

1 INTRODUCTION AU PROJET

L'entreprise Casual Games Everywhere GmbH (CGE GmbH), dont le siège est à Bochum, souhaite élargir son offre de jeux occasionnels jouables en ligne. D'une part, CGE GmbH agit en tant que partenaire de fournisseurs de services Internet et met à leur disposition son assortiment de jeux occasionnels contre paiement, en tant qu'instrument de marketing. D'autre part, CGE distribue ses jeux en ligne via différents portails de jeux, également en propre régie.

Les jeux occasionnels permettent aux fournisseurs de services Internet, d'une part, d'augmenter le temps passé par les internautes sur leurs sites et, d'autre part, d'augmenter les revenus publicitaires. d'augmenter la fréquentation des sites web et donc de générer des revenus publicitaires. et, d'autre part, de fidéliser davantage leurs clients.

Dans le cadre de différents projets de développement de jeux casual, il s'est avéré que en raison des nombreux bugs (erreurs) signalés, un système de suivi des bugs basé sur une base de données est nécessaire.

Un système de suivi des bugs (BTS) est un outil de développement de logiciels qui permet d'enregistrer et de documenter les erreurs de produits/programmes. illustrer le processus. Un utilisateur (joueur, gamer) signale à l'entreprise de développement du jeu un bug qui est apparu pendant qu'il jouait.



Abb. 1: Bug im Spiel **The Witcher**: Schwebendes Pferd.

Un membre de l'équipe de test du logiciel poste le bug signalé à l'intérieur de l'entreprise. Une brève analyse des captures d'écran soumises par le joueur aboutit à la création d'un nouveau bug. le bug signalé est créé en tant que bug valide (réel) avec le statut new.

Sur la base d'une enquête plus approfondie, le degré de gravité du bug est évalué à Critical (critique) et sa priorité est déclarée Urgent (urgent), c'est-à-dire qu'il doit être corrigé. doit être corrigé dès le prochain build.

En outre, le bug est associé au produit correspondant (ici : The Witcher). et l'associe à des mots-clés appropriés.

La correction du bug est ensuite confiée à un membre de l'équipe de développement. ce qui confère au bug le nouveau statut assigné. La communication entre les testeurs et les développeurs est documentée en permanence.

Après que le développeur responsable a élaboré une solution pour corriger le bug, il considère que le bug est terminé / corrigé.

Un membre de l'équipe de test examine ensuite la solution élaborée et vérifie si le bug a été résolu. Le bug a effectivement pu être entièrement corrigé. Si c'est le cas, le bug reçoit le statut Closed / verified

Si ce n'est pas le cas, il est renvoyé au développeur responsable pour être traité.

Le bug passe alors au statut reassigned.

Le bug est alors classé comme "non résolu".

La délimitation ou la localisation d'une erreur se fait généralement par une série de questions et de réponses entre les développeurs et les utilisateurs. de questions et de réponses entre les utilisateurs, les développeurs et les testeurs. Leur La documentation de la communication réciproque est l'une des tâches centrales. d'un système de suivi des bugs.

Après une longue phase de prise de décision, la direction a décidé d'investir dans la gestion des bugs. la direction de la CGE a décidé de réaliser elle-même le système de suivi des bugs. de réaliser ce projet.

2 MISSION & EXIGENCES

L'objectif du projet proposé Bug Tracking System (BTS) est de réaliser un système de suivi de bugs basé sur une base de données et utilisable via un client shell. peut être utilisé.

Un système de suivi des bugs (BTS) est un outil de développement de logiciels qui sert à saisir et à documenter les erreurs de produit / de programme.

La délimitation ou la localisation d'une erreur se fait généralement par une série de questions et de réponses entre les utilisateurs et les développeurs. de questions et de réponses entre les utilisateurs et les développeurs. Cette communication réciproque

entre les utilisateurs et les développeurs est l'une des tâches les plus importantes. tâche centrale d'un système de suivi des bugs. Voici quelques questions typiques qui peuvent être résolues à l'aide d'un système de suivi des bugs.

de suivi doivent être abordés :

Quels sont les problèmes rencontrés ?

