- Le périmètre actuel et futur
- Contacts

Présentation

➤ Besoins récurrents d'une application web :

- ★ Couche de persistance
- ★ Couche de présentation avec les fonctionnalités standard
- ★ Choix d'une architecture pertinente (conception et produits)
- ★ Code clair permettant une maintenance aisée

Présentation

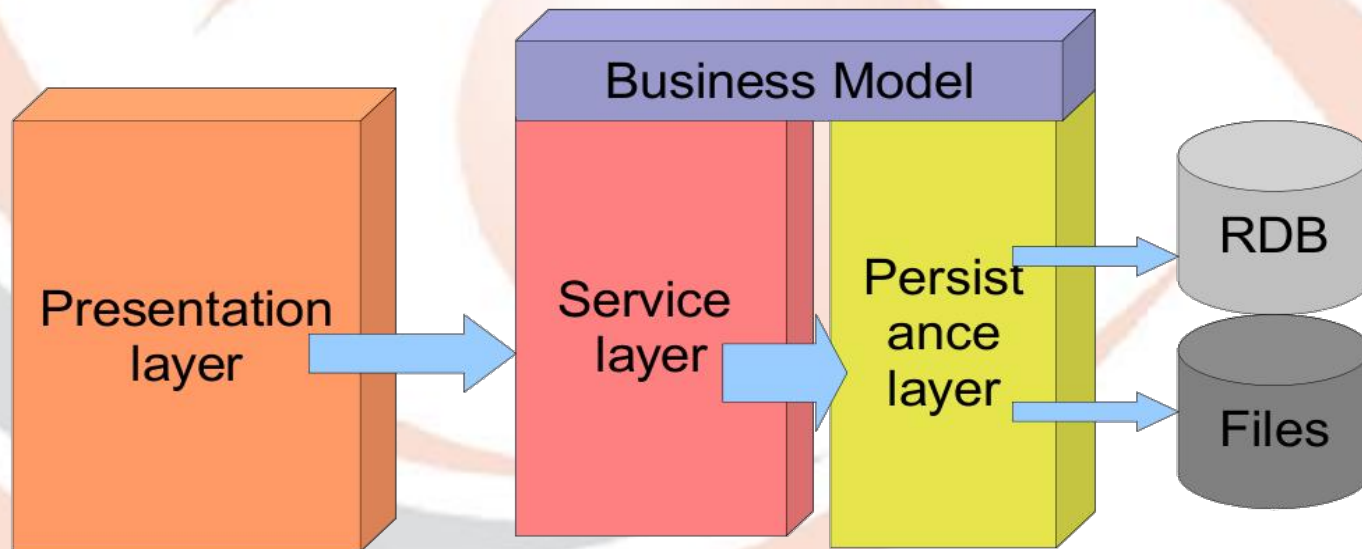
- Solution proposée : la génération automatique
 - ★ Un générateur en Java qui crée automatiquement une application web J2EE à partir d'une base de données relationnelle.
 - ★ Un outil qui permet de faire des projets J2EE beaucoup plus rapidement
 - ★ Une solution basée sur les expériences d'une équipe d'architectes très expérimentés (architecture et code)
 - ★ Une solution validée par des projets clients.

Présentation

- Résultat de la génération : Une application
 - ★ Java/J2EE
 - ★ Compatible avec les principales bases de données (Oracle, Mysql, ...)
 - ★ Disposant d'une architecture validée, basée sur des standards reconnus
 - ★ Utilisant les frameworks les plus courants (Hibernate, Spring, Struts ...)
 - ★ Compatible avec Java 1.4 ou 1.5
 - ★ Basé sur des outils Open-source
 - ★ Directement déployable sur un serveur d'application ou conteneur web.
 - ★ Facile à customiser et à faire évoluer.

Architecture de l'application

- Architecture à 3 couches :
 - ★ Couche présentation
 - ★ Couche métier (service)
 - ★ Couche d'accès au données



Architecture de l'application

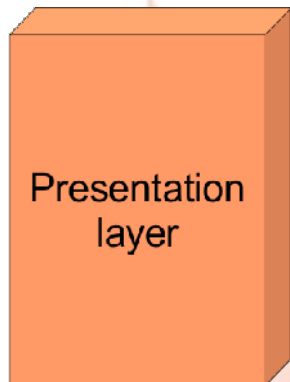
Avantages d'une architecture à 3 couches :

- Couches indépendantes les unes des autres
- Facilité de maintenance
- La gestion des données et la logique métier peuvent être indépendantes du type d'interface
- Le travail en équipe est optimisé
- Factorisation de code et utilisation de framework ou composants génériques (gain de temps et performances)

Architecture de l'application

➤ la couche présentation :

