NORME DI PROGETTO

Versione 0.1.2 in data 12-12-2017

Gruppo 353 - Progetto *Marvin*

Informazioni sul documento

Responsabili	Elena Mattiazzo
	Valentina Marcon
	Mirco Cailotto
	Riccardo E. Giorato
	Gianluca Marraffa
	Parwinder Singh
	Davide Stocco
Redazione	
Verifica	Davide Stocco
	Riccardo E. Giorato
Stato	In corso
Uso	Interno
Destinato a	Gruppo 353
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
Email di contatto	353swe@gmail.com

Diario delle modifiche

Versione	Data	Descrizione	Autore	Ruolo
0.1.2	12-12-2017	Verifica processi primari	Riccardo E. Giorato	Verificatori
0.1.1	11-12-2017	Verifica processi organizzativi	Riccardo E. Giorato	Verificatori
0.1.0	9-12-2017	Verifica documento processi di supporto	Davide Stocco	Verificatori
0.0.2	4-12-2017	Stesura introduzione, fornitura e stile di codifica per Solidity e Scss	Elena Mattiazzo	Analista
0.0.1	26-11-2017	Creazione scheletro del documento	Elena Mattiazzo	Analista

Indice

1. Introduzione

1.1 Scopo del documento

Lo scopo di questo documento è fissare tutte le regole e le procedure fondamentali per assicurare al gruppo un modo di lavorare comune, le quali verranno discusse in tutte le sezioni seguenti.
Ciò garantirà una collaborazione efficiente tra tutti i diversi membri del gruppo.
Saranno inoltre elencati e discussi tutti gli strumenti software scelti internamente al gruppo per rispettare le regole e per attuare le procedure.

1.2 Scopo del prodotto

Lo scopo del prodotto è quello di realizzare una piattaforma web chiamata *Marvin* che simuli le funzionalità di base per studenti, docenti e università di Uniweb$_{\text{G}}$.
L'applicativo dovrà utilizzare al posto del database la rete Ethereum$_{\text{G}}$ interagendo con degli smart contracts.

1.3 Glossario

All'interno del documento sono presenti termini che presentano significati ambigui a seconda del contesto. Per evitare questa ambiguità è stato creato un documento di nome Glossario che conterrà tali termini con il loro significati specifico. Per segnalare che un termine del testo è presente all'interno del *Glossario v 1.0.0* verrà aggiunta una G a pedice a fianco del termine.

1.4 Riferimenti utili

1.4.1 Riferimenti normativi

- Standard ISO/IEC 12207:1995
  http://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf

1.4.2 Riferimenti informativi

- ***Piano di Progetto v 1.0.0***
- ***Piano di Qualifica v 1.0.0***
- **Airbnb JavaScript Style Guide**
  https://github.com/airbnb/javascript
- **Documentazione officiale di Solidity**
  http://solidity.readthedocs.io/en/develop/style-guide.html
- **Airbnb CSS/Sass Style Guide**
  https://github.com/airbnb/css
- **Sito del corso di Ingegneria del Software - Regolamento Organigramma**
  http://www.math.unipd.it/~tullio/IS-1/2017/Progetto/RO.html
- **Sito del corso di Ingegneria del Software - Analisi dei Requisiti**
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/L08.pdf

2. Processi primari

2.1 Fornitura

In questa sezione vengono tratta le norme che i membri del gruppo 353 sono tenuti a rispettare al fine di proporsi e diventare fornitori nei confronti della Proponente Red Babel e dei Committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin nell'ambito della progettazione, sviluppo e consegna del prodotto *Marvin*.

2.1.1 Studio di fattibilità

In seguito alla presentazione ufficiale dei **Capitolati d'appalto** avvenuta Venerdì 10 novembre 2017 alle ore 10.30 presso l'aula 1C150 di Torre Archimede è stata convocata una riunione interna al gruppo per discutere in merito alla varie proposte presentate.
Una volta stabilita la scelta del capitolato per il quale proporsi come fornitori, gli analisti hanno condotto un'ulteriore e approfondita attività di analisi dei rischi e delle opportunità culminata con la redazione del documento *Studio di fattibilità v 1.0.0*. Tale documento include le motivazioni che hanno portato il gruppo 353 a proporsi come fornitore per il prodotto indicato e riporta per ciascun capitolato:

- **Descrizione generale:** una sintesi del prodotto da sviluppare secondo quanto stabilito dal capitolato d'appalto;
- **Obbiettivo finale:** rappresenta il Dominio Applicativo, cioè l'ambito di utilizzo del prodotto da sviluppare;
- **Tecnologie richieste:** rappresenta il Dominio Tecnologico richiesto dal capitolato, raggruppando le tecnologie da impiegare nello sviluppo del progetto;
- **Valutazione finale:** racchiude le motivazioni, i rischi, le criticità evidenziate per le quali il capitolato in questione è stato respinto o accettato.

2.2 Sviluppo

2.2.1 Analisi dei Requisiti

L'obbiettivo dell'analisi dei requisiti è quello di individuare ed elencare tutti i requisiti$_G$ del capitolato. I requisiti possono essere estrapolati da più fonti:

- **Capitolato d'appalto**;
- **Verbali di riunioni esterne**;
- **Casi d'uso**.

Il risultato di quest'analisi sarà il documento *Analisi dei requisiti v 1.0.0* redatto dagli Analisti, dopo essere stato verificato, al fine di:

- Descrivere l'obiettivo del lavoro del gruppo;
- Fornire ai Progettisti riferimenti precisi ed affidabili;
- Fissare le funzionalità e i requisiti$_G$ concordati col cliente;
- Fornire una base per raffinamenti successivi al fine di garantire un miglioramento continuo del prodotto e del processo di sviluppo;
- Facilitare le revisioni del codice;
- Fornire ai Verificatori riferimenti per l'attività di test circa i casi d'uso principali e alternativi;
- Stimare i costi.

Ogni requisito dovrà essere meno ambiguo possibile e rispettare le seguenti norme.

