SEVEN CLICKERS

You Wish We Click

7clickersgroup@gmail.com

Norme di Progetto

Versione	0.0.3
Stato	In Sviluppo
Uso	Interno
Approvazione	-
Redazione	Mirko Stella Giacomo Mason Gabriele Mantoan
Verifica	-
Distribuzione	*Seven Clickers* Prof. Vardanega Tullio Prof. Cardin Riccardo

Descrizione
Questo documento contiene le norme di progetto seguite dal gruppo Seven Clickers per il progetto ShowRoom3D

Registro delle modifiche

Vers.	Data	Autore	Ruolo	Descrizione
0.0.3	16-11-22	Gabriele Mantoan	-	Modificate norme riguardo verifica$_g$ e vita delle issue$_g$; aggiunte norme riguardo Jira$_g$ e indici di glossario
0.0.2	13-11-22	Gabriele Mantoan	-	Aggiunte norme riguardo Issue tracking system$_g$, ruoli e nomi dei branch liberi
0.0.1	06-11-22	Mirko Stella Giacomo Mason	-	Creazione documento

Indice

1 Introduzione

1.1 Scopo del documento

Lo scopo del documento è quello di stabilire le regole che ogni componente del gruppo SevenClickers deve rispettare per mantenere un ambiente di lavoro che mira a massimizzare l'economicità dei processi durante il ciclo di vita del prodotto ShowRoom3D.
Le norme verranno inserite in modo incrementale per regolamentare le attività di progetto imminenti rimandando quelle meno urgenti a quando se ne presenterà la necessità.
Inoltre tali norme potranno subire modifiche nel tempo in modo da garantire un miglioramento continuo della qualità del lavoro svolto. Il responsabile di progetto ha il compito di comunicare l'aggiunta di una nuova norma o la modifica di una già esistente a tutti i componenti del gruppo e di assicurarsi che siano comprese a pieno.

1.2 Scopo del prodotto

Il prodotto in questione nasce dalla necessità dell'azienda SanMarco Informatica di fornire una soluzione agli sprechi derivati dall'adozione di uno ShowRoom tradizionale proponendo uno ShowRoom3D che sia ugualmente o ancora più immersivo.

2 Documentazione

Questa sezione descrive le convenzioni,gli strumenti e le modalità con cui il gruppo si impegna a stilare la documentazione interna ed esterna relativa al progetto.

2.1 Convenzioni generali

Le convenzioni di seguito riportate vengono applicate a tutti i documenti. Esse rendono i documenti stilati omogenei tra loro contribuendo a rendere il progetto professionale.

2.1.1 Versionamento

Il numero di versione permette di capire lo stato in cui si trova un documento. Un documento può trovarsi nei seguenti stati:

- **Approvato**:Il documento è verificato ed approvato dal Responsabile di progetto
- **Verificato**:Il documento risulta verificato ma non ancora visionato dal Responsabile di progetto
- **In Sviluppo**:Sono presenti delle modifiche che non sono state verificate

Il numero di versione ha il formato **X.Y.Z** dove:

- **X** indica una versione approvata dal Responsabile di progetto,la numerazione parte da 0 e la prima versione approvata è la 1.0.0
- **Y** indica una versione verificata dal Verificatore,la numerazione inizia da 0 e si azzera ad ogni incremento di X.La prima versione verificata è la 0.1.0
- **Z** indica una versione in fase di modifica da parte dei redattori che ne incrementano il numero ad ogni modifica, la numerazione parte da 1 e si azzera ad ogni incremento di X o Y.La prima versione modificata è la 0.0.1

2.1.2 Struttura Generale

Ogni documento deve presentare le seguenti sezioni nell'ordine in cui vengono presentate:

- **Intestazione**: Contiene:
  - Logo compreso di motto
  - Indirizzo email di gruppo
  - Titolo
  - Tabella contenente le informazioni generali
    - Versione
    - Stato
    - Uso
    - Approvazione: indica il responsabile di progetto che ha approvato il documento
    - Redazione: elenco dei collaboratori che hanno partecipato alla stesura del documento
    - Verifica: elenco dei verificatori che hanno verificato il documento
    - Distribuzione: elenco delle persone o organizzazioni a cui è destinato il documento
  - Breve descrizione del documento
- **Registro delle modifiche**: Tabella che identifica ogni versione del documento indicandone:
  - Versione
  - Data
  - Autore
  - Ruolo
  - Descrizione
- **Indice**: Elenco ordinato dei titoli dei capitoli, ovvero delle varie parti di cui si compone il documento.
- **Contenuto**: Varia a seconda del tipo di documento.

