Cours 1 : Architecture Web

Mercredi 17 Février 2021

Tomy Bezenger (tbezenger@excilys.com)
Antoine Flotte (aflotte@excilys.com)

Excilys
Développeurs de passion

Excilys
Développeurs de passion

SOMMAIRE

Architecture d’une application web

Définition

Définition

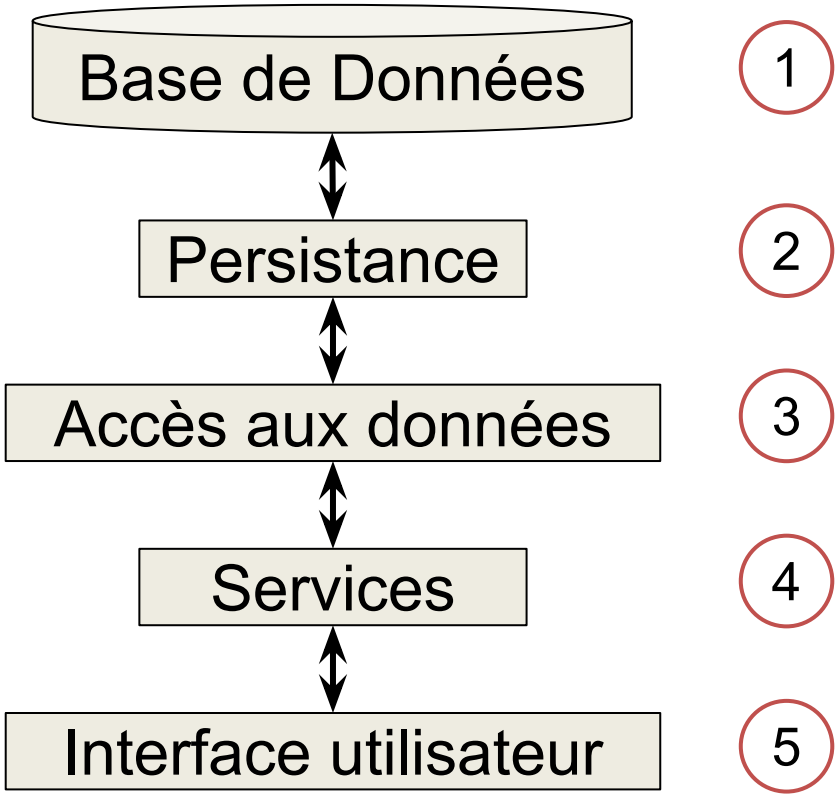
```java
public class User {
    private String name;
    private int age;
    private String email;

    // Constructeur paramétré
    public User(String name, String age, String email) {
        this.name = name;
        this.age = age;
        this.email = email;
    }

    // Getters
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }

    public String getEmail() {
        return email;
    }

    // Setters
    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

Un exemple de modèle de données

- On peut structurer une application web selon un **trois couches** distinctes (**architecture 3-tiers**) :
  - La couche de **données** ⇒ concerne le stockage et les mécanismes d'accès aux données afin de les utiliser au niveau des traitements ;
  - La couche de **traitement** ⇒ concerne les tâches à réaliser sur les données et les traitements suite à une action de l'utilisateur (authentification par exemple) ;
  - La couche de **présentation** ⇒ concerne l'affichage des données et les interactions avec l'utilisateur.

- On peut affiner le découpage précédent :

1. Le **Système de gestion de base de données** ⇒ stockage des données utilisées par l'application.
2. La couche de **persistance** ⇒ gère le mécanisme de sauvegarde et de restauration des données.
3. La couche d'**accès aux données** ⇒ gère la manipulation des données, quelque soit le SGBD utilisé.
4. La couche de **services**, ou couche **métier** ⇒ gère la logique de l'application et les traitements à appliquer aux données.
5. La couche d'**interface utilisateur** ⇒ gère l'affichage des données du service et les actions de l'utilisateur.


Base de Données
1
Persistance
2
Accès aux données
3
Services
4
Interface utilisateur
5

Architecture d’une application web

Couche de persistance

- Il existe de nombreux Systèmes de Gestion de Base de Données (SGBD), et chacun a son mode de communication propre.
- JDBC dispose de nombreux **drivers** lui permettant de **communiquer avec ces SGBD**. Un driver est un composant logiciel (un ensemble de classes Java) qui donne des méthodes à JDBC pour communiquer avec un SGBD précis.

- Pour charger un driver dont la version est <= 3.0 :