Quelle est la nature des problèmes ?

Comment le problème a-t-il été documenté ?
Qui a signalé le problème ?
Quel est le développeur responsable du problème ?
Qui a vérifié le problème en tant que tel ?
Quel produit est concerné par le problème ?
Quelles versions du produit sont concernées ?
Qu'est-ce qui a déjà été entrepris pour résoudre le problème ?
Quels efforts seront probablement nécessaires pour résoudre le problème ?
Quel a été l'effort réel ?
Afficher l'historique complet d'un bug

En raison des nombreuses possibilités d'analyse requises, les systèmes de suivi des bugs sont conçus sont généralement conçus comme des systèmes basés sur une base de données.

Les systèmes de suivi de bugs typiques soutiennent le concept du cycle de vie d'un bug. (bug life cycle), qui peut être suivi par le biais du statut d'un bug. Pour ce faire, il est bien entendu nécessaire que le statut d'un bogue au sein d'un système de gestion des bogues (BTS) puisse être suivi. Les personnes autorisées doivent pouvoir modifier un bug selon un schéma prédéfini. peut être modifiée. Sur la base d'une première analyse interne du modèle de données du BTS, les exigences suivantes sont apparues. les exigences suivantes concernant les entités de base (relations / tables) :

Nr	Klasse	Attribute (vorläufig)
1	Bug	bug_id {PK} , date_reported , summary , description, resolution , priority , hours
2	BugHistory	bug_history_id {PK}, all bug attributes , change_by, date_changed, change_action
3	BugPriority	priority_id {PK} , priority_name
4	BugStatus	status_id {PK} , status_name
5	BugSeverity	severity_id {PK} , severity_name, description
6	Tag	tag_id {PK} , tag
7	Comment	comment_id {PK} , author , comment_date , comment
8	Account	account_id {PK} , account_name , first_name , last_name, email, password_hash , portrait_image , hourly_rate , account_type , account_role
9	Product	product_id {PK} , product_name, description
10	Screenshot	image_id {FK} , screenshot_image , caption
11	AccountType	type_id {PK} , type_name
12	AccountRole	role_id {PK}, role_name

«entity» BugPriority
priority_id {PK} priority_name

«entity» BugStatus
status_id {PK} status_name

«entity» Comment
comment_id {PK} author comment_date comment

«entity» Tag
tag_id {PK} tag description

«entity» Bug
bug_id {PK} date_reported summary description resolution priority hours

«entity» Account
account_id {PK} account_name first_name last_name email password_hash portrait_image account_type account_role hourly_rate

«entity» Screenshot
image_id {FK} screenshot_image caption

«entity» BugSeverity
severity_id {PK} severity_name description

«entity» BugHistory
bug_id {PK} date_reported summary description resolution priority hours changed_By date_changed change_action

«entity» AccountType
type_id {PK} type_name description

«entity» Product
product_id {PK} product_name description

«entity» AccountRole
role_id {PK} role_name

En outre, il existe les associations suivantes entre les classes

Un seul BugStatus est attribué à chaque bug. Pour chaque BugStatus, il peut y avoir

plusieurs Bugs.

Une seule BugPriority est attribuée à chaque Bug. Pour chaque BugPriority, il peut y avoir

plusieurs Bugs.

Une seule BugSeverity est attribuée à chaque Bug. Pour chaque BugSeverity, il peut y avoir

il y a plusieurs Bugs.

Chaque bug peut se voir attribuer un nombre quelconque de tags et vice versa.

inversement, c'est-à-dire que chaque balise peut se référer à un nombre

quelconque de bugs

se rapporte à un tag.

Chaque bug peut être associé à un nombre quelconque de commentaires, alors que chaque

Comment doit être associé à un seul bug.

Un commentaire peut référencer un nombre quelconque d'autres commentaires.

Il existe donc une association réflexive (0..n, 0...n) entre les commentaires.