2.1.3 Verbali

Rispettano tutta la struttura generale. In aggiunta presentano:

- **Informazioni Generali** Contengono:
  - Luogo
  - Data
  - Ora
  - Partecipanti
- **Tabella tracciamento temi affrontati**: tabella che riassume i punti salienti della riunione indicandone
  - Codice: ha il formato V**X** **Y**.**Z** dove X indica la tipologia di verbale,Y indica il numero di verbale (incrementale rispetto agli altri verbali) e Z indica il numero dell'argomento trattato (incrementale rispetto agli altri argomenti del verbale)
  - Descrizione: breve descrizione di uno specifico argomento trattato

2.1.4 Date

Le date devono rispettare il seguente formato: **dd-mm-yyyy** All'interno delle tabelle il formato deve essere il seguente: **dd-mm-yy**

2.1.5 Nomi di persona

All'interno dei documenti i nomi di persona rispetteranno l'ordine nome seguito dal cognome della persona menzionata.

2.2 Verifica

La verifica viene svolta da due verificatori prima del merge con il branch documentation. Consiste nell'esaminare i file prodotti da chi ne ha fatto la stesura e segnalarne la non validità o la presenza di errori nei concetti esposti.
Un verificatore dovrà verificare il documento a partire dalle modifiche fatte dopo l'ultima versione verificata. Le modifiche da verificare quindi possono essere dedotte dal registro dei cambiamenti presente in ogni documento. Una volta controllato il documento, il primo verificatore segnalerà eventuali errori e successivamente dovrà spuntare come approvata la Pull Request nella sezione dedicata su GitHub$_g$.
A questo punto se un secondo verificatore noterà la necessità di qualche altro cambiamento da apportare, chi dovrà apportare le modifiche farà una pull in locale per allineare il proprio branch con quello in remoto e continuare con il proprio lavoro.
Dopo il push delle modifiche se i file risultano corretti anche dal secondo verificatore esso aggiungerà i nomi dei verificatori all'intestazione modificando il file titlepage_input.tex e creerà la riga nel registro delle modifiche, nel file changelog_input.tex, inserendo la nuova versione secondo le regole di versionamento e scrivendo nella colonna Autore sia il suo nome, che quello del primo verificatore, e in descrizione "Verifica". Dopo aver eseguito un commit sul ramo da integrare approva la Pull Request e conferma il merge secondo le norme di progetto descritte nella sezione dedicata.

2.3 Approvazione

IN SOSPESO

2.4 Strumenti per la stesura

- Latex: IN SOSPESO

3 Strumenti collaborativi

3.1 GitHub$_g$

Servizio di hosting per progetti software che implementa uno strumento di controllo versione distribuito Git$_g$. Oltre alla copia in remoto del repository$_g$ di progetto ogni componente del gruppo ha una propria copia in locale.
Per ottenere una copia del repository$_g$ ogni componente ha scaricato lo strumento Git$_g$ ed eseguendo il comando 'git clone' da git$_g$ bash viene creata una cartella collegata alla repository$_g$ di progetto.
Non sono state imposte modalità specifiche sull'interazione con il repository$_g$ remoto in modo da non sconvolgere le abitudini di lavoro di ciascun componente.
I componenti del gruppo abituati ad interagire con GitHub$_g$ da interfaccia grafica possono continuare a farne uso.

3.1.1 Repository$_g$

Il repository$_g$ si può trovare all'indirizzo ***https://github.com/7clickers/ShowRoom3D*** ed è pubblico.
I collaboratori sono i componenti del gruppo SevenClickers che utilizzano il proprio account GitHub$_g$ personale per collaborare al progetto.

3.1.2 Branching

Branches protetti:

- main: contiene le versioni di release$_g$ del software
- documentation: contiene i template latex$_g$ e rispettivi pdf della documentazione

documentation: I documenti presenti in documentation sono stati approvati dal Responsabile di progetto o almeno verificati dai verificatori.
Branches liberi: Vengono utilizzati per creare nuove funzionalità e gli sviluppatori possono effettuare i commit senza l'approvazione degli altri componenti del gruppo in quanto ciascun componente sviluppa su un solo branch alla volta salvo casi eccezionali. Un branch$_g$ libero avrà il nome del documento che si sviluppa su quel branch$_g$, oppure della feature$_g$ che va ad implementare.

3.1.3 Commits

È preferibile che ogni commit abbia una singola responsabilità per cambiamento.

I commits non possono essere effettuati direttamente sui branch protetti ma per integrare delle aggiunte o modifiche sarà necessario aprire una Pull Request. All'approvazione di una Pull Request tutti i commit relativi al merge verranno raggruppati in un unico commit che rispetti la struttura sintattica descritta in seguito.