```
try {
 Class.forName("com.mysql.jdbc.Driver");
} catch (ClassNotFoundException e) {
   // Gestion des erreurs
}
```

- Les drivers dont la version est >= 4.0 se chargent automatiquement. Il nous suffit d'ajouter le fichier .jar au classpath de notre projet.

- On se connecte à un SGBD à l'aide de son URL et des identifiants de connexion.
  - L'URL se construit selon le modèle suivant : `jdbc:db-type://host-address:port/db-name`
  - On appelle la méthode `getConnection` avec les paramètres

```
Connection connexion = DriverManager.getConnection(url, user, password);
```

  - Lorsqu'on a terminé, il faut **fermer la connexion**

```
connexion.close();
```

Architecture d’une application web

Couche d’accès aux données

Excilys
Développeurs de passion

- On pourrait dès à présent faire des requêtes sur la base de données, à partir de la couche métier.
  Mais cela créerait trop de dépendance entre la couche métier et la couche de persistance.
- On va donc utiliser des éléments intermédiaires pour gérer l'**accès aux données** : les **DAO**.

- Les DAO…
  - Encapsulent la logique liée à la base de données ;
  - Rendent le **code modulable** en séparant l'accès aux données et leur traitement depuis la couche métier ;
  - Suivent généralement le design pattern « **Singleton** » ;
  - Respectent le **CRUD.**

- Le **CRUD** = les 4 opérations de base pour la persistance
  - **Create** : ajout de nouvelles données ;
  - **Read** : lecture de données ;
  - **Update** : modification de données ;
  - **Delete** : suppression de données.

- L'accès aux données via JDBC se fait à l'aide de requêtes appliquées à un objet de classe `Statement`.
- On crée un `Statement` de la façon suivante :

```
Statement statement = connexion.createStatement();
```

- On peut ensuite écrire des requêtes et les exécuter...
  - avec `statement.executeQuery()` pour une requête SELECT ;
  - avec `statement.executeUpdate()` pour les autres requêtes.
- Le résultat d'une requête SELECT est un `ResultSet`. Pour une requête autre, le résultat est un `int`.

- Il est cependant conseillé d'utiliser la classe `PreparedStatement` qui fournit des mécanismes de protection contre les **injections SQL** et d'**optimisation** lorsque les requêtes sont exécutés plusieurs fois.

```
String DELETE_USER_QUERY = "DELETE FROM User WHERE id=?;";
PreparedStatement preparedStatement =
connexion.prepareStatement(DELETE_USER_QUERY);
preparedStatement.setInt(1, user.getId()); // ATTENTION /!\ :
l'indice commence par 1, contrairement aux tableaux
preparedStatement.execute();
```

Utilisation de la classe PreparedStatement

- Lorsqu'on a terminé de traiter une requête, il faut penser à **fermer les objets** `Statement` et `ResultSet` à l'aide de leur méthode `close()`.
- De manière générale, il est conseillé d'utiliser des blocs `try...catch` lorsqu'on fait une requête avec JDBC.
- Vous trouverez beaucoup de détails dans **la Javadoc**.

Couche d'accès aux données

```
public class UserDao {

   private static final String CREATE_USER_QUERY = "INSERT INTO
User(name, age, email) VALUES(?, ?, ?);";
   private static final String FIND_USER_QUERY = "SELECT name,
age, email FROM User WHERE id=?;";

   public void create(User user) throws DaoException {
      try {
         Connection connection = ConnectionManager.getConnection();
         PreparedStatement ps =
            connection.prepareStatement(CREATE_USER_QUERY);

         ps.setString(1, user.getName());
         ps.setInt(2, user.getAge());
         ps.setString(3, user.getEmail());

         ps.execute();
```

```
         ps.close();
         connection.close();
      } catch (SQLException e) {
         throw new DaoException();
      }
   }

   public User findById(int id) throws DaoException {
      // Contenu de la méthode
   }

  // Autres méthodes du CRUD
}
```

Un exemple de Data Access Object (DAO)

Architecture d’une application web

Services

Excilys
Développeurs de passion

Services

- Font le lien entre la couche d’accès aux données (les DAO) et la couche présentation (les contrôleurs).
- Orchestrent les opérations métier non atomiques (appel à plusieurs DAOs par exemple).
- Valident les **contraintes métier** (ex: âge > 18 ans).

Services

```java
public class UserService {

  private UserDao userDao;

  public UserService() {
     this.userDao = new UserDao();
  }

  public void create(User user) throws ServiceException {
     if (user.getAge() < 18) {
         throw new ServiceException("L'utilisateur doit être majeur");
      }

     try {
        userDao.create(user);
     } catch (DaoException e) {
        throw new ServiceException("Une erreur a eu lieu lors de la
création de l'utilisateur");
     }
```