Chaque Bug peut se référer à un nombre quelconque de Products et ainsi de suite.

et inversement, c'est-à-dire que chaque produit peut contenir un nombre quelconque de bugs.

A des fins de documentation, plusieurs captures d'écran peuvent être attribuées à un bug.

peuvent être utilisées. En revanche, chaque capture d'écran doit correspondre à un seul bug.

Un produit peut être géré par plusieurs comptes (utilisateurs). Ceci est valable pour

l'inverse est également vrai, c'est-à-dire qu'un compte peut gérer plusieurs produits.

Un Account (utilisateur) appartient à un seul AccountType. Un AccountType peut comprendre un nombre quelconque de comptes.

Pour chaque bug, un historique des bugs (BugHistory) est tenu, dans lequel toutes les modifications sont consignées.

d'un bug sont consignées. Pour chaque bug, il peut donc y avoir autant d'entrées que souhaité.

entrées dans l'historique des bugs, et inversement, chaque entrée de l'historique des bugs

doit être associé à un bug précis.

Chaque entrée dans le BugHistory doit correspondre à un seul utilisateur (Account).

mais chaque utilisateur peut faire autant d'entrées dans le BugHistory qu'il le souhaite.

peut effectuer.

Il existe trois associations différentes entre les bugs et les comptes, à savoir reported_by, assigned_to et verified_by. Celles-ci indiquent ,

- o qui a signalé un bug,

- o à qui un bug a été attribué pour traitement,

- o qui a examiné / vérifié un bug.

Les exigences suivantes s'appliquent aux trois types de relations :

- o Reported_by : chaque utilisateur (comptes) peut signaler plusieurs bugs, un bug étant lié à un seul utilisateur (compte) qui le signale.

est associé.

- o Assigned_to : chaque utilisateur (compte) peut avoir plusieurs bugs à traiter.

être attribués pour un traitement ultérieur. Un bug doit cependant être traité par un seul

utilisateur (Accounts) doit être traité.

- o Verified_by : chaque utilisateur (Accounts) peut vérifier plusieurs bugs,

c'est-à-dire être classé comme corrigé, un bug donné ne pouvant être traité que par
doit avoir été vérifié par un seul utilisateur (compte).

Les commentaires (= questions et / ou réponses) déposés sur les bugs doivent être suivis dans leur intégralité.
peuvent être suivis dans leur historique complet. Cela signifie qu'en règle générale, des
des fils de commentaires doivent être suivis (voir aussi l'exemple de l'illustration 3).