2.2.1.1 Classificazione dei requisiti

Ogni requisito è identificato da un codice, costruito come descritto di seguito:

R[Importanza][Classificazione][Identificativo]

- La prima lettera (R) è l'abbreviazione di requisito;
- Il secondo valore indica l'**importanza**. Assume il valore:

  - **Zero (0):** indica un requisito obbligatorio, il cui soddisfacimento dovrà necessariamente avvenire per garantire le funzioni base del sistema;
  - **Uno (1):** se si tratta di un requisito desiderabile, cioè un requisito il cui soddisfacimento può dare maggiore completezza al sistema ma il non soddisfarlo non pregiudica alcuna funzione di base;
  - **Due (2):** indica un requisito opzionale;

- La terza lettera indica la **classificazione**. Assume valore F se si tratta di un Requisito funzionale, Q se di qualità, P se prestazionale e V se di vincolo;
- L'**identificativo** indica invece un numero progressivo.

2.2.1.2 Classificazione casi d'uso

gli Analisti hanno anche il compito di identificare i casi d'uso, elencandoli con un grado di precisione che va dal generico verso il dettaglio. Ogni caso d'uso è descritto dalla seguente struttura:

- **Codice identificativo e nome:** ogni caso d'uso è identificato da una serie di cifre separate dal punto. L'ultima cifra indica il numero del figlio, la penultima cifra indica il numero del padre e UC è l'abbreviazione di Use Case (caso d'uso in inglese). Queste cifre sono seguite dopo il trattino (-) dal nome del caso d'uso:

  UC[Codice padre].[Codice figlio] - Nome

- **Attori:** indica gli attori principali (ad esempio l'utente generico) e secondari (ad esempio ufficio universitario) del caso d'uso;
- **Scopo e descrizione:** riporta una breve descrizione del caso d'uso;
- **Scenario principale:** rappresenta il flusso degli eventi come lista numerata, specificando per ciascun evento: titolo, descrizione, attori coinvolti e casi d'uso generati;
- **Precondizione:** specifica le condizioni che sono identificate come vere prima del verificarsi degli eventi del caso d'uso;
- **Postcondizione:** specifica le condizioni che sono identificate come vere dopo il verificarsi degli eventi del caso d'uso;
- **Inclusioni (se presenti):** usate per non descrivere più volte lo stesso flusso di eventi, inserendo il comportamento comune in un caso d'uso a parte;

- **Estensioni (se presenti):** descrivono i casi d'uso che non fanno parte del flusso principale degli eventi, allo stesso modo di quanto descritto in "Scenario principale".

2.2.2 Progettazione

2.2.2.1 Scopo

L'attività di Progettazione consiste nel descrivere una soluzione del problema che sia soddisfacente per tutti gli stakeholders. È compito dei Progettisti svolgere tale attività, definendo l'architettura logica del prodotto identificando componenti chiare, riusabili e coese rimanendo nei costi fissati. L'architettura definita dovrà:

- Soddisfare i requisiti$_G$ definiti nel documento *Analisi dei requisiti* e adattarsi facilmente nel caso essi evolvano o se ne aggiungano di nuovi;
- Essere comprensibile, modulare e robusta riuscendo a gestire situazioni erronee improvvise;
- Essere affidabile, cioè avere una elevata capacità di rispettare le specifiche nel tempo;
- Essere sicura rispetto ad intrusioni e malfunzionamenti;
- Avere una elevata disponibilità, riducendo al minimo i tempi di manutenzione;
- Avere componenti semplici, coesi nel raggiungere gli obiettivi, incapsulati e con scarse dipendenze tra loro.

2.2.2.2 Diagrammi

Al fine di rendere più chiare le scelte progettuali adottate e ridurre le possibili ambiguità, sarà necessario fare largo uso di vari tipi di diagrammi **UML 2.0**, realizzandone secondo le seguenti rappresentazioni:

- Diagrammi dei casi d'uso, dedicati alla descrizione delle funzioni offerte dal sistema.

2.2.2.3 Obiettivi della progettazione

Essa serve a garantire che il prodotto sviluppato soddisfi le proprietà e i bisogni specificati nell'attività di analisi ponendo i seguenti obiettivi:

- Garantire la qualità di prodotto sviluppato, perseguendo la *correttezza per costruzione*;
- Organizzare e ripartire compiti implementativi, riducendo la complessità del problema originale fino alle singole componenti facilitandone la codifica da parte dei singoli Programmatori;
- Ottimizzare l'uso di risorse.

2.2.3 Codifica

In questa sotto-sezione vengono elencate le norme alle quali i Programmatori devono attenersi durante l'attività di programmazione e implementazione.
All'inizio verranno elencate delle norme generali a cui i Programmatori devono attenersi con qualsiasi linguaggio di programmazione utilizzato, mentre di seguito verranno elencate delle norme specifiche per i linguaggi JavaScript, JavaScript Syntax eXtension, Solidity e SCSS.
Ogni norma è rappresentata da un paragrafo; ciascuna ha un titolo, una breve descrizione e, se necessario, un esempio che illustri le convenzioni da applicare. Alcune di esse includono inoltre una lista di possibili eccezioni d'uso.
L'uso di norme e convenzioni è fondamentale per permettere la generazione di codice leggibile e uniforme, agevolando così le fasi di manutenzione, verifica e validazione e migliorando la qualità del prodotto.

Convenzioni per i nomi:

- Alcuni nomi sono da evitare in quanto facilmente confondibili con i numeri `1` e `0`, elencati di seguito:
  - `l` (lettera minuscola elle);
  - `O` (lettera maiuscola o);
  - `I` (lettera maiuscola i).
- Tutti i nomi devono essere **unici** ed **esplicativi** al fine di evitare al più possibile ambiguità e comprensione.

