Progetto basi di dati

1. Raccolta/Analisi dei requisiti

(testo completo delle specifiche sui dati, lista delle operazioni, tavola media dei volumi, glossario dei dati)

1.1 Specifiche

GIT-R è un gestore di versionamento tramite interfaccia web pensato per lo schema git-flow.
Per utilizzare GIT-R, è richiesto che l'utente sia registrato al servizio. È presente una funzionalità di login che permette l'accesso, inserendo username (univoco) e password.
Di norma i comuni servizi di hosting per il versionamento che utilizzano git (es. GitHub, GitLab, BitBucket, ...) prevedono uno spazio online visibile a tutti, in cui vengono rese disponibili a tutti le modifiche ai branch (e quando l'utente vuole pubblicarle, i push), ed uno spazio offline, in cui si effettuano commit. Git-R, porta l'intero processo online, dando all'utente la possibilità di avere uno spazio pubblico (la sezione "Projects"), visibile a tutti, ed uno privato (la sezione "Workbench"), in cui lavorare, effettuare commit e push.

Un utente registrato può:

- *Creare un nuovo progetto*, specificando il nome del progetto (univoco). Al momento della creazione del progetto il sistema crea la repository relativa, contenente un branch master ed un branch develop. Il branch master ed il branch development sono unici per ogni progetto.
- *Contribuire a un progetto pubblico*, effettuando un fork dei branch su cui è possibile lavorare secondo git-flow (ovvero: feature, hotfix e release, non master e develop). Verrà fatta una copia nella sezione privata, dei branch pubblici selezionati dall'utente.
- *Creare un feature branch*: unico modo di aggiungere nuove feature al progetto secondo git-flow. L'utente (che può essere sia il creatore del progetto che uno qualunque) deve inserire il nome (univoco) del branch che vuole creare, il sistema genera all'interno della directory del progetto una cartella pubblica con la versione pubblica del codice del branch, inizialmente copiandolo dal develop, e una cartella privata nel Workbench dell'utente contenente il codice su cui sta lavorando. Sullo stesso feature branch possono lavorare più utenti: in questo caso, la cartella pubblica sarà una ma ogni utente avrà la propria cartella privata. Per aggiornare il branch pubblico con il lavoro del branch privato si effettua un push, se quest'azione è effettuata dall'utente creatore del progetto viene approvata in automatico, altrimenti viene messa in attesa di approvazione da parte del creatore.
- *Creare un hotfix branch*: vale lo stesso discorso del feature branch, ma anziché essere creato a partire dal develop, è creato a partire dal master.
- *Creare un release branch*: vale lo stesso discorso del feature branch.
- *Effettuare un push/merge di un feature branch*: gli scenari sono due, uno in cui l'utente vuole effettuare un push del proprio branch privato, per rendere pubbliche le proprie modifiche nel feature branch pubblico, in questo caso se è l'utente creatore del progetto a fare l'operazione il push è approvato in automatico, altrimenti è messo in attesa di approvazione da parte del creatore. Il secondo scenario è che l'utente padre del progetto voglia effettuare un merge del feature branch pubblico nel develop branch, in questo caso il codice del feature pubblico è copiato nel develop.
- *Effettuare un push/merge di un hotfix branch*: analogo al push da feature branch, ma in caso di merge il codice è inserito sia nel develop branch che nel master branch.

- *Effettuare un push/merge di un release branch*: analogo ad un hotfix branch sia per push che per merge, è consigliabile che in questo branch si effettuino solo bugfix.
- *Effettuare un commit*: aggiornare il proprio lavoro all'interno di un branch della propria workbench.

Oltre la creazione di un progetto, di un branch feature, hotfix o release (pubblico o privato) è prevista anche l'eliminazione.
Nel caso in cui esistano conflitti tra un push di un utente non creatore e il branch pubblico, l'utente padre del progetto ha potere di visualizzare riga per riga cosa è stato aggiunto e cosa tolto, potendo poi approvare o rifiutare il push.
È prevista una utility di log per tenere traccia degli eventi accaduti, specificando data, ora, utente che ha causato l'evento, tipo di evento e ulteriori specificazioni.
È possibile in qualunque momento, se l'utente lo desidera, tornare ad una versione precedente del lavoro, sia all'interno del proprio workbench sia nei projects.