- ★ Affiche les données
- ★ Envoie les demandes de l'utilisateur à la couche métier pour qu'elle les effectue
- ★ Reçoit les résultats renvoyés par la couche métier et les affiche




Basée sur l'utilisation de framework MVC notamment Struts

L'utilisation de frameworks MVC permet d'organiser la couche présentation pour qu'elle soit plus robuste, plus fiable et plus facile à faire évoluer

Architecture de l'application

➤ la couche métier (service) :

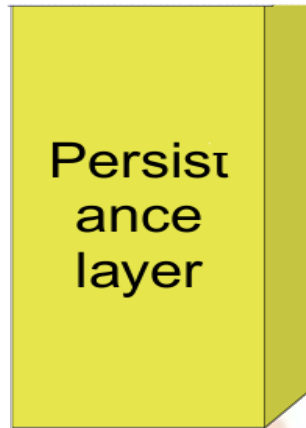


Service
layer

- ★ Reçoit et analyse les demandes de l'utilisateur
- ★ Retrouve et modifie les données via la couche données
- ★ Renvoie les résultats à la couche présentation
- ★ Basée sur des Javabeans

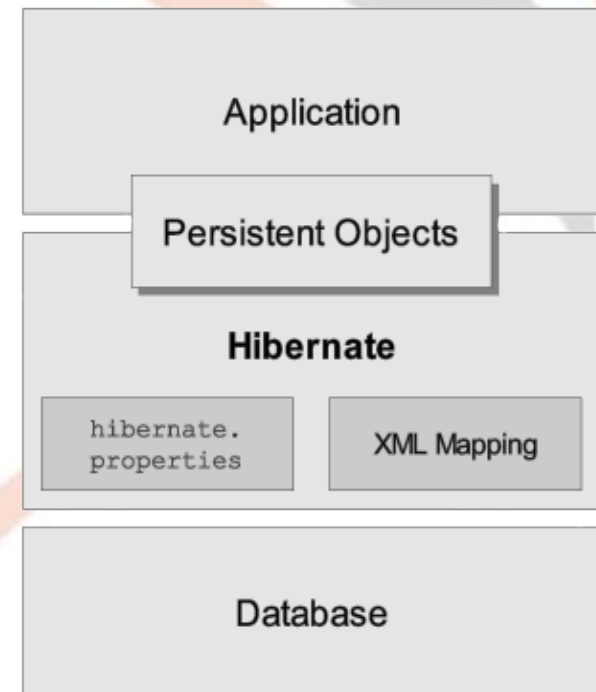
Architecture de l'application

➤ Utilisation d'hibernate pour la couche d'accès aux données:



- ★ Assure les fonctions CRUD
- ★ Assure la sécurité et l'intégrité des données

- ★ Support d'hibernate 2 et 3
- ★ Portabilité vers différentes bases de données
- ★ Possibilité de customiser hibernate pour améliorer les performances



Architecture de l'application

➤ Utilisation de Spring pour les liens entre couches:

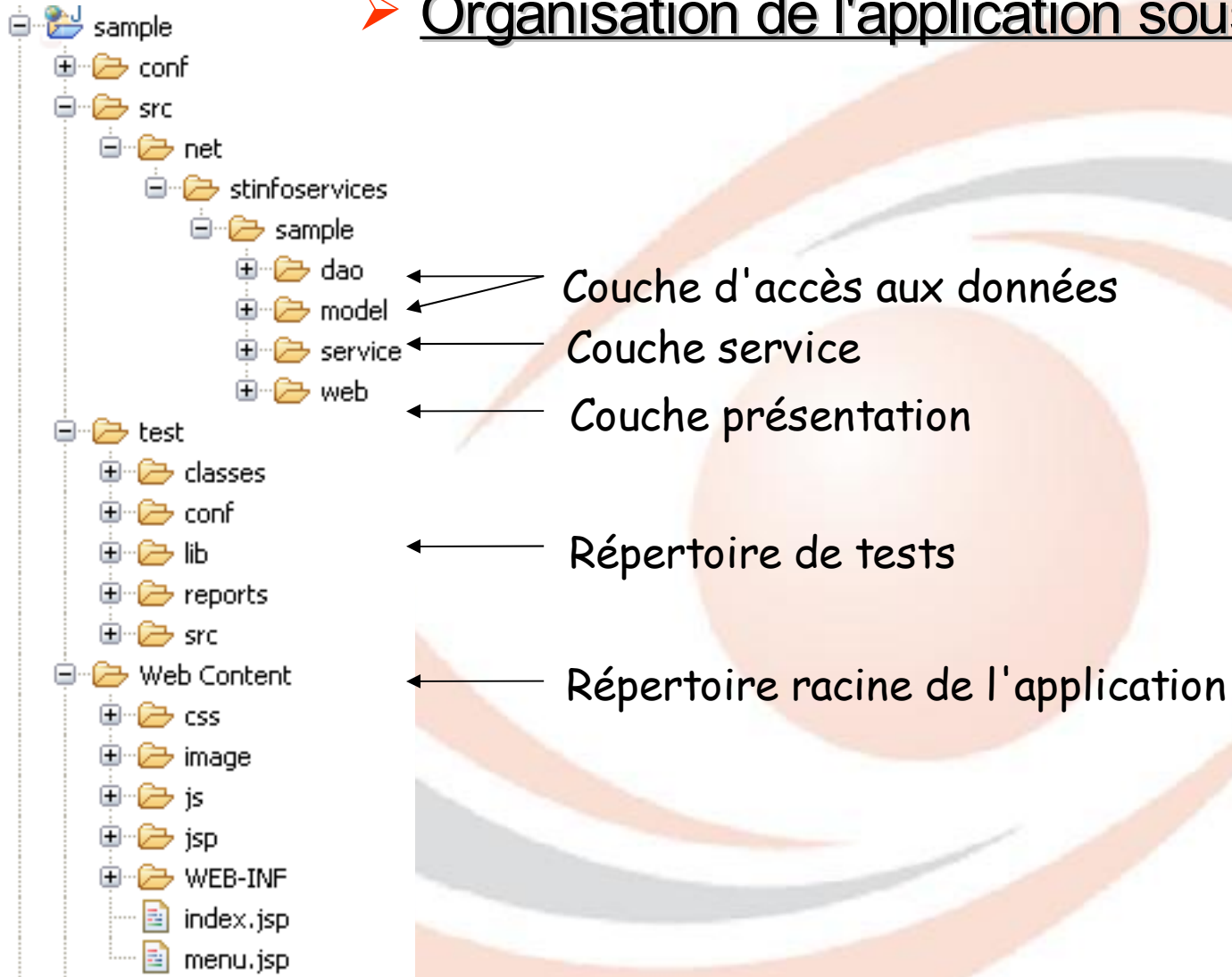
The logo for the Spring Framework, featuring the text "Spring Framework" in a white serif font on a green background with a small yellow leaf icon above the 'i' in "Spring".

Spring Framework

- ★ Permet de laisser les couches indépendantes les unes des autres
 - ✧ configuration au niveau du conteneur
 - ✧ Idéal pour de la conception à base d'interfaces (DAO)

Code source

➤ Organisation de l'application sous forme de package:



Code source

- Facilement maintenable
- Peut être importé directement dans Eclipse
- Le projet contient des Junits (pour les couches service et d'accès aux données) ainsi que des HttpUnits (pour la couche présentation)
- Internationalisation de l'application (anglais et français)

➤ Code commenté et formaté

Exemple code source de struts pour la couche présentation

```
/**
 * Execute the <code>ListAAction</code>.
 * @param mapping The <code>ActionMapping</code> used to select this instance.
 * @param form The optional <code>ActionForm</code> bean for this request (if any).
 * @param request The HTTP request we are processing
 * @param response The HTTP response we are processing
 * @throws Exception if an error occurs.
 */
public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response)
    throws java.lang.Exception {
    try {
        // Get the AService
        AService srvc = ServiceFactory.getInstance().getAService();

        // Get all A
        List<A> list = srvc.getAll();
        request.setAttribute("ALIST", list);
    } catch (ServiceException e) {
        if (e.getMessageKey() != null) {
            ActionErrors messages = new ActionErrors();
            messages.add(ActionMessages.GLOBAL_MESSAGE,
                new ActionMessage(e.getMessageKey(), e.getValues()));
            saveErrors(request, messages);
        } else {
            throw e;
        }
    }

    return mapping.findForward("success");
}
```

Code source

Exemple code source de la couche présentation

```
/**
 * Get the <code>A</code> specified by its primary key.
 * @param pk The primary key of the <code>A</code> to get.
 * @return The <code>A</code> found, null otherwise.
 * @throws ServiceException if an error occurs.
 */
public A findByPK(Serializable pk) throws ServiceException
{
    try {
        // Check arguments
        if (pk == null) {
            throw new IllegalArgumentException("pk
parameter is null");
        }

        // Find by primary key
        A a = this.aDao.findByPK(pk);

        return a;
    } catch (Throwable t) {
        if (trace.isErrorEnabled()) {
            trace.error("Could not find A by PK", t);
        }

        throw new ServiceException(t);
    }
}
```

```
/**
 * Save the specified <code>A</code>.
 * @param a The <code>A</code> to save.
 * @return The newly created <code>A</code>.
 * @throws ServiceException if an error occurs.
 */
public A create(A a) throws ServiceException {
    try {
        // Check arguments
        if (a == null) {
            throw new IllegalArgumentException("a
parameter is null");
        }

        // Create
        A result = this.aDao.create(a);

        return result;
    } catch (Throwable t) {
        if (trace.isErrorEnabled()) {
            trace.error("Could not create A", t);
        }

        throw new ServiceException(t);
    }
}
```