I commit dovranno essere accompagnati da una descrizione solo se ritenuta indispensabile alla comprensione del commit stesso.

I messaggi di commit sui **BRANCH PROTETTI** dovranno seguire la seguente struttura sintattica:

<label><#n_issue><testo>

dove:

label: può assumere i seguenti valori

- feat: indica che è stata implementata una nuova funzionalità
- fix: indica che è stato risolto un bug
- update: indica che è stata apportata una modifica che non sia fix o feat
- test: qualsiasi cosa legata ai test
- docs: qualsiasi cosa legata alla documentazione

n_issue: indica il numero della issue a cui fa riferimento il commit (se non fa riferimento a nessuna issue viene omesso).

testo:indica con quale branch è stato effettuato il merge e deve rispettare la forma: merge from <nome branch da integrare> to <nome branch corrente>

descrizione: se aggiunta ad un commit deve rispondere alle domande WHAT?,WHY?,HOW? ovvero cosa è cambiato,perchè sono stati fatti i cambiamenti,in che modo sono stati fatti i cambiamenti.

I messaggi di commit sui **BRANCH LIBERI** dovranno seguire la struttura sintattica dei branch protetti ad eccezione del testo. Il testo dei commit sui branch liberi non è soggetto a restrizioni particolari a patto che indichi in maniera intuitiva i cambiamenti fatti in modo che possano essere compresi anche dagli altri collaboratori.

3.1.4 Pull Requests

Per effettuare un merge su un branch protetto si deve aprire da GitHub$_g$ una Pull Request. La Pull Request permette di verificare il lavoro svolto prima di integrarlo con il branch desiderato. I verificatori in carica hanno il compito di trovare eventuali errori o mancanze e fornire un feedback riguardante il contenuto direttamente su GitHub$_g$ richiedendo una review con un review comment sulla parte specifica da revisionare o con un commento generico.
Non sarà possibile effettuare il merge finchè tutti i commenti di revisione non saranno stati risolti. Il merge potrà avvenire dopo aver risolto tutti i commenti di revisione ed approvato da almeno due verificatori. Per i commit relativi alle Pull Requests seguire le regole descritte nella sottosezione Commits per i branch protetti.

3.1.5 Issue Tracking System$_g$

Il gruppo utilizza l'issue tracking system$_g$ di Github$_g$ per tenere traccia delle issue$_g$. Le issue verranno determinate dal responsabile, ma la loro assegnazione verrà effettuata dai membri del gruppo, in base alle loro preferenze, i loro ruoli e alle loro disponibilità temporali.
Nel caso un membro del gruppo dovesse rendersi conto che l'issue$_g$ che sta svolgendo potrebbe essere suddiviso in ulteriori issue$_g$, dovrà rivolgersi al responsabile, che è l'unico che può aggiungere, modificare o eliminare le issue$_g$.
Ogni issue ha almeno un tag associato per specificarne lo scopo, inoltre il gruppo ha aggiunto il seguente tag a quelli già forniti da Github$_g$:

- **approvable**: indica una issue$_g$ che dopo esser stata verificata$_g$, deve anche essere approvata$_g$ dal responsabile per andare in fase di release$_g$
- **tag di priorità (alta/media/bassa)**:descrivono quanto è urgente il completamento dell'issue$_g$

Verranno inoltre predisposte delle **milestone**$_g$ con una scadenza temporale a cui potranno essere associate le issue. La scadenza delle milestone$_g$ e la loro creazione non seguono regole specifiche, ma è preferibile avere almeno una milestone ogni 2 o 3 settimane, allineate quindi con il cambio di ruoli dei membri del gruppo.
Le issue verranno tracciate nella **projectboard**$_g$ della repo, che è visibile e modificabile da tutti i membri del gruppo. La projectboard$_g$ è suddivisa in queste sezioni:

- **Todo**: Issue$_g$ che non sono ancora state iniziate o che non sono ancora state assegnate
- **In Progress**: Issue$_g$ che sono state assegnate e a cui almeno un membro a cui è stata assegnata ha iniziato a lavorarci
- **Pull Request**: Issue$_g$ che è in fase di integrazione e necessita della verifica dei verificatori. Corrisponde all'inizio di una pull request
- **Done**: Issue$_g$ che sono state chiuse e che sono state verificate (se necessitano di verifica$_g$)
- **Approved**: Issue$_g$ col tag "approvable", e che hanno ottenuto l'approvazione$_g$ del responsabile

Inoltre nella project board vengono registrate delle issue che non richiedono verifica, approvazione o neanche integrazione, con lo scopo di monitorare meglio il lavoro di ogni membro del team.
Queste issue verranno chiuse e archiviate manualmente una volta che avranno terminato la loro utilità, un esempio può essere la seguente issue:
diario di bordo 21-11-22; questa issue non necessità verifica, approvazione o integrazione perchè non è di interesse caricare il file nella repo, però è utile tracciare lo svolgimento della issue

3.2 Jira$_g$

All'utilizzo di Github$_g$ si affianca l'issue tracking system$_g$ di Jira alle quali se applicano le stesse convenzioni del precedente.
Il motivo per cui si è fatta questa scelta è la possibilità di aggiungere alle issue una data di scadenza e le possibilità di visualizzazione e automazione aggiuntive che offre Jira$_g$.