Per i vari linguaggi verranno successivamente descritte altre norme per nomi di variabili, funzioni e altro facendo riferimento ai seguenti stili:

- `lowercase`;
- `lower_case_with_underscores`

- `UPPERCASE`
- `UPPER_CASE_WITH_UNDERSCORES`
- `CapitalizedWords` (o `CapWords`)
- `mixedCase`
- `Capitalized_Words_With_Underscores`
- `lower-case-with-dashes`

Convenzioni per la documentazione:

- Tutti i nomi e i commenti al codice per la documentazione vanno scritti in **inglese**;
- È possibile utilizzare in un commento la keyword **TODO** per indicare codice temporaneo e soluzioni a breve termine o evidentemente migliorabili;
- Ogni file contenente codice deve avere la seguente **intestazione** contenuta in un commento e posta all'inizio del file stesso:

```
File : nome file
Version : versione file
Type : tipo file
Date : data di creazione
Author : nome autore/i
E- mail : email autore/i

License : tipo licenza

Advice : lista avvertenze e limitazioni

Changelog :
Autore || Data || breve descrizione delle modifiche
```

- La **versione** del codice viene inserita all'interno dell'intestazione del file e rispetta il seguente formalismo:

X.Y

- **X**: è l'indice di versione principale, un incremento di tale indice rappresenta un avanzamento della versione stabile, che porta il valore dell'indice Y ad essere azzerato;
- **Y**: è l'indice di modifica parziale, un incremento di tale indice rappresenta una verifica o una modifica rilevante, come per esempio la rimozione o l'aggiunta di una istruzione.

La versione *1.0* deve rappresentare la prima versione del file completo e stabile, cioè quando le sue funzionalità obbligatorie sono state definite e si considerano funzionanti. Solo dalla versione *1.0* è possibile testare il file, con degli appositi test predefiniti, per verificarne l'effettivo funzionamento.

2.2.3.1 JavaScript

In questa sotto-sezione vengono elencate le norme tratte dal **Airbnb JavaScript Style Guide**[1]; il controllo delle norme è implementato direttamente all'interno dell'IDE Webstorm tramite una apposita procedura seguita da tutti i membri del team[2].
La versione dello standard ECMA seguito è la 2017[3].

Indentazione 1: vanno utilizzati due (2) spazi per ogni livello di indentazione.

SI

```
function() {
..let name;
}
```

NO

```
function() {
....let name;
}
```

Indentazione 2: bisogna mettere uno (1) spazio prima della graffa principale.

SI

```
function() {
..let name;
}
```

NO

```
function(){
..let name;
}
```

[1] https://github.com/airbnb/javascript
[2] https://github.com/doasync/eslint-config-airbnb-standard
[3] https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

Indentazione 3: va inserito uno (1) spazio prima della parentesi di apertura degli statement di controllo (`if`, `while` etc.).
Non va inserito alcuno spazio prima della lista degli argomenti nelle chiamate di funzioni e nelle dichiarazioni.

SI

```
function fight() {
  console.log('353');
}
```

NO

```
function fight () {
  console.log ('353');
}
```

Spazi: è vietato l'utilizzo di tabulazioni, che devono essere necessariamente sostituite da spazi. Al fine di assicurare il rispetto di questa regola si consiglia di configurare adeguatamente il proprio editor o IDE.

Capo riga: tutti i capo riga dovranno essere effettuati mediante l'uso di LF (line feed, 0xA), come definito dallo standard UNIX, CRLF (carriage return line feed, 0xDA) non è ammissibile. Gli utenti che utilizzano il sistema operativo Microsoft Windows dovranno fare attenzione ad impostare correttamente l'editor Webstorm seguendo la guida ufficiale[4].

Punto e virgola: non è stata utilizzata la nuova funzionalità "ASI" o "Automatic Semicolon Insertion" in quanto introdotta nello standard ECMA 2018, mentre questo progetto fa riferimento allo standard 2017.
L'introduzione di ASI consentirebbe l'omissione del punto e virgola alla fine di una dichiarazione nel caso l'interprete riesca a dedurre autonomamente la fine dell'istruzione.

Linee vuote: bisogna lasciare una riga vuota dopo blocchi e prima di un nuovo statement.

[4] https://www.jetbrains.com/help/webstorm/configuring-line-separators.html

SI

```
if (foo) {
  return bar;
}

return baz;

//oppure

var obj = {
  foo: function() {
  },

  bar: function() {
  }
};
```

NO

```
if (foo) {
  return bar;
 }
return baz;

//oppure

var obj = {
  foo: function() {
  },
  bar: function() {
  }
};
return obj;
```

Parentesizzazione 1: i blocchi di codice multi-riga vanno racchiusi tra parentesi graffe. I blocchi di codice con una sola riga possono essere scritti senza parentesi, ma devono essere scritti sulla stessa riga.

SI

```
if (test) return false;

if (test) {
  return false;
}

function() {
  return false;
}
```

NO

```
if (test)
  return false;

function() { return false; }
```

Parentesizzazione 2: le parantesi graffe devono iniziare nella stessa riga del codice, non in quella sottostante.

SI

```
if (test) {
  \\..
  if(test2) {
  \\..
    if(test3) {
          \\..
    }
  }
}
```

NO

```
if (test)
{
  if(test2)
  {
    if(test3)
    {
      ...
    }
  }
}
```

Parentesizzazione 3: in caso di blocchi multi-riga con `if` e `else`, è necessario mettere `else` nella stessa riga della parentesi graffa che chiude il blocco `if`.

SI

```
if (test) {
  thing1();
  thing2();
} else {
  thing3();
}
```

NO

```
if (test) {
  thing1();
  thing2();
}
  else {
  thing3();
}
```

Commenti: utilizzare `//` per una singola riga di commento, la quale deve essere inserita su una nuova riga sopra il soggetto del commento. Inoltre bisogna lasciare la riga precedente al commento vuota.

SI

```
// comment
var active = true;

function getType() {
  console.log('353');

  // comment
  var type = this._type;

  return type;
}
```

NO

```
var active = true; // comment

function getType() {
  console.log('353');
  // comment
  var type = this._type;

  return type;
}
```

Variabili 1: una variabile deve sempre essere dichiarata usando `var` o `let`. In caso contrario le variabili dichiarate avranno una visibilità globale, pratica da evitare assolutamente per non inquinare lo spazio dei nomi.