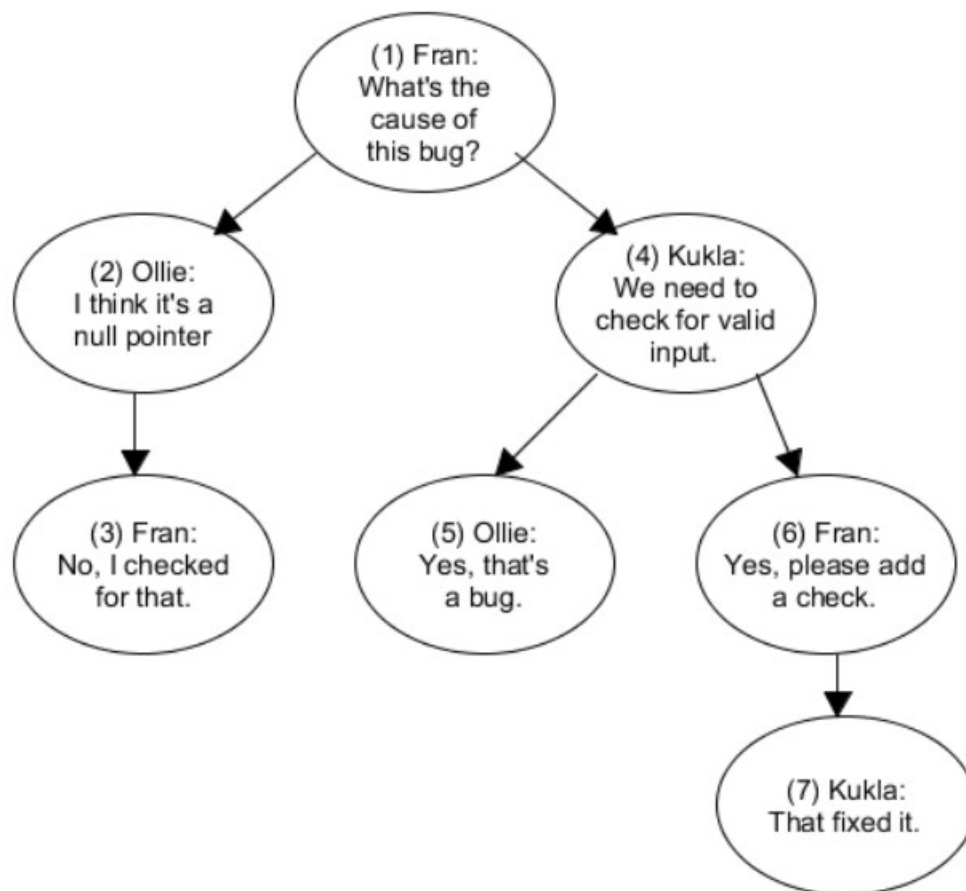


Fig. 3 : Exemple de commentaires filtrés.

Il peut y avoir plusieurs interlocuteurs pour les différents produits (Products).
(Accounts) et vice versa. Cela signifie qu'un interlocuteur (= compte) peut être
responsable de plusieurs produits.
peut être responsable de différents produits, et ce dans différents rôles.

2.1 Le cycle de vie d'un bug (Bug Life Cycle)

Dans le cadre des tests logiciels, on utilise le paradigme du cycle de vie des bugs, c'est-à-dire que l'on suppose qu'un bug possède son propre cycle de vie. C'est pourquoi les bugs à traiter au sein d'un système de suivi des bugs peuvent être caractérisés par différents états.

Le diagramme de transition d'état (State Chart) suivant montre les différents états d'un bug, états d'un bogue au sein du système de suivi des bogues et leurs liens ou leurs

