TP .Net

Bug tracking like

Lemettre Arnaud

Version 1.0

10 Pages

02/04/2009

EPITA-MTI2014-NET-TP-bug-track

Propriétés du document

Auteur	**Lemettre Arnaud**
Version	1.0
Nombre de pages	10
Références	EPITA-MTI2014-NET-TP-bug-track

Historique du document

Date de vision	Version	Auteur	Changements
27/02/2009	0.1	Lemettre	création

Site de référence

description	url
Site MTI	
Blog MTI	

Sommaire

Introduction

Le but de cette série de TP sera la réalisation d'un système de bug tracking. Ce tp se décompose en plusieurs parties. Toutes les semaines vous aurez une partie à rendre. Cette partie sera évaluée. Toute fois si pour une raison quelconque vous n'avez pas réussi à faire convenablement cette partie une correction sera postée une journée avant le début du prochain TP. Vous aurez à charge de l'intégrer dans votre code si celui ne produit pas le fonctionnement attendu.

Ce travail est à faire individuellement, tout code similaire sur deux personnes sera considéré comme un travail non rendu et non négociable.

Durant cette série de TP les notions suivantes seront abordées :

- modélisation BDD
- création de script SQL (Base, tables, données)
- création procédures stockées
- Entities Framework
- Linq To SQL
- Manipulation Open XML
- Génération de Flux XML
- Dynamic Data Entities
- Membership provider / Role provider
- utilisation des pages aspx et composants
- WCF (Webservice et sécurisation)

Chaque partie à rendre ne demande pas plus de 3h de travail par semaine, Bonne chance ;)

Contexte

Un système de suivi de problèmes (de l'anglais bug tracking system ou système de suivi de bogues) est une application informatique qui permet d'aider développeurs et utilisateurs à améliorer la qualité d'un logiciel. Les utilisateurs soumettent les bogues (défauts de fonctionnement) rencontrés dans le logiciel. Les développeurs sont alors toujours au fait des problèmes rencontrés.

L'ensemble de la solution devra utiliser SQL Server 2012 pour la base de données, et le Framework 4.5 de .Net.

Le rendu de chaque lot, correspond au rendu de chaque partie.

Partie 1

Renseignements

Vous devez coder les bases du serveur de tracking. Il vous faut donc commencer par réaliser la base de données. Comme vous le savez déjà votre serveur devra donc gérer les bugs d'un projet. Nous allons voir un peu plus grand, effectivement un développeur peut avoir plusieurs projets. Le type de bugs que peut lui soumettre un utilisateur peut être de différents types (critique, mineur, typographique). Un bug peut par la suite avoir un statut (refusé, accepté, terminé). Pour favoriser, la collaboration on peut ajouter des commentaires aux bugs.

Travail à Faire

A partir des informations fournies ci-dessus vous devez modéliser votre base de données.

Pour se faire vous devez utiliser Sql ServerManagement Studio.

Le nom de la base de données sera : BugTrackLike

Il devra y avoir 5 tables.