SI

```
var superPower = new SuperPower();
```

NO

```
superPower = new SuperPower();
```

Variabili 2: utilizzare una sola dichiarazione `var` o `let` per ogni variabile. È più facile aggiungere dichiarazioni di variabili in questa maniera e non si corre il rischio di scambiare un punto e virgola `";"` con una virgola `","`. Inoltre, le variabili non inizializzate immediatamente vanno sempre dichiarate per ultime.

SI

```
var objects = getItems();
var cond = true;
var stranger = 'things';
var notAssign;
```

NO

```
var objects = getItems(),
        cond = true,
        notAssign,
        stranger = 'things';
```

Variabili 3: le variabili vanno assegnate o dichiarate solo quando si ha la necessità di usarle, in modo da evitare operazioni inutili.

SI

```
function cName(hasName) {
  if (hasName === 'test') {
    return false;
  }

  const name = getName();

  if (name === 'test') {
    this.setName('');
    return false;
  }

  return name;
}
```

NO

```
function cName(hasName) {
  const name = getName();

  if (hasName === 'test') {
    return false;
  }

  if (name === 'test') {
    this.setName('');
    return false;
  }

  return name;
}
```

Nomi 1: vietato utilizzare nomi delle variabili o delle funzioni con una singola lettera.

SI

```
function query() {
  // ...
}
```

NO

```
function q() {
  // ...
}
```

Nomi 2: utilizzare lo stile `mixedCase` per i nomi delle variabili, delle funzioni e delle istanze. Da evitare l'utilizzo degli underscore nei nomi.

SI

```
const thisIsMyObject = {};
var myVar = "Hello";
function thisIsMyFunction() {}
```

NO

```
const OBJEcttssss = {};
const this_is_my_object = {};
var _myVar_= "Hello";
function c() {}
```

Nomi 3: utilizzare lo stile `CapWords` per i nomi delle classi.

SI

```
class User {
  constructor(options) {
    this.name = options.name;
  }
}
```

NO

```
class user {
  constructor(options) {
    this.name = options.name;
  }
}
```

2.2.3.2 JavaScript Syntax eXtension

La scrittura di codice in JavaScript Syntax eXtension, comunemente abbreviato in JSX, deve seguire tutte le norme descritte precedentemente per il linguaggio di programmazione JavaScript.
In aggiunta sono elencate ulteriori norme riguardanti esclusivamente JSX, tratte da **Airbnb React/JSX Style Guide**[5].

Indentazione 1: va inserito sempre uno (1) spazio nei tag di chiusura.

SI

```
<Foo />
```

NO

```
<Foo/>
```

Proprietà 1: utilizzare sempre lo stile `mixedCase` per i nomi delle proprietà di un tag.

SI

```
<Foo
  userName="hello"
  phoneNumber={12345678}
/>
```

NO

```
<Foo
  UserName="hello"
  phone_number={12345678}
/>
```

Proprietà 2: il valore di una proprietà va omesso quando è implicitamente vero (`true`).

SI

```
<Foo
  hidden
/>
```

NO

```
<Foo
  hidden={true}
/>
```

[5] https://github.com/airbnb/javascript/tree/master/react

I tag 1: si deve chiudere tutti i tag che non hanno i tag figli o altro contenuto senza utilizzare il relativo tag di chiusura.

SI

```
<Foo variant="stuff" />
```

NO

```
<Foo variant="stuff"></Foo>
```

I tag 2: è vietato chiudere un tag con più di una proprietà sulla stessa riga.

SI

```
<Foo
  bar="bar"
  baz="baz"
/>
```

NO

```
<Foo
  bar="bar"
  baz="baz" />
```

r

2.2.3.3 Solidity

In questa sotto-sezione vengono elencate le norme tratte dalla **documentazione officiale di Solidity**[6], la cui applicazione viene favorita tramite l'utilizzo di determinati plugin[7] di supporto.

Identazione: vanno utilizzati quattro (4) spazi per ogni livello di indentazione.

Spazi: è vietato l'utilizzo di tabulazioni, che devono essere necessariamente sostituite da spazi. Al fine di assicurare il rispetto di questa regola si consiglia di configurare adeguatamente il proprio editor o IDE.

Linee vuote 1: devono essere lasciate due (2) linee vuote tra la dichiarazione di più smart contract.

[6] http://solidity.readthedocs.io/en/develop/style-guide.html
[7] https://github.com/intellij-solidity/intellij-solidity

SI

```
contract A {
    ...
}


contract B {
    ...
}


contract C {
    ...
}
```

NO

```
contract A {
    ...
}
contract B {
    ...
}

contract C {
    ...
}
```

Linee vuote 2: le linee vuote possono essere omesse tra le dichiarazioni di funzioni in linea.

SI

```
contract A {
    function spam();
    function ham();
    }


contract B is A {
    function spam() {
        ...
    }

    function ham() {
        ...
    }
}
```

NO

```
contract A {
    function spam() {
        ...
    }
    function ham() {
        ...
    }
}
```

Codifica dei codici sorgenti: i codici sorgente devono essere codificati in UTF-8.

Imports: tutte le dichiarazioni di `import` devono essere effettuate solamente all'inizio del file.

SI

```
import "owned";


contract A {
    ...
}


contract B is owned {
    ...
}
```

NO

```
contract A {
    ...
}


import "owned";


contract B is owned {
    ...
}
```

Ordine delle funzioni: all'interno di un contratto l'ordine è fondamentale per identificare meglio quali funzionalità sono disponibili e per ricercare più velocemente le definizioni del costruttore e di eventuali funzioni di fallback.
Le funzioni devono essere raggruppate secondo la loro visibilità e ordinate nel seguente modo:

1. costruttore
2. funzioni di fallback (se esistono)
3. esterne
4. pubbliche
5. interne
6. private

In ogni gruppo le funzioni costanti vanno dichiarate per ultime.