```java
  public User findById(long id) throws ServiceException {
     try {
        User user = userDao.findById(id);

        if (user != null) {
           return user;
        }

        throw new ServiceException("L'utilisateur n°" + id + " n'a pas été
trouvé dans la base de données");
     } catch (DaoException e) {
        throw new ServiceException("Une erreur a eu lieu lors de la
récupération de l'utilisateur");
     }
  }

  // Autres méthodes du CRUD
}
```

Un exemple de service

Architecture d’une application web

Contrôleurs

Contrôleurs

- Points d'entrée de l'application
  - Application console
  - Application JEE servant des JSP
  - Web Service
- Valident les **contraintes techniques** (ex: la chaîne de caractères saisie par l'utilisateur doit représenter un email)

Contrôleurs

```
public class UserController {

  private UserService userService;

  private UserController() {
    this.userService = new UserService();
  }

  public static void main(String[] args) {
    UserController cmd = new UserController();

    String name = cmd.readString();
    int age = cmd.readInt();
    String email = null;

    // Validation technique (la variable représente un email)
    do {
      email = cmd.readString();
    } while (!email.contains("@"));

    User user = new User(name, age, email);
      try {
        cmd.getUserService().create(user);
        System.out.println("L'utilisateur a bien été créé");
      } catch (ServiceException e) {
        System.out.println("Une erreur a eu lieu lors de la création de
l'utilisateur");
      }

  public String readString() {
    Scanner scanner = new Scanner(System.in);