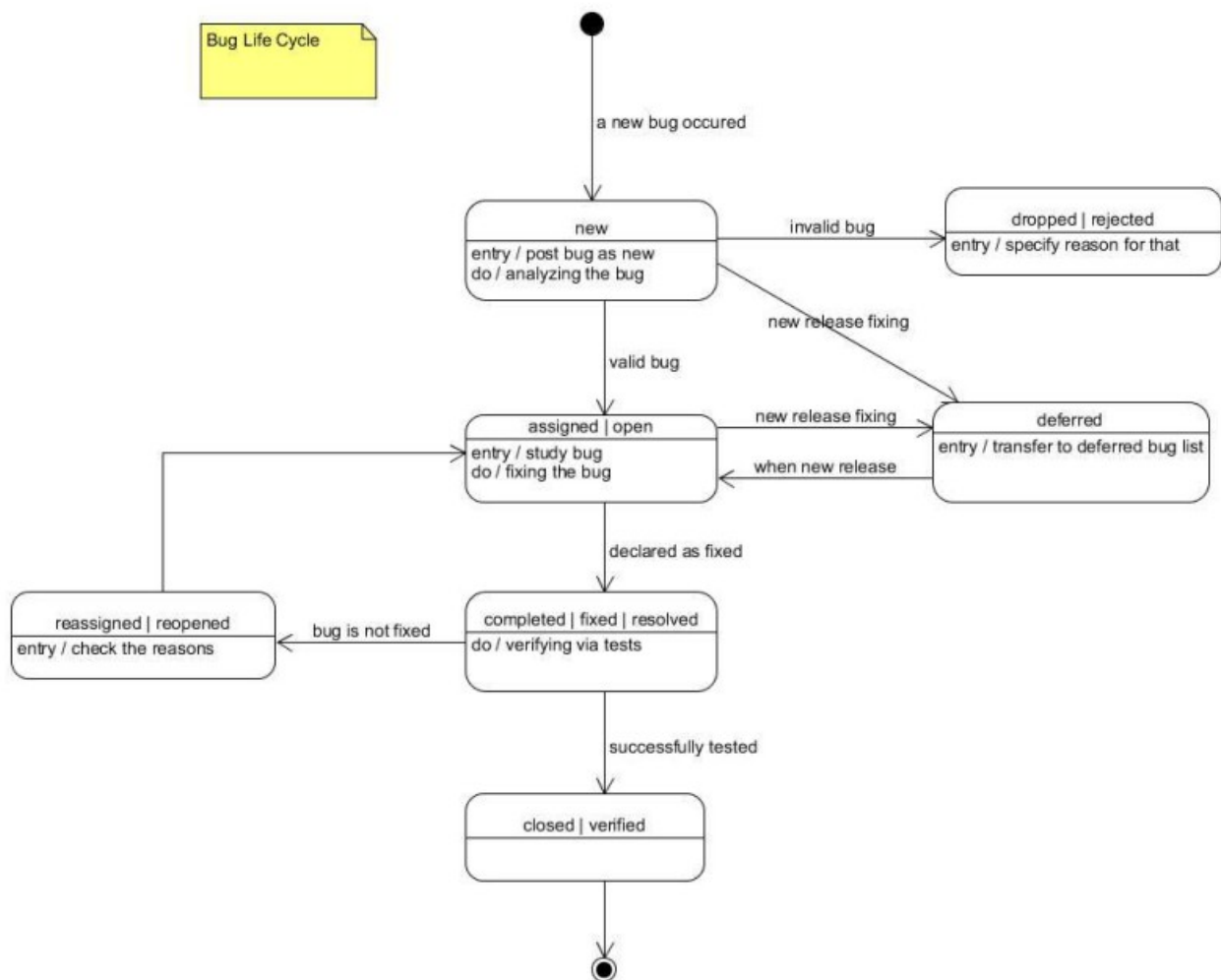


Fig. 4 : Diagramme d'état pour le cycle de vie des bugs.

On part du principe que les états suivants existent :

new : Un nouveau bug a été trouvé. Le bug est d'abord analysé.

dropped / rejected : le nouveau bug a été déclaré invalide ou rejeté en tant que bug.

rejeté.

assigned / open : un bug valide a été attribué à un développeur pour être corrigé.

deferred : la correction d'un bug valide a été reportée à la prochaine version.

reportée à une date ultérieure.

completed / fixed / resolved : Le développeur responsable du bug a déclaré le bug comme étant

corrigé, a été déclaré corrigé. L'équipe de test vérifie si le bug a effectivement été éliminé.

reassigned / reopened : l'équipe de test a constaté que le bug a été soit totalement, soit partiellement corrigé.

n'a pas été corrigé ou ne l'a été que partiellement. Le bug est renvoyé au développeur responsable.

développeur.

closed / verified : L'équipe de test a vérifié que le bug a été résolu.

a été résolu avec succès. Le bug est donc considéré comme résolu et le cas peut être clôturé.

peut être clôturé.

Les transitions possibles sont représentées par des flèches entre les états d'état (voir aussi figure 4).

Une transition d'état nécessite en règle générale un événement déclencheur (trigger).

En général, les déclencheurs sont notés sur la flèche de transition.

Exemple (voir aussi figure 4) :

Pour qu'un nouveau bug puisse passer de l'état new à l'état assigned / open. open, il doit être spécifié au préalable comme un bug valide, c'est-à-dire auparavant, l'événement valid bug s'est produit.

2.2 Informations détaillées sur la classification des bugs

2.2.1 Statut d'un bug (Bug Status) :

Le statut actuel d'un bug (Bug Status) est déterminé par le cycle de vie d'un bug (Bug Life Cycle).