SI

```
contract A {
    function A() {
        ...
    }

    function() {
        ...
    }

    // External functions
    // ...

    // External functions
    // that are constant
    // ...

    // Public functions
    // ...

    // Internal functions
    // ...

    // Private functions
    // ...
}
```

NO

```
contract A {

    // External functions
    // ...

    // Private functions
    // ...

    // Public functions
    // ...

    function A() {
        ...
    }

    function() {
        ...
    }

    // Internal functions
    // ...
}
```

Spazi bianchi nelle espressioni: non vanno messi spazi bianchi nelle seguenti situazioni:

- immediatamente dopo l'apertura o la chiusura di una parentesi tonda, quadrata o graffa con un'espressione di una singola riga

SI

```
spam(ham[1], Coin({name: "ham"}));
```

NO

```
spam( ham[ 1 ], Coin( { name: "ham" } ) );
```

ECCEZIONE

```
function singleLine() { spam(); }
```

- immediatamente prima di una virgola o di un punto e virgola

SI

```
function spam(uint i, Coin coin);
```

NO

```
function spam(uint i , Coin coin) ;
```

- prima o dopo un'assegnazione o con altri operatori per permettere l'allineamento con altre istruzioni

SI

```
x = 1;
y = 2;
long_variable = 3;
```

NO

```
x             = 1;
y             = 2;
long_variable = 3;
```

- nella dichiarazione di funzioni fallback

SI

```
function() {
    ...
}
```

NO

```
function () {
    ...
}
```

Strutture di controllo: le parentesi graffe che denotano l'apertura di un contratto, una libreria o una funzione devono:

- essere aperte nella stessa riga della dichiarazione e precedute da uno spazio bianco;
- chiuse nello stesso livello di indentazione della dichiarazione iniziale.

SI

```
contract Coin {
    struct Bank {
        address owner;
        uint balance;
    }
}
```

NO

```
contract Coin
{
    struct Bank {
        address owner;
        uint balance;
    }
}
```

Le stesse regole valgono anche per strutture di controllo come `if`, `else`, `while` e `for`.
Inoltre, deve essere messo un singolo spazio tra la dichiarazione della struttura di controllo `if`, `while` e `for` e la condizione e anche tra quest'ultima e l'apertura della parentesi graffa.

SI

```
if (...) {
    ...
}

for (...) {
    ...
}
```

NO

```
if (...)
{
    ...
}

while(...){
}

for (...) {
    ...;}
```

Per le strutture di controllo che nel corpo hanno una singola istruzione è possibile omettere le parentesi graffe solo se l'istruzione è contenuta in una singola linea.

SI

```
if (x < 10)
    x += 1;
```

NO

```
if (x < 10)
    someArray.push(Coin({
        name: 'spam',
        value: 42
    }));
```

È consentita una sola eccezione: nei blocchi di `if` che contengono clausole di `else` o `else if`, l'`else` deve stare nella stessa linea della chiusura del blocco `if`.

SI

```
if (x < 3) {
    x += 1;
} else if (x > 7) {
    x -= 1;
} else {
    x = 5;
}


if (x < 3)
    x += 1;
else
    x -= 1;
```

NO

```
if (x < 3) {
    x += 1;
}
else {
    x -= 1;
}
```

Dichiarazione di funzioni: le dichiarazioni brevi devono presentare uno spazio bianco prima dell'apertura della graffa che racchiude il codice.
La graffa che chiude il corpo della funzione va chiusa allo stesso livello di indentazione della dichiarazione iniziale.

SI

```
function increment(uint x) returns (uint) {
    return x + 1;
}

function increment(uint x) public onlyowner returns (uint) {
    return x + 1;
}
```

NO

```
function increment(uint x) returns (uint)
{
    return x + 1;
}

function increment(uint x) returns (uint){
    return x + 1;
}

function increment(uint x) returns (uint) {
    return x + 1;
}

function increment(uint x) returns (uint) {
    return x + 1;}
```

La visibilità di una funzione va dichiarata prima di altri modificatori.

SI

```
function kill() public onlyowner {
    selfdestruct(owner);
}
```

NO

```
function kill() onlyowner public {
    selfdestruct(owner);
}
```

Per dichiarazioni lunghe, ogni argomento va dichiarato in una linea allo stesso livello di indentazione del corpo della funzione. La parentesi di chiusura della dichiarazione degli argomenti va collocata allo stesso livello di indentazione della dichiarazione della funzione.

SI

```
function thisFunctionHasLotsOfArguments(
    address a,
    address b,
    address c,
    address d,
    address e,
    address f
) {
    doSomething();
}
```

NO

```
function thisFunctionHasLotsOfArguments(address a, address b,
    address c, address d, address e, address f) {
    doSomething();
}

function thisFunctionHasLotsOfArguments(address a,
                                        address b,
                                        address c,
                                        address d,
                                        address e,
                                        address f) {
    doSomething();
}

function thisFunctionHasLotsOfArguments(
    address a,
    address b,
    address c,
    address d,
    address e,
    address f) {
    doSomething();
}
```

Se una lunga dichiarazione ha molti modificatori, ciascuno di essi deve essere collocato in una nuova linea.

SI

```
function thisFunctionNameIsReallyLong(address x, address y)
    public
    onlyowner
    priced
    returns (address)
{
    doSomething();
}

function thisFunctionNameIsReallyLong(
    address x,
    address y,
    address z
)
    public
    onlyowner
    priced
    returns (address)
{
    doSomething();
}
```

NO

```
function thisFunctionNameIsReallyLong(address x, address y)
                                      public
                                      onlyowner
                                      priced
                                      returns (address) {
    doSomething();
}

function thisFunctionNameIsReallyLong(address x, address y)
    public onlyowner priced returns (address)
{
    doSomething();
}

function thisFunctionNameIsReallyLong(address x, address y)
    public
    onlyowner
    priced
    returns (address) {
    doSomething();
}
```

Per costruttori o per contratti estesi che richiedono degli argomenti, i costruttori di base vanno dichiarati uno per linea.