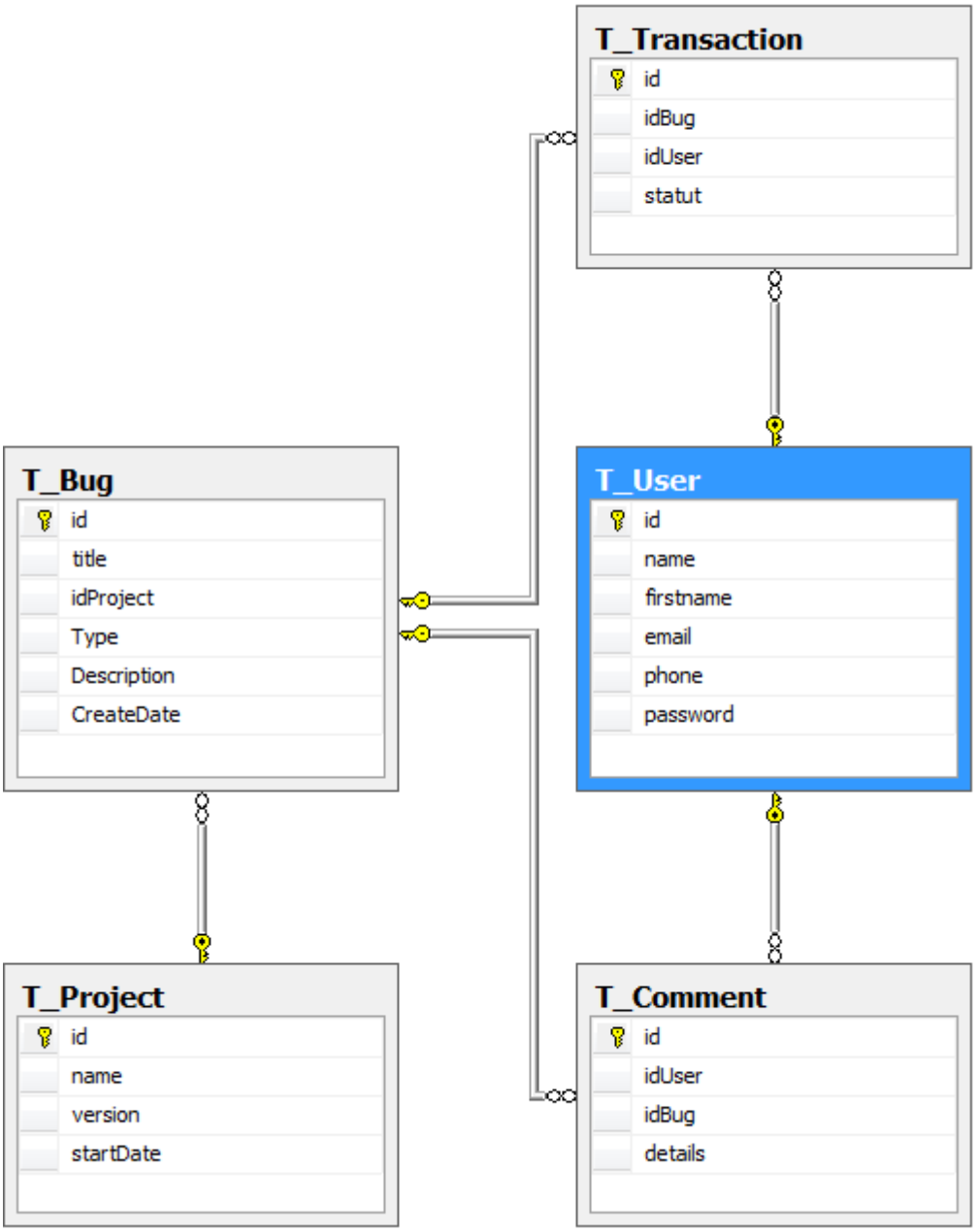

Figure 1 - Exemple de schéma

Dans cette partie vous devez utiliser Entity Framework. Pour cela créer un nouveau projet de type ASP.NET MVC 3.0 avec pour nom login_lBugTrack où login_l correspond à votre login EPITA.

Puis réaliser les différentes fonctionnalités :

- création d'un utilisateur (procédure stockée) devra se trouver dans le schéma DBO et se nommer CreateUser
- suppression d'un utilisateur (procédure stockée) devra se trouver dans le schéma DBO et se nommer DeleteUser
- mise à jour d'un utilisateur (procédure stockée) devra se trouver dans le schéma DBO et se nommer UpdateUser
- le reste des procédures pour les autres tables peuvent être faite directement avec Entity Framework

Pour cela créer une classe par entité le nom de la classe devra être le même que celui de la table sans le « T_ ». Chacune des classes devra également comporter une méthode nommée CreateENTITY, DeleteENTITY, UpdateENTITY où ENTITY correspondra au nom de la classe.