NOTA: come detto sopra il progetto GIT-R vuole gestire attraverso una base di dati e un'applicazione web il solo processo git-flow, per questo non è stato ritenuto opportuno gestire tramite database la sezione dei pending push. L'implementazione esula dal processo che si vuole mostrare, nel nostro caso è stata gestita tramite creazione e cancellazione di particolari cartelle nel server.

1.2 Lista operazioni

1. Registrare un nuovo utente
2. Effettuare il login come utente registrato
3. Effettuare il logout come utente registrato
4. Creare un nuovo progetto
5. Cancellare un progetto esistente
6. Creare un branch di un proprio progetto
7. Creare un branch di un progetto altrui
8. Fare il fork dei branch di un progetto esistente
9. Cancellare un branch privato (visibile solo all'utente che lo possiede)
10. Cancellare un branch pubblico (visibile a tutti gli utenti)
11. Effettuare un commit del proprio lavoro
12. Tornare ad una versione precedente di un branch privato
13. Tornare ad una versione precedente di un branch pubblico
14. Effettuare un push di un branch privato in uno pubblico
15. Effettuare un merge di un branch pubblico in develop (e a volte anche in master)
16. Effettuare un merge di develop in master
17. Visualizzare la versione del master branch
18. Visualizzare le operazioni effettuate dagli utenti tramite un log

[Ogni operazione riguardante i branch è da ritenersi valida per feature branch, release branch, hotfix branch]

1.3 Tavola media dei volumi

Entità	Stima del volume

User	10.000
Project	40.000
Repository	40.000
Master Branch	40.000
Development Branch	40.000
Feature branch public	400.000
Feature branch private	800.000
Release branch public	40.000
Release branch private	120.000
Hotfix branch public	20.000
Hotfix branch private	60.000
Log	10.000.000

1.4 Glossario

Nome	Descrizione	Sinonimi	Collegamenti
Workbench	Spazio privato dell'utente, contiene i branch privati.	Spazio privato	Feature branch privato, hotfix branch privato, release branch privato
Project	Spazio pubblico, contiene i branch pubblici.	Spazio pubblico	Feature branch pubblico, hotfix branch pubblico, release branch pubblico, master branch, development branch
Public	Disponibile per la visualizzazione ed il fork a tutti.	Pubblico	
Private	Disponibile solo all'utente proprietario. Al suo interno è possibile la modifica del lavoro ed il commit.	Privato	
Master branch	Branch principale dove il lavoro effettuato è in stato	Master	Development branch, repository, hotfix branch,

	production-ready. È unico per ogni progetto.		release branch
Develop branch	Branch principale dove il lavoro effettuato riflette gli ultimi cambiamenti aggiunti in vista della prossima release. È unico per ogni progetto.	Develop	Feature branch, release branch, master branch
Release branch	Branch che preparano una nuova release. Possono contenere aggiustamenti minori e piccoli bugfix.	Release	Develop branch, master branch
Hotfix branch	Simili ai release branch, preparano una nuova release quando questa si rende necessaria a causa di bug non individuati al rilascio della precedente.	Hotfix	Master branch, release branch, development branch
Feature branch	Servono a sviluppare le nuove feature presenti nelle successive (anche non immediatamente) release.	Feature	Master branch, release branch, development branch
Repository	Cartella remota contenente le directory del progetto.	Cartella	master branch, project
User	Utente registrato.	Utente	Project, log
Utente padre	User in rispetto al progetto che ha creato		Project, Repository
Log	Entità contenente traccia dei cambiamenti effettuati.		User
Push	Aggiornamento (o richiesta di aggiornamento) del contenuto di un branch pubblico, con il lavoro svolto in privato da un utente.		Feature branch pubblico, hotfix branch pubblico, release branch pubblico, master branch, development branch
Merge	Aggiornamento del contenuto di un branch in uno di “grado” più alto (feature -> develop ; hotfix, release -> develop/master ; develop -> master).		Feature branch pubblico, hotfix branch pubblico, release branch pubblico, master branch, development branch
Commit	Aggiornamento del contenuto di una cartella privata (all’interno della *workbench*)		Feature branch privato, hotfix branch privato, release branch privato

2. Progettazione Concettuale

2.1 Diagramma E-R

Sono disponibili i file '*allegati/ER.png*' e '*allegati/er.er*' contenenti rispettivamente il diagramma ER come immagine e come file JDER.