(bug life cycle) et est donc défini par les niveaux suivants :

new : Un nouveau bug a été trouvé. Le bug est d'abord analysé.

dropped / rejected : le nouveau bug a été déclaré invalide ou rejeté en tant que bug.

rejeté.

assigned / open : un bug valide a été attribué à un développeur pour être corrigé.

deferred : la correction d'un bug valide a été reportée à la prochaine version. reportée à une date ultérieure.

completed / fixed / resolved : Le développeur responsable du bug a déclaré le bug comme étant

corrigé, a été déclaré corrigé. L'équipe de test vérifie si le bug a effectivement été éliminé.

reassigned / reopened : l'équipe de test a constaté que le bug a été soit totalement, soit partiellement corrigé.

n'a pas été corrigé ou ne l'a été que partiellement. Le bug est renvoyé au développeur responsable.

développeur.

closed / verified : L'équipe de test a vérifié que le bug a été résolu.

a été résolu avec succès. Le bug est donc considéré comme résolu et le cas peut être clôturé.

2.2.2 Priorité d'un bug (Bug Priority) :

La priorité d'un bug (Bug Priority) est définie par les niveaux suivants :

Urgent : doit avoir été résolu dans le build suivant.

High : doit être résolu dans l'un des builds suivants, mais au plus tard pour la release.

Medium : peut être résolu après la release ou dans la prochaine release.

Low : peut être épuré, mais pas nécessairement

2.2.3 Degré de gravité d'un bug (Bug Severity) :

La gravité d'un bug (Bug Severity) est définie par les niveaux suivants :

Critical : le bug affecte des fonctions ou des données critiques du système. Il existe

pas de solution de contournement. Exemple : Installation non réussie, Absence totale d'une fonctionnalité.

Major : le bug affecte des fonctions ou des données importantes. Il existe une solution de contournement, mais elle est trop opaque pour être une véritable solution.

ne peut pas être utilisée. Exemple :

Mineure : le bug n'affecte que des fonctionnalités secondaires ou des données non critiques.

données. Il existe une solution de travail facile à mettre en œuvre.

Trivial : le bug n'affecte aucune fonctionnalité ou donnée. Un workaround est nécessaire

n'est pas nécessaire. La productivité ou l'efficacité du système n'est pas réduite.

Exemples : le bug n'a pas été corrigé : Mise en page inesthétique, fautes d'orthographe / erreurs grammaticales.

2.2.4 Marquage des bugs (tags) :

Pour le marquage des bugs, voir la liste des tags à la page suivante.


de la liste des mots-clés (tags).

Tag	Description
alert	For critical bugs requiring immediate attention. Triggers IRC notification.
ci	A Bug affecting the Continuous Integration system
config-agent	A bug affecting os-collect-config, os-refresh-config, os-apply-config.
containers	A bug affecting container based deployments.
documentation	A bug that is specific to documentation issues.
il8n	A bug related to internationalization issues.
low-hanging-fruit	A good starter bug for newcomers.
networking	A bug that is specific to networking issues.
promotion-blocker	Bug that is blocking promotion job(s).
quickstart	A bug affecting quickstart.
selinux	A bug related to SELinux.
pyclient	A bug affecting python-client.
ui	A bug affecting the user interface.
upgrade	A bug affecting upgrades.
validations	A bug affecting the Validations.
apport-bug	A bug reported using 'Report a Problem' in an applications Help menu contains lots of details.
apport-crash	A crash reported by apport - Ubuntu's automated problem reporter.
package-conflict	A bug reported by apport when a package operation failed due to a conflict with a file provided by another package.
desktop-file	The bug requests the addition/fix of a .desktop file.
fix-to-verify	A bug that is Fix Released and should be verified when performing iso testing of daily builds or milestones.
ftbfs	Bugs describing build failures of packages.
hw-specific	A bug that requires a specific piece of hardware to duplicate.
iso-testing	A bug found when performing iso testing.
likely-dup	The bug is likely a duplicate of another bug ((maybe an upstream bug too) but you can't find it.
manpage	This bug is about a package's manpage being incorrect.
metabug	This bug has a high probability of duplicate reports being filed.
desktop-icons	Bugs related to the desktop, especially the alignment, display and grid of icons.
needs-reassignment	A bug that was reported about the wrong package but the package maintainer isn't sure which package it belongs to.

packaging	The bug is likely to be a packaging mistake.
screencast	This bug report includes a screencast of the bug in action.