Ainsi nous aurons :

Bug.cs

- permet la création d'un bug `bool CreateBug(T_Bug bug, long idProject)`
- permet de supprimer un bug `bool DeleteBug(long id)`
- permet de mettre à jour le bug `bool UpdateBug(T_Bug bug)`
- permet de récupérer un bug `T_Bug GetBug(long id)`
- retourne la liste complète de tous les bugs `List<T_Bug> GetListBug()`

Comment.cs

- permet la création d'un comment
  `bool CreateComment(T_Comment comment, long idUser, long idBug)`
- permet de supprimer un commentaire `bool DeleteComment(long id)`
- permet de mettre à jour le commentaire `bool UpdateComment(T_Comment comment)`
- permet de récupérer un commentaire `T_Comment GetComment(long id)`
- retourne la liste complète de tous les comments
  `List<T_Comment> GetListComment()`
- retourne la liste complète de tous les comments
  `List<T_Comment> GetListCommentByIdBug(long idBug)`

Project.cs

- permet la création d'un projet `bool CreateProject(T_Project project)`
- permet de supprimer un projet `bool DeleteProject(long id)`
- permet de mettre à jour le projet `bool UpdateProject(T_Project project)`
- permet de récupérer un projet `T_Project GetProject(long id)`
- retourne la liste complète de tous les projets `List<T_Project> GetListProject()`

Transaction.cs

- permet la création d'une transaction
  `bool CreateTransaction(T_Transaction Transaction, long idBug, long idUser)`
- permet de supprimer une transaction `bool DeleteTransaction(long id)`
- permet de mettre à jour la transaction `bool UpdateTransaction(T_Transaction Transaction)`
- permet de récupérer une transaction `T_Transaction GetTransaction(long id)`
- retourne la liste complète de tous les transactions `List<T_Transaction> GetListTransaction()`

User.cs

- permet la création d'un utilisateur `bool CreateUser(T_User user)`
- permet de supprimer un utilisateur `bool DeleteUser(long id)`
- permet de mettre à jour l'utilisateur `bool UpdateUser(T_User user)`
- permet de récupérer un utilisateur `T_User GetUser(long id)`
- retourne la liste complete de tous les users `List<T_User> GetListUser()`

Pour l'edmx nommé le *modele.edmx,* il devra avoir l'implémentation suivante :