2.2 Dizionario Entità/Relazioni

Entità	Descrizione	Attributi	Identificatore
user	Utente registrato al sistema	username, password	username
project	Progetto di un utente	name, creator, creation_date	name
repository	Cartella contenente il project	id, url, project_name	id
log	Azione effettuata nell'ambiente GIT-R	id, date, time, username, action, element	id
access	Collegamento tra repository e utente che vi lavora	username, repo_id	username, repo_id
master	Branch master di un progetto	repo_id, url, version	repo_id
develop	Branch develop di un progetto	master_id, url, version	master_id
feature_public	Branch visibile a tutti, che sviluppa una particolare feature	develop_id, name, url, notes, version	develop_id
feature_private	Branch su cui un utente lavora, per contribuire ad un feature branch pubblico	feature_pub_id, feature_pub_name, username, url, version	feature_pub_id, feature_pub_name, username
release_public	Branch visibile a tutti, che sviluppa una versione consegnabile	develop_id, url, notes, version	develop_id
release_private	Branch su cui un utente lavora, per contribuire ad un release branch pubblico	release_pub_id, username, url, version	release_pub_id, username
hotfix_public	Branch visibile a tutti, che sviluppa una bugfix urgente	develop_id, url, notes, version	develop_id

hotfix_private	Branch su cui un utente lavora, per contribuire ad un hotfix branch pubblico	hotfix_pub_id, username, url, version	hotfix_pub_id, username

Relazione	Descrizione	Tipo	Componenti
performs	Associa un log ad un utente	uno-a-molti	log, user
creates	Associa un utente ad un progetto	uno-a-molti	user, project
points	Associa un progetto ad una repository	uno-a-uno	project, repository
access	Associa un utente ad una repository	molti-a-molti	user, repository
R1	Associa una repository ad un master	uno-a-uno	repository, master
R2	Associa un master ad un develop	uno-a-uno	master, develop
features	Associa un branch feature pubblico ad un develop	uno-a-molti	feature_public, develop
releases	Associa un branch release pubblico ad un develop	uno-a-molti	release_public, develop
fixes	Associa un branch hotfix pubblico ad un develop	uno-a-molti	hotfix_public, develop
R3	Associa un branch feature privato ad un branch feature pubblico	uno-a-molti	feature_private, feature_public
R4	Associa un branch release privato ad un branch release pubblico	uno-a-molti	release_private, release_public
R5	Associa un branch hotfix privato ad un branch hotfix pubblico	uno-a-molti	hotfix_private, hotfix_public
R6	Associa un utente ad un branch feature privato	uno-a-molti	user, feature_private
R7	Associa un utente ad un branch hotfix privato	uno-a-molti	user, hotfix_private
R8	Associa un utente ad un branch release privato	uno-a-molti	user, release_private

2.3 Business rules

1. Non può esistere più di un master branch per progetto.
2. Non può esistere più di un development branch per progetto.
3. Non può esistere più di un hotfix branch per progetto.
4. Non può esistere più di un release branch per progetto.
5. Non può esistere più di un feature branch per progetto con lo stesso nome.

6. Un utente per lavorare su un progetto deve averne l'accesso.
7. Per effettuare modifiche al codice, un utente deve creare o essere all'interno di un branch che non sia né master né development.
8. Le azioni significative sulle cartelle ed i file devono essere presenti all'interno del log.

3. Progettazione Logica

3.1 Ristrutturazione schema concettuale

Tutte le relazioni uno-a-molti sono state trasformate in attributi (ad esempio, fixes è stata inglobata da hotfix_public con l'aggiunta del campo develop_id). Le relazioni uno-a-uno sono state trasformate in maniera analoga, ad esempio R2 è stata inglobata da develop includendo il campo master_id. Relazioni di altro tipo sono state tradotte come tabelle (ad esempio, access).