    return scanner.nextLine();
  }

  // Fonction de lecture d'entier

  public UserService getUserService() {
    return userService;
  }
}
```

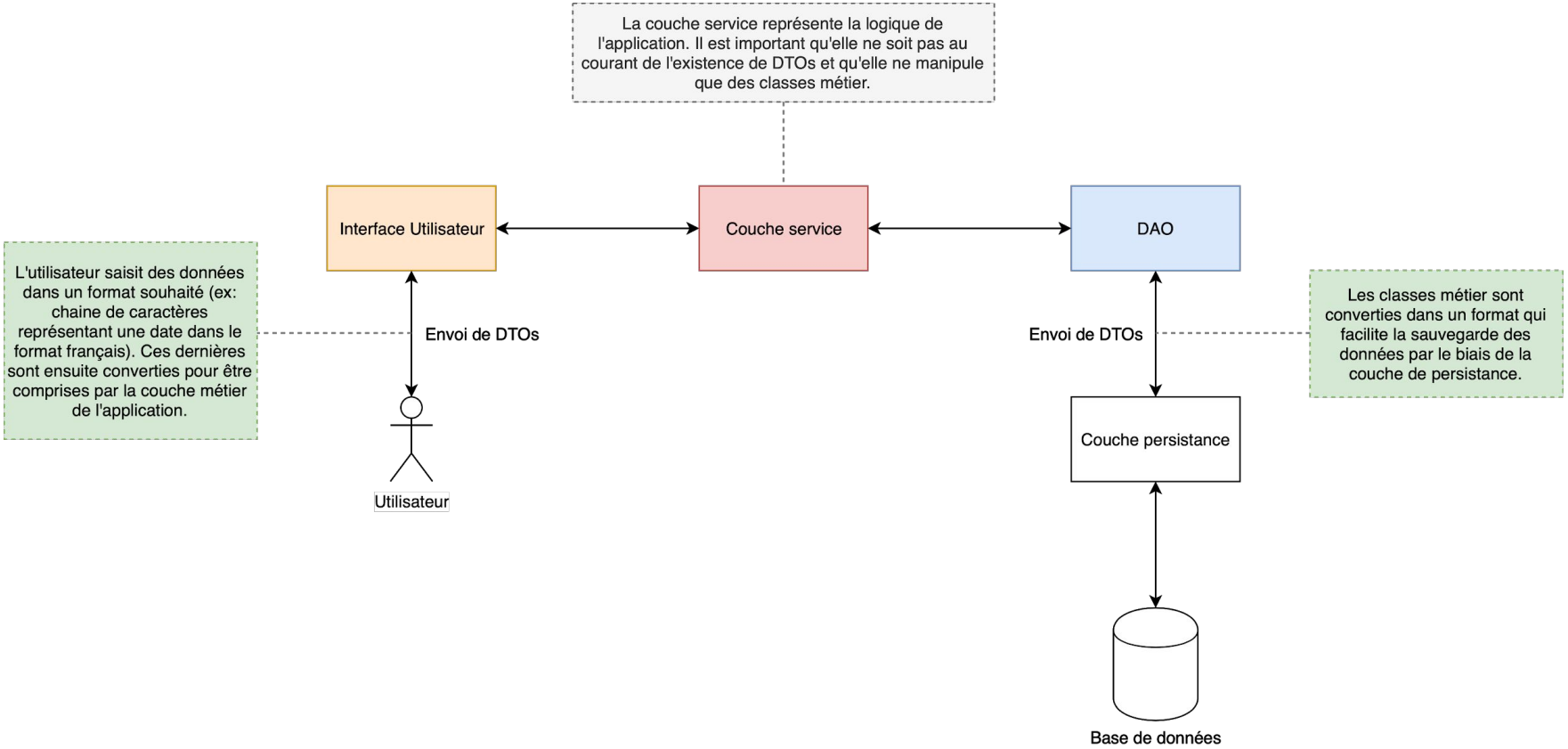
Un exemple de contrôleur

Architecture d’une application web

Data Transfer Objects (DTO)

- Utilisé pour le transfert d'informations entre les différentes couches de l'application (mapping des classes métier vers DTOs et inversement).
- Plain Old Java Object sans logique.
- Découplage du format de transmission de données et du format de persistance, changement d'implémentation plus facile.
- Allègent la quantité de données transférées.

Data Transfer Object (DTO)



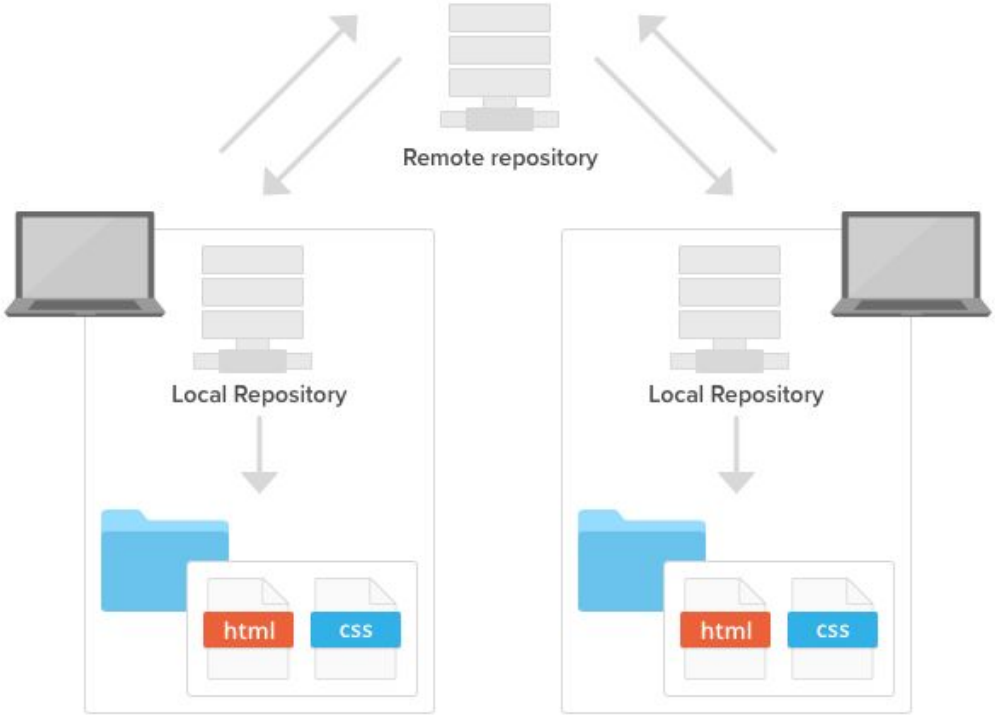
Git

Système de gestion de version

Présentation

- Logiciel de gestion de version de code créé en 2005 par Linus Torvalds
- Suivi de l'avancement du projet
- Travail collaboratif

Répertoire local/distant



Objets manipulés

- blob

  Contenu d'un fichier

- tree

  Arborescence de fichiers

- commit

  État de l'arborescence des fichiers (et de leur contenu) à un moment donné. Les commits sont ordonnés les uns à la suite des autres et sont classés par date.

Objets manipulés

- tag

  Étiquette permettant de retrouver un commit plus facilement (ex: numéro de version)

- branch

  Espace séparé contenant une partie de l'avancement du projet. Des développeurs peuvent travailler sur des branches séparées avant de mettre en commun leur travail

Commandes de base

- git init

  Initialise un dépôt git local

- git remote add [remote-name] [remote-address]

  Ajoute un dépôt distant

- git add [files]

  Ajoute les fichiers spécifiés dans l'espace contenant les fichiers prêts à être versionnés

- git commit -m [commit-message]

  Crée un commit, c'est-à-dire une représentation des changements

Commandes de base

- git branch [branch-name]