02.05.2022	Lastenheft	Version: 2.0
------------	------------	--------------

P_BTS_Lastenheft_02_rev1.doc

	Datenbank-Projekt Bug Tracking System (BTS)	SS 2022
		Seite: 16/21

string-fix	This bug is a string fix (not code) and is great for new contributors. For spelling and grammatical errors.
unmetdeps	Bugs that indicate packages not being installable due to missing dependencies.
upgrade-software-version	Bugs that request new software versions - please help reviewing them carefully.
work-intensive	Triaging requires intensive work to validate/reproduce.
dist-upgrade	A bug that was encountered when upgrading between releases of specified product.
testcase	A bug which contains a test case - steps to recreate the bug.

2.3 Droits des utilisateurs basés sur les rôles (Account Role) :

Pour le SST, il faut s'assurer que seuls les utilisateurs/utilisateurs autorisés peuvent apporter des modifications
peuvent effectuer des modifications.

Pour ce faire, un concept de droits d'accès basé sur les rôles (AccountRole) s'impose.

droits d'accès à tous les objets de la base de données.

Le rôle d'un utilisateur (AccountRole) est défini par les entrées suivantes :

BTS_Admin : fait office d'administrateur du BTS et possède tous les droits.

BTS_Simple_User : ne peut accéder à tous les objets BTS qu'en lecture seule.

BTS_Advanced_User : ne peut généralement accéder qu'en lecture seule aux objets BTS. Un accès

accès en écriture n'est autorisé que sur les objets BTS suivants :

- o Bug, Captures d'écran, Comment.

2.4 Types d'utilisateurs (Account Type) :

Certains types d'utilisateurs (Account Type) doivent être créés pour le SST.

Pour l'instant, les types d'utilisateurs suivants sont supposés exister :

Product Manager	Software Engineer	Software Tester
Artist / Illustrator	Programmer	Community Manager
Game Designer	Administrator	°Individual (private person)

2.5 Cas d'utilisation du système de suivi des bugs

Une analyse préliminaire des cas d'utilisation a permis d'identifier les exigences fonctionnelles suivantes :

Enregistrer les collaborateurs / utilisateurs

- o pour les collaborateurs, le taux horaire doit être indiqué

- o pour les utilisateurs, ceci est obsolète

- o télécharger la photo de l'employé / l'image

Fonctions liées aux bugs :

- o Signaler un bug (par le collaborateur / l'utilisateur)

- o Attribuer le bug à l'employé

- o Vérification du bug par l'employé

- o Définir / modifier le statut du bug

- o Définir / modifier la priorité du bug

- o Définir / modifier la gravité du bug

- o Associer un bug à un produit

- o Attribuer des tags de bug

- o Télécharger des captures d'écran et les enregistrer dans la base de données

- o Suivre / consulter les fils de commentaires

- o Saisir des commentaires

- o Afficher l'historique des bugs (complet ou partiel)

fonctions liées au produit :

- o Produit CRUD

- o Inscrire les personnes de contact pour les produits

CRUD pour les tableaux : BugStatus, BugPriority, BugSeverity , Products, Accounts,

Captures d'écran, Tags,

CRUD : Create, Read, Update, Delete.

Requêtes liées à l'application à mettre en œuvre :

- o Quels sont les problèmes rencontrés ?

- o Quelle est la nature des problèmes ?

- o Comment le problème a-t-il été documenté ?

- o Qui a signalé le problème ?