Code source

Exemple code source de la couche d'accès aux données

```
/**
 * ADaoHb8 Hibernate3 implementation.
 * @author STI Generator
 * @spring.bean id="aDao"
 * @spring.property name="sessionFactory" ref="sessionFactory"
 */
public class ADaoHb8 extends HibernateDaoSupport implements ADao {
    /**
     * Default constructor, constructs a <code>ADaoHb8</code>.
     */
    public ADaoHb8 () {
        super();
    }

    /**
     * Get all the <code>A</code> from the database.
     * @return All the <code>A</code> from the database.
     * @throws PersistenceException if an error occurs.
     */
    public List<A> getAll() throws PersistenceException {
        Session session = SessionFactoryUtils.getSession(getSessionFactory(),
            false);

        try {
            Query query = session.getNamedQuery("a.all");
            List<A> objList = query.list();

            return objList;
        } catch (Throwable t) {
            throw new PersistenceException(t);
        }
    }
    ...
    ....
}
```

Les avantages

- Amélioration de la qualité
 - ★ Conception et framework créés par des architectes J2EE
 - ★ Architecture et intégration des produits validées
 - ★ Code basé sur les bonnes pratiques
 - ★ Permet aux développeurs de focaliser leur énergie sur l'implémentation du code spécifique métier
 - ★ Applications faciles à tester
- Réduction des risques liés aux choix des technologies
 - ★ Solutions choisies par des experts
 - ★ Technologies matures et fortement utilisées
 - ★ Technologies connues des développeurs

Les avantages

- Réduction des coûts
 - ★ Moins de code à implémenter
 - ★ Pas besoin d'experts pour les évolutions
 - ★ Facilité de maintenance (architecture, code)
 - ★ Optimisation de la maintenance (architecture commune)
- Réduction du temps de développement
 - ★ Pas besoin de faire de la conception sur le design applicatif
 - ★ Pas besoin d'implémenter le framework de l'application
 - ★ Pas besoin d'écrire les fonctionnalités basiques
 - ★ Le code généré sert d'exemple pour les évolutions

L'application obtenue

- Page d'affichage avec la liste de toutes les tables
- Sélection d'une table : affichage colonnes et lignes
 - ★ Consulter, créer, modifier et supprimer pour chaque ligne
 - ★ Création: ouverture d'un formulaire de saisie
 - ★ Modification : affichage du formulaire pré-rempli
 - ★ Tri de 1er niveau sur une colonne
- Écran d'affichage des relations entre tables
- Validation de la saisie
 - ★ Contrôle du type :date, nombre
 - ★ Validation de la saisie obligatoire (not-null), si nécessaire

La customisation : le metier, la présentation

- La couche métier
 - ★ Possibilité de détacher la couche métier de la présentation pour la lier à une autre couche (par exemple un client lourd)
 - ★ Les liens entre la couche présentation et la couche métier et ceux entre les couches métiers et de persistance sont réalisés. Il suffit d'ajouter le code spécifique au métier.
- La couche présentation
 - ★ L'architecture de la navigation étant faite, pour customiser la présentation il suffit de changer la configuration et le code Struts
 - ★ Le style des pages se fait par l'intermédiaire de feuilles de style CSS, il suffit donc de les changer pour changer de style

Le périmètre actuel et futur

➤ Le périmètre actuel

★ Choix des frameworks

- ✧ Struts, JSTL et CSS pour la couche présentation

- ✧ Spring pour les liens entre couches

- ✧ Hibernate 2 et 3 pour la couche de persistance

- ✧ Support de JDK 1,4+

- ★ Possibilité de générer des tests JUnits et HttpUnit

- ★ Personnalisation du style CSS.

- ★ Support d'Oracle, MySql, PostgreSql, HsqlDB, SqlServer

- ★ Opération de création, de modification, de suppression, de visualisation et de tri sur les Objets.

Le périmètre actuel et futur

➤ Le périmètre futur

- ★ Support de Spring MVC et Struts2 pour la couche présentation
- ★ Mise en place de filtres sur les colonnes
- ★ Introduction de composants AJAX.
- ★ Génération d'application client lourd (type swing)

Contacts

Pour toute information complémentaire:

ST Informatique

65, allée de Bellefontaine - 31100 Toulouse

Tel: 05 61 43 76 24 Fax: 05 61 41 72 48

dominique.turpin@stinfoservices.net

www.stinfoservices.net