3.2 Analisi ridondanze

Non sono presenti ridondanze eliminabili. L'inserimento di ridondanze per velocizzare le operazioni di select non è utile o perché non esistono operazioni dove ciò sia possibile (ad esempio, non sono previste operazioni di *count),* o perché l'attributo che si potrebbe pensare di replicare nella ridondanza è già presente come chiave primaria (ad esempio, username nella tabella feature_pri).

3.3 Lista tabelle con vincoli di chiave

user(username, password)
project(name, *creator*, creation_date)
repository(id, url, *project_name*)
log(id, date, time, *username*, action, element)
access(*username, repo_id*)
master(*repo_id*, url, version)
develop(*master_id*, url, version)
feature_pub(*develop_id*, name, url, notes, version)
feature_pri(*feature_pub_id, feature_pub_name*, *username*, url, version)
release_pub(*develop_id*, url, notes, version)
release_pri(*release_pub_id*, *username*, url, version)
hotfix_pub(*develop_id*, url, notes, version)
hotfix_pri(*hotfix_pub_id*, *username*, url, version)

3.4 Lista vincoli inter-relazionali

project.creator -> **user**.username
repository.project_name -> **project**.name
log.username -> **user**.username
access.username -> **user**.username
access.repo_id -> **repository**.id
master.repo_id -> **repository**.id
develop.master_id -> **master**.repo_id
feature_pub.develop_id -> **develop**.master_id
feature_pri.feature_pub_id -> **feature_pub**.develop_id
feature_pri.feature_pub_name -> **feature_pub**.name
feature_pri.username -> **user**.username
release_pub.develop_id -> **develop**.master_id

release_pub.release_pub_id -> **release_pub**.develop_id
release_pub.username -> **user**.username
hotfix_pub.develop_id -> **develop**.master_id
hotfix_pub.hotfix_pub_id -> **hotfix_pub**.develop_id
hotfix_pub.username -> **user**.username

4. Normalizzazione

La normalizzazione non è necessaria in quanto le tabelle sono in terza forma normale.

5. Descrizione ad alto livello delle funzionalità dell'applicazione Web

Una volta effettuata la registrazione e il login l'utente si troverà nella schermata principale dell'applicativo. La barra in alto è divisa in due parti: a sinistra ci sono le sezioni di lavoro ("Projects" per i progetti pubblici, "Workbench" per i branch privati, ecc.), a destra informazioni (sull'utente loggato e il branch selezionato) ed alcune azioni (push/merge e commit dove permesso). È possibile creare/eliminare un progetto, richiedere l'associazione o creare/eliminare un branch; una volta selezionato il documento da modificare nella propria workbench si può procedere tramite l'editor presente. È possibile anche riportare un branch ad una versione precedente, e gestire le richieste di push ai propri progetti, visualizzando le differenze tra il push e la versione attuale del branch.
Infine, è accessibile a tutti una pagina di visualizzazione del log.

NOTE PER L'UTILIZZO: il sistema è testato e funzionante sulle macchine Debian (GNU/Linux), come quelle disponibili agli studenti in laboratorio. È stato inoltre utilizzato il browser Firefox. È presente un file *initdb.php* nella cartella *system/* che permette di autosettare il database in modo automatico, è però necessario impostare manualmente il path assoluto (se diverso da quello attuale) per raggiungere il file */sql/db.sql* nella cartella dei sorgenti del progetto, contenente tutte le istruzioni SQL. Creare un utente "user" in MySQL, con permessi di creazione database e tutti i permessi per il database "gitr".
Vanno infine forniti alla cartella *Users/* nel server locale tutti i permessi:

```
$ sudo chmod -R 777 Users/
```

- Appendice

È disponibile il file '*allegati/db.sql*' contenente tutto il codice SQL utilizzato dal progetto. Tale file è lo stesso presente nella cartella dei sorgenti del progetto '/sql/db.sql' che verrà anche eseguito in automatico all'esecuzione di '/system/initdb.php'.