3.3 Glossario

All'interno del documento si possono trovare dei termini che possono risultare ambigui a seconda del contesto,o non conosciuti dagli utilizzatori.
Per ovviare ad errori di incomprensione che possono portare a problemi di vario genere e rallentamenti si è deciso di stilare un elenco di termini di interesse accompagnati da una descrizione dettagliata del loro significato.
I termini presenti all'interno del glossario vengono indicati con il pedice 'g' come nell'esempio seguente: termine$_g$. Il glossario ordina i termini in ordine alfabetico in modo da permetterne una facile e veloce ricerca. Ogni componente del gruppo all'inserimento di un termine ritenuto ambiguo deve preoccuparsi di aggiornare il glossario in modo da mantenerlo sempre aggiornato.

4 Organizzazione del gruppo

4.1 Ruoli

I componenti del gruppo si suddivideranno nei seguenti ruoli per periodi di circa 2-3 settimane (dipendentemente dalle esigenze del periodo) e al termine del periodo i ruoli verranno risuddivisi. Visto che nelle varie fasi di sviluppo del progetto le attività da svolgere variano, non sempre sarà necessario coprire tutti i ruoli.
Inoltre sarà necessario tenere traccia delle ore che ogni componente dedica al progetto ed il ruolo associato a quelle ore, in modo da andare a rispettare la tabella degli impegni individuali.
I ruoli e le loro competenze sono i seguenti:

4.1.1 Responsabile

Deve avere la visione d'insieme del progetto e coordinare i membri, inoltre si occupa di rappresentare il gruppo con le interazione esterne (proponente, committente ecc...). Le sue competenze specifiche sono:

- ad ogni iterazione$_g$ c'è un solo responsabile
- presentare il diario di bordo in aula
- redarre l'ordine del giorno prima di ogni meeting interno del gruppo
- suddivide le attività del gruppo in singole issue (ma non le assegna ai membri del gruppo)
- in fase di release$_g$ si occupa di approvare$_g$ tutti i documenti che necessitano approvazione

4.1.2 Analista

Si occupa di trasformare i bisogni del proponente nelle aspettative che il gruppo deve soddisfare per sviluppare un prodotto professionale. Le sue competenze specifiche sono:

- interrogare il proponente riguardo allo scopo del prodotto e le funzionalità che deve avere
- studiare le risposte del proponente per identificare i requisiti$_g$ e redarre l'analisi dei requisiti

4.1.3 Amministratore

Si occupa del funzionamento, mantenimento e sviluppo degli strumenti tecnologici usati dal gruppo. Le sue competenze specifiche sono:

- ad ogni iterazione$_g$ basta un solo amministratore
- gestione delle segnalazioni e problemi dei membri del gruppo riguardanti problemi e malfunzionamenti con gli strumenti tecnologici
- valuta l'utilizzo di nuove tecnologie e ne fa uno studio preliminare per poter presentare al gruppo i pro e i contro del suo utilizzo

4.1.4 Progettista

Si occupa di scegliere la modalità migliore per soddisfare le aspettative del committente che gli analisti hanno ricavato dall'analisi dei requisiti$_g$. Le sue competenze specifiche sono:

- scegliere eventuali pattern architetturali da implementare
- sviluppare lo schema UML$_g$ delle classi$_g$

4.1.5 Programmatore

Si occupa di implementare le scelte e i modelli fatti dal progettista. Le sue competenze specifiche sono:

- scrivere il codice atto a implementare lo schema delle classi
- scrivere eventuali test
- scrivere la documentazione per la comprensione del codice che scrive

4.1.6 Verificatore

Si occupa di controllare che ogni file che viene caricato in un branch protetto della repository$_g$ sia conforme alle norme di progetto. Le sue competenze specifiche sono:

- controllare i file modificati o aggiunti durante una pull request tra un ramo non protetto e un ramo protetto siano conformi alle norme di progetto e cercano errori di altra natura (ortografici, sintattici, logici, build ecc...).