- o Quel est le développeur responsable du problème ?
- o Qui a vérifié le problème en tant que tel ?
- o Quel produit est concerné par le problème ?
- o Quel produit présente quel problème ?
- o Quelles mesures ont déjà été prises pour résoudre le problème ?
- o Quel a été le coût réel de la résolution du problème ?
- o Quels collaborateurs ont été impliqués dans la résolution du problème ?

2.6 Autres exigences

Le SST doit être une application basée sur une base de données selon le principe client-serveur.

L'objectif est de mettre en œuvre les fonctionnalités de base du SST sans recourir à un site web ou à un site Web.

être mise en œuvre autant que possible sans clients web basés sur HTML ni PHP côté serveur.

Cela signifie que les utilisateurs peuvent envoyer des demandes à l'application serveur via des clients shell.

dont le traitement s'effectue principalement par des commandes exécutées sur le serveur.

Les routines stockées sont exécutées sur le serveur.

Pour ce faire, les cas d'utilisation du BTS ont été définis à l'aide de PL/SQL, le langage de programmation procédural de MySQL, sous la forme de déclencheurs, de procédures stockées ou de fonctions stockées, le cas échéant en combinaison avec des transactions. Comme

Les déclencheurs / procédures stockées sont des objets de base de données directement stockés dans le SGBDR (MySQL).

sont stockés, ils peuvent être appelés via le client MySQL basé sur Shell.

afin de tester des cas d'utilisation individuels.

Comme il s'agit d'une application basée sur une base de données, l'intégrité référentielle et l'intégrité des données doivent être garanties.

L'intégrité des données doit être assurée par le SGBDR (MySQL) utilisé.

doivent être garantis. Pour ce faire, MySQL doit utiliser le moteur de stockage transactionnel InnoDB

doit être utilisé pour les tables. Le cas échéant, il faut également utiliser des procédures stockées, des fonctions User

Defined Functions et différentes procédures de déclenchement.

Pour répondre aux différentes questions posées au SST, il convient d'utiliser des SQL adaptés à chaque question.

des vues SQL adaptées aux différentes questions, qui servent de base de données pour celles-ci

servent de base de données.

Les opérations CRUD (Create, Read, Update, Delete) pour les tables de base doivent être effectuées à l'aide de
doivent être effectuées par des Stored Procedures spécialement conçues à cet effet.

Les captures d'écran doivent pouvoir être téléchargées par l'utilisateur et être ensuite enregistrées dans les tables de base de données.

être stockées dans la table de base de données Screenshots. Pour cela, la fonction MySQL

LOAD_FILE(...) peut être utilisée.

Pour le SST, un concept de sécurité de base de données adéquat, qui prévoit pour l'utilisateur

définir des droits d'accès aux objets de la base de données en fonction des rôles.

2.6.1 Exigences relatives au contenu des données

Les tables de base suivantes doivent contenir au moins 10 enregistrements :

Comptes

Bugs ,

Captures d'écran,

Products,

Les tables suivantes doivent contenir au moins 20 enregistrements dont le contenu est cohérent.

sont cohérents entre eux.

Comments ,

Les listes de valeurs proposées dans le cahier des charges doivent être utilisées pour les tableaux de base suivants.

doivent être utilisées dans le cahier des charges :

Tags

BugPriority

BugStatus

BugSeverity

2.7 Environnement

2.7.1 Logiciel

Système d'exploitation PC / Notebook

- Windows 8 / 10

- Linux

Bundle XAMPP composé d'un serveur web, d'un SGBDR et du langage de script PHP

- Apache 2.x, MariaDB 10.4.x, PHP 5.7.x

Client de la base de données :

- HeidiSQL 11.x,

Outil de dessin de diagrammes UML pour la modélisation des données :

- UMLet 14.x

- UMLetino (version web d'UMLet)

Outil de conception de base de données :

- DB Designer 4.x

- MySQL Workbench 6.3 Community Edition (CE)

2.7.2 Matériel informatique

PC, ordinateur portable

...

2.7.3 Interfaces de produits

vers l'opérateur

vers le SGBD