  Créée une branche à partir de la branche courante

- git checkout [branch]

  Change de branche

- git push [remote-name] [branch-name]

  Envoie le code de la branche courante sur la branche distante spécifiée

- git pull [remote-name] [branch-name]

  Récupère l'historique des changements de la branche spécifiée depuis le dépôt distant

Commandes de base

- git clone [remote-address]

  Clône le dépôt git distant sur le dépôt local de l'ordinateur

- git merge [branch-name]

  Fusionne le contenu d'une branche sur la branche courante

- git status

  Affiche la liste des modifications depuis le dernier commit (répertoires créés/supprimés/renommés, fichier créés/renommés/supprimés)

Commandes de base

Développeur 1

touch file.txt

git init

git remote add origin https://github.com/excilys/asi311-demo-git.git

git add file.txt

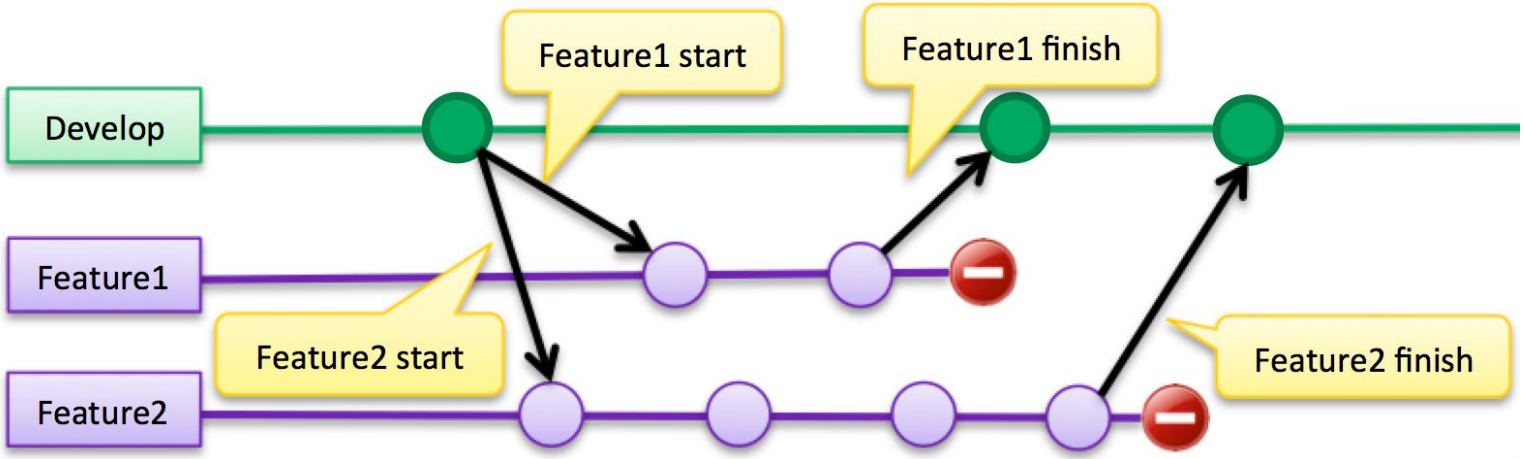
git commit -m ‘initial commit’

git push origin master

Développeur 2

git clone https://github.com/excilys/asi311-demo-git.git

Feature flow



Feature flow

Développeur 1

```
git branch feature/1
git checkout feature/1
touch file1.txt
git add file1.txt
git commit -m ‘[Add] file1.txt’
git push origin feature/1

git checkout develop
git pull
git merge feature/1
```

Développeur 2

```
git branch feature/2
git checkout feature/2
touch file2.txt
git add file2.txt
git commit -m ‘[Add] file2.txt’
git push origin feature/2

git checkout develop
git pull
git merge feature/2
```

HEAD

- Commit le plus avancé de la branche courante

.gitignore

- fichier utilisé pour spécifier la liste des dossiers et des fichiers que git ne doit pas versionner (ex: répertoires contenant les fichiers compilés)

Références

- https://docs.oracle.com/javase/8/docs/api/
- https://www.w3schools.com/sql/sql_quickref.asp
- https://git-scm.com

Excilys
Développeurs de passion