SI

```
contract A is B, C {
    function A(uint param1, uint param2, uint param3, uint param4)
        B(param1)
        C(param2, param3)
    {
        // do something with param4
    }
}
```

NO

```
contract A is B, C {
    function A(uint param1, uint param2, uint param3, uint param4)
    B(param1)
    C(param2, param3)
    {
        // do something with param4
    }
}

contract A is B, C {
    function A(uint param1, uint param2, uint param3, uint param4)
        B(param1)
        C(param2, param3) {
       // do something with param4
    }
}
```

È possibile dichiarare funzioni in una sola linea solo se possiedono una singola istruzione.

```
function shortFunction() { doSomething(); }
```

Dichiarazione di variabili: nella dichiarazione di array non devono essere messi spazi tra il tipo e le parentesi quadrate.

SI

```
uint[] x;
```

NO

```
uint [] x;
```

Le stringhe vanno dichiarate solo utilizzando i doppi apici.

SI

```
str = "foo";
str = "Hamlet says, 'To be or not to be...'";
```

NO

```
str = 'bar';
str = '"Be yourself; everyone else is already taken." -O.W.';
```

Operatori: lasciare uno spazio tra un operatore e una variabile.

SI

```
x = 3;
x = 100 / 10;
x += 3 + 4;
x |= y && z;
```

NO

```
x=3;
x = 100/10;
x += 3+4;
x |= y&&z;
```

Per una maggiore leggibilità e comprensione del codice si può avere come eccezione l'assenza di determinati spazi in istruzioni composte da più operazioni, in modo tale da evidenziare l'operazione con priorità maggiore.

SI

```
x = 2**3 + 5;
x = 2*y + 3*z;
x = (a+b) * (a-b);
```

NO

```
x = 2** 3 + 5;
x = y+z;
x +=1;
```

Convenzioni per i nomi:

- contratti e librerie: seguono il `CapWords` style.
  es. `SimpleToken`, `SmartBank`, `CertificateHashRepository`, `Player`;
- eventi: seguono il `CapWords` style.
  es. `Deposit`, `Transfer`, `Approval`, `BeforeTransfer`, `AfterTransfer`;
- funzioni: seguono il `mixedCase` style.
  es. `getBalance`, `transfer`, `verifyOwner`, `addMember`, `changeOwner`;
- argomenti di funzioni: seguono il `mixedCase` style.
  es. `initialSupply`, `account`, `recipientAddress`, `senderAddress`, `newOwner`;
- variabili locali: seguono il `mixedCase` style.
  es. `totalSupply`, `remainingSupply`, `balanceOf`, `creatorAddress`, `isPreSale`;
- costanti: seguono il `CAPITAL_CASE_WITH_UNDERSCORE` style.
  es. `MAX_BLOCKS`, `TOKEN_NAME`, `TOKE_TICKET`;
- modificatori: seguono il `mixedCase` style.
  es. `onlyBy`, `onlyAfter`, `onlyDuringThePreSale`;

2.2.3.4 SCSS

In questa sotto-sezione vengono elencate le norme tratte dalla **Airbnb CSS/Scss Styleguide**[8], il cui utilizzo viene supportato tramite plugin Webstorm scss-lint[9] utilizzando la corretta configurazione[10] ed il tool scss-lint[11].

Formattazione:

- Utilizzare due (2) spazi per ogni livello di indentazione;
- Per i nomi delle classi utilizzare la `lower-case-with-dashes`;
- Non usare selettori ID;
- In caso di selettori multipli per una singola regola, inserire ogni selettore in una linea singola;
- Lasciare uno spazio prima dell'apertura della parentesi graffa di una regola;
- Nelle proprietà lasciare uno spazio dopo e non prima i due punti;
- Chiudere la parentesi graffa di una regola in una nuova linea;
- Lasciare una linea vuota tra una dichiarazione e l'altra;
- Dichiarare le classi raggruppate per tipologia e in ordine alfabetico.

SI

```
.avatar {
  border-radius: 50%;
  border: 2px solid white;
}

.one,
.per-line,
.selector
 {
  // ...
}
```

NO

```
.wrong {
  // ...}
.avatar{
  border-radius:50%;
  border:2px solid white; }
.no, .nope, .not_good {
  // ...
}
#lol-no {
  // ...
}
```

[8] https://github.com/airbnb/css
[9] https://plugins.jetbrains.com/plugin/7530-scss-lint
[10] https://github.com/airbnb/css/blob/master/.scss-lint.yml
[11] https://github.com/brigade/scss-lint

BEM: utilizzare la notazione BEM (Block-Element-Modifier) come convenzione per i nomi delle classi.

```
// ListingCard.jsx
function ListingCard() {
  return (
    <article class="listingcard listingcard-featured">

      <h1 class="listingcard-title">Adorable Mission</h1>

      <div class="listingcard-content">
        <p>Vestibulum id ligula porta euismod semper.</p>
      </div>

    </article>
  );
}
```

```
/* listingcard.css */
.listingcard { }
.listingcard-featured { }
.listingcard-title { }
.listingcard-content { }
```

In questo esempio:

- `.listingcard` è il 'blocco' e rappresenta il livello più alto del componente;
- `.listingcard-title` è un 'elemento' e rappresenta un discendente di `.listingcard`;
- `.listingcard-featured` è un 'modificatore' e rappresenta un diverso stato o una variante del blocco `.listingcard`;

Commenti:

- Non usare commenti di blocco;
- Fare commenti in linee a se stanti, non in linee contenenti anche codice.

Ordine di dichiarazione delle proprietà:

1. **Proprietà standard:** vanno dichiarate per prime e in ordine alfabetico;

```
.btn-green {
  background: green;
  font-weight: bold;
  // ...
}
```

2. **Dichiarazioni di `@include`:** vanno raggruppate sotto alla dichiarazione delle proprietà standard

```
.btn-green {
  background: green;
  font-weight: bold;
  @include transition(background 0.5s ease);
  // ...
}
```

3. **Selettori nidificati:** vanno dichiarati dopo tutte le altre dichiarazioni e deve essere lasciata una linea vuota prima della loro dichiarazione

```
.btn {
  background: green;
  font-weight: bold;
  @include transition(background 0.5s ease);

  .icon {
    margin-right: 10px;
  }
}
```

Extend: la direttiva `@extend` non va utilizzata perché ha un comportamento poco intuitivo e pericoloso, specialmente quando usata in selettori nidificati.