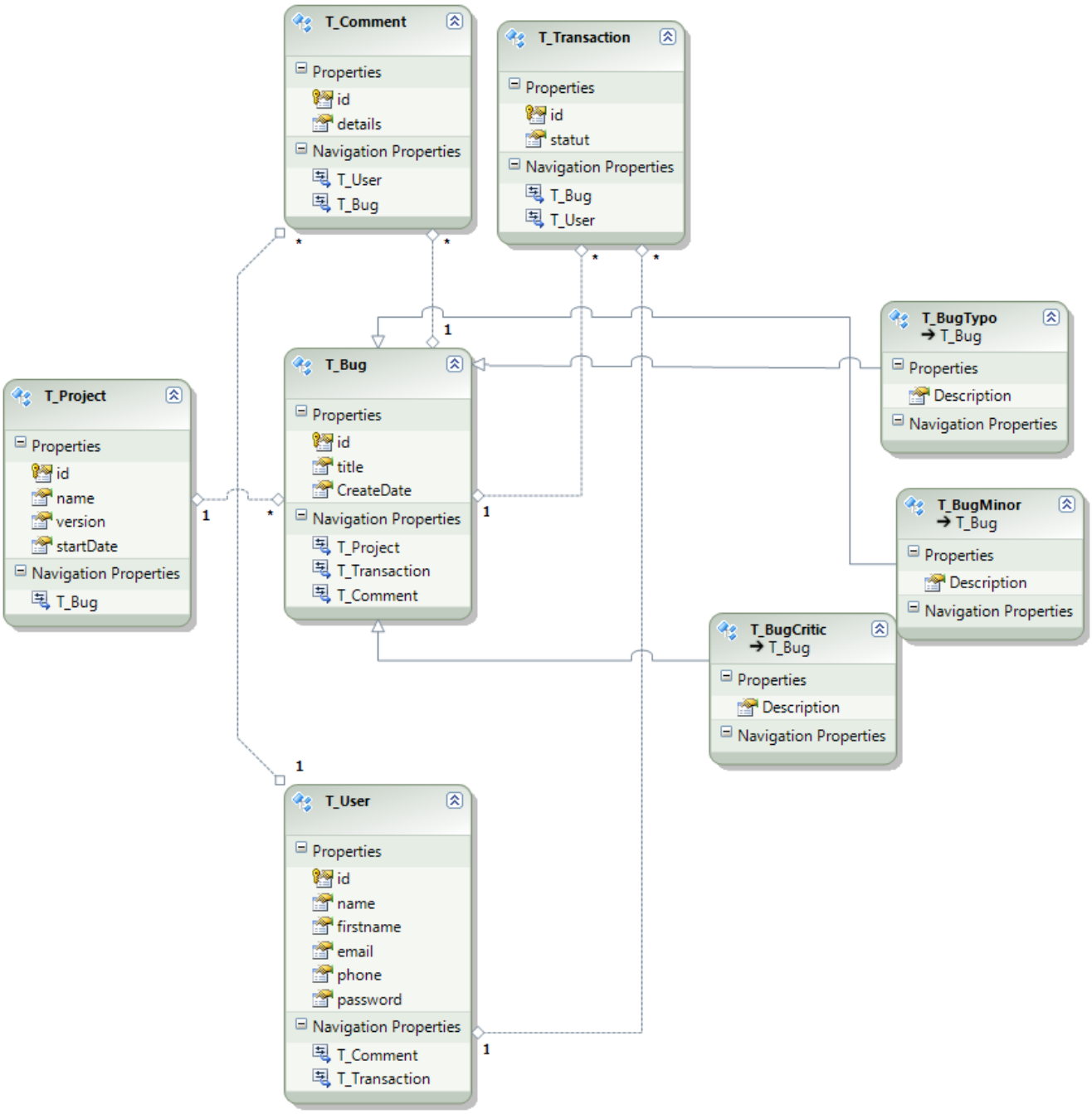

Figure 2 - Exemple de l'edmx

- Vous devez être capable de remonter des informations jusqu'à la couche interface. Penser aux TP précédents.

Pour cela créer une couche businessManagement comportant également le nom des différentes classes de la dataAccess (mais pas dans le même namespace) et qui devront donc faire appel aux méthodes de la DataAccess.

Exemple de l'architecture :

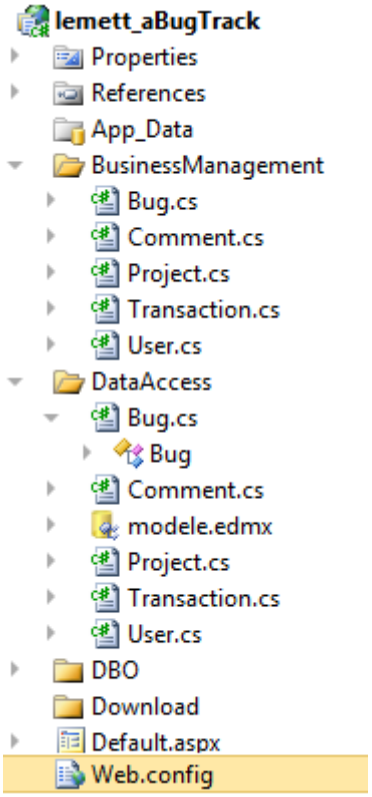

Figure 3 - Détails de l'architecture

Modalité de rendu

Les fichiers seront à rendre dans une tarball ayant pour nom :

login_l.zip

Cette tarball devra comprendre :

- un script de création de votre base ainsi que des tables.
  Nom : CreateBugTrack.sql
- Un script pour remplir votre base avec des données
  Nom : DataBugTrack.sql
- Un dossier contenant la solution Visual Studio qui devra compiler.
  Nom : login_lBugTrack

Le tout à envoyer sur l'adresse mti.rendu.dotnet@gmail.com avec les balises suivantes :

[MTI2014][NET][login_l][BugTrack] partie 1