Selettori annidati: non utilizzare più di tre livelli di annidamento

```
.page-container {
  .content {
    .profile {
      // STOP!
    }
  }
}
```

2.2.4 Procedure

2.2.4.1 Tracciamento componenti-requisiti

Si è scelto di utilizzare il software Trender per il tracciamento automatizzato di tutte le informazioni ricavate dall'analisi dei requisiti$_G$.
Il database MySQL che garantisce la persistenza e l'accesso ai dati si trova in remoto, in modo tale da rendere sempre disponibile a tutti le informazioni che esso contiene.
Qualsiasi componente del gruppo, utilizzando la parte front-end dell'applicazione scritta in PHP e JavaScript, può accedervi utilizzando l'account comune predisposto.
Una volta effettuato l'accesso si ha a disposizione le funzionalità di modifica e di esportazione in codice LaTeX dei dati presenti.

2.2.5 Strumenti

- **Tracciamento requisiti:** il gruppo 353 ha deciso di utilizzare il software **Trender**[12] installato su server remoto condiviso;
- **Creazione diagrammi UML:** UML Designer[13] ;
- **Scrittura del codice:** Webstorm by Jetbrains[14];
- **Toolkit sviluppo:** si utilizzano i pacchetti installabili tramite Node.js con gestore di pacchetti npm per risolvere dipendenze delle librerie e tool richiesti;
- **Tracciamento dei test:** Codecov[15];
- **Esecuzione dei test:**

[12] https://github.com/campagna91/Trender
[13] http://www.umldesigner.org/
[14] https://www.jetbrains.com/webstorm/
[15] https://codecov.io/

  - React: Mocha[16] e Enzyme[17];
  - Redux: Jest[18];
  - Web3/Truffle: Mocha e Chai[19].
- **Continuos Integration:** Travis[20].
- **Tool di supporto:** Scss-lint[21].

[16] https://github.com/mochajs/mocha
[17] https://github.com/airbnb/enzyme
[18] https://facebook.github.io/jest/
[19] http://chaijs.com/
[20] http://travis-ci.com/
[21] https://github.com/brigade/scss-lint

3. Processi di supporto

3.1 Documentazione

3.1.1 Descrizione

Questo capitolo descrive le scelte intraprese per la stesura, verifica e approvazione riguardante la documentazione ufficiale. Tali norme sono tassative per tutti i documenti formali. Tali documenti sono elencati successivamente nella sotto-sezione "Documenti correnti".

3.1.2 Ciclo di vita documentazione

Ogni documento formale deve passare gli stadi di "Sviluppo", "Verifica" e "Approvato".

- **Sviluppo:** inizia con la creazione del documento e termina con la conclusione della stesura di tutte le sue parti. In questa fase i Redattori aggiungono le parti assegnate tramite ticket. Il passaggio alla fase di Verifica è automatizzato con la segnalazione al Responsabile;
- **Verifica:** il documento entra nella fase di verifica dopo l'assegnazione da parte del Responsabile. I Verificatori effettueranno le procedure di controllo dello stesso elencate nella sezione
  Al termine del controllo in caso positivo il documento entra automaticamente in fase di "Approvato", altrimenti i loro riscontri vengono consegnati al Responsabile di Progetto, che provvederà ad assegnare nuovamente il documento ad un Redattore attraverso una nuova fase di Sviluppo;
- **Approvato:** l'approvazione di un documento coincide con il superamento positivo da parte di un Verificatore dello stadio di Verifica e diventa una versione ufficiale.

3.1.3 Separazione documenti interni ed esterni

Ogni documento formale dovrà essere classificato come documento Interno oppure Esterno con le seguenti differenze:

- **Interno:** ha utilizzo interno al gruppo 353, redatto in lingua Italiana;
- **Esterno:** verrà condiviso con la Proponente ed i Committenti, nel caso sia utile per il deploy dell'applicazione web o del suo utilizzo da parte degli utenti deve essere redatto in lingua Inglese.

Un documento formale farà sempre parte di una delle due categorie elencate.

3.1.4 Nomenclatura documenti

Tutti i documenti formali tranne "..." seguiranno questo sistema di nomenclatura:

- **NomeDocumento:** indica il nome del documento senza spazi;
- **vX.Y.Z:** indica il numero di versionamento con X,Y e Z numeri interi e non negativi.

Di seguito la spiegazione che assumono le versione del documento:

- **X:** rappresenta il numero di pubblicazioni formali del documento, ogni qual volta il documento verrà pubblicato il valore di Y e Z verrà azzerato;
- **Y:** rappresenta il numero di verifiche completate con successo, in caso di mancato superamento della fase di verifica il valore non va modificato, mentre se superata viene incrementato di uno e si azzera il valore della Z;
- **Z:** rappresenta il numero di modifiche effettuate al documento durante il suo sviluppo.

Formato dei file: ogni documento si trova nel formato .tex durante il suo ciclo di vita. Dopo lo stato di "Approvato" per il documento viene quindi creato un PDF contenente la versione approvata dal Responsabile.

3.1.5 Documenti correnti

Qui di seguito si presentano i documenti formali in ordine alfabetico e la loro classificazione tra Interno od Esterno:

- **Analisi dei Requisiti:** utilizzo Esterno, sigla(AR)
  Documento per esporre e scomporre i requisiti$_G$ del progetto contenente casi d'uso relativi al prodotto e diagrammi di interazione con l'utente. Viene scritto dagli Analisti dopo aver analizzato il capitolato e interagendo con il Proponente in riunioni esterne;

- **Glossario:** utilizzo Esterno, sigla (GL)
  Documento per raccogliere le definizioni dei termini o concetti che saranno usati nei documenti formali per facilitarne la comprensione.

- **Norme di Progetto:** utilizzo Interno, sigla (NdP)
  Documento per mostrare le direttive e gli standard utilizzati all'interno del gruppo di lavoro 353 per lo sviluppo del progetto.

- **Piano di Progetto:** utilizzo Esterno, sigla (PdP)
  Documento per l'analisi e la pianificazione della gestione delle risorse di tempo e delle risorse umane.

- **Piano di Qualifica:** utilizzo Esterno, sigla (PdQ)
  Documento per descrivere gli standard e gli obiettivi di qualità che il gruppo vorrà raggiungere per garantire la qualità di prodotto e di processo.

- **Studio di Fattibilità:** utilizzo Interno, sigla (SdF)
  Documento per indicare le riflessioni, punti di forza e caratteristiche sfavorevoli per ogni capitolato che ha portato alla scelta finale del gruppo.

3.1.6 Norme

3.1.6.1 Struttura dei documenti

Ogni documento è realizzato a partire da una disposizione prestabilita che dovrà essere conforme per ogni documento ufficiale ad eccezione dei verbali:

- **Frontespizio:** questa sezione si troverà nella prima pagina di ogni documento e conterrà:
  - Logo del gruppo;
  - Titolo del documento;
  - Versione del documento con l'ultima data di modifica;
  - Nome del gruppo;
  - Nome del progetto.

- **Informazioni sul documento:** conterrà la lista di responsabili, redattori, verificatori del documento e infine lo stato e la tipologia di uso vedi sotto sezione Separazione documenti interni ed esterni
- **Diario delle modifiche:** Questo diario sarà presente nella seconda pagina del documento sotto forma di tabella in ordine di versione decrescente con righe composte da: versione, data, descrizione modifiche, autore, ruolo
- **Indice delle sezioni:** L'indice delle sezioni conterrà l'indice di tutti gli argomenti trattati nel documento con la seguente struttura: titolo, argomento e numero pagina
- **Indice delle tabelle:** Sezione contenente l'indice delle tabelle con struttura identica alle sezioni.
- **Indice delle figure:** Sezione contenente l'indice delle figure con struttura identica alle sezioni.
- **Introduzione:** scopo del documento, info glossario e riferimenti utili?
- **Contenuto del documento:** il resto del documento è occupato dal contenuto.

3.1.6.2 Norme tipografiche

- **Intestazione** ogni pagina dopo frontespizio presenterà sulla sinistra il logo del gruppo e a destra il nome del capitolo corrente;
- **Piè pagina:** a sinistra è presente il nome del documento corrente e a destra il numero di pagina;
- **Virgolette:** alte singole ' ' per singolo carattere, alte doppie " " per racchiudere stringhe mentre quelle basse '<<' '>>' per racchiudere citazioni;
- **Parentesi:** tonde per descrivere esempi o fornire sinonimi o precisazioni mentre quelle quadre sono usate per rappresentare uno standard ISO, uno stato relativo a un ticket o un riferimento ad un codice definito all'interno del documento stesso;
- **Punteggiatura:** ogni segno di punteggiatura deve essere seguito da uno spazio e non avere spazi precedenti al segno stesso;
- **Stile del testo:**

  - Corsivo: Per dare enfasi ad una parola, un concetto o per indicare il nome di un tool/tecnologia;
  - Grassetto: Per i titoli, sottotitoli ed elementi di elenchi e liste;
  - Sottolineato: Per indicare dei collegamenti ipertestuali.

- **Elenchi:** ogni elenco avrà la prima parola di ogni elemento maiuscola seguita dal carattere 'due punti' (:) seguito dalla descrizione dell'elemento, mentre al termine dell'elemento si inserirà sempre sempre i carattere 'punto e virgola' (;) tranne per l'ultimo elemento della lista per cui si userà il carattere 'punto' (.);

- **Note a Piè pagina:** seguono le seguenti regole:

  - Devono presentare una numerazione progressiva all'interno del documento
  - Devono essere scritte solo una volta
  - Il primo carattere di ogni nota deve essere maiuscolo. Fanno eccezione i casi in cui la parola sia un acronimo.

- **Formati:**

  - Date: scritte con lo standard DD-MM-YYYY dove YYYY indica l'anno, MM il mese e DD il giorno, inseribili nel formato corretto attraverso il comando \nData;
  - Grassetto: lo stile grassetto va utilizzato per i titoli dei paragrafi e per i titoli degli elementi di un elenco;
  - URI: lo stile utilizzato per un URI è il corsivo di colore blu, in modo da mantenere una continuità con lo standard web attuale, utilizzabile col comando personalizzato latex \nURI;

- **Ruoli/Fasi/Revisioni di Progetto**

- **Nomi:** sono stati realizzati dei comandi personalizzati per poter richiamare la visualizzazione dei seguenti termini:

  - nome gruppo: \gruppo, visualizza "*353*"
  - nomi propri: \NomeProprioPersona
  - nome progetto: \progetto, visualizza "*Marvin*"
  - nome di un file: \nFile
  - nome di un documento: \nDoc

  - percorso cartelle: \nPath

- **Componenti grafiche:**
  - immagini: i formati ammessi sono PNG e PDF
  - tabelle: devono rispettare lo stile del template realizzato

3.1.7 Struttura documentazione

E' stato creato un template di documento utilizzabile per tutti i documenti ufficiali in modo tale da favorire lo sviluppo dei documenti.
Il template è basato sulle norme di documentazione elencate nella sezione precedente, inoltre è stata scritta una pagina di showcase per facilitare il mantenimento corretto delle strutture dei documenti.

3.1.8 Gestione termini Glossario

Il glossario è un documento unico per tutti i documenti, esso conterrà tutte le definizioni, in ordine lessicografico crescente dei termini inerenti al tema del progetto o che possono essere fraintesi o non ben comprensibili. I termini che dovranno essere inseriti nel glossario saranno contrassegnati da una G pedice all'interno dei documenti.
Ovviamente prima di inserire un nuovo termine bisognerà assicurarsi che non sia già presente, in modo da evitare duplicati.
Il comando LaTeX da utilizzare per contrassegnare un termine da glossario all'interno dei documenti è \citGloss, mentre per l'inserimento di una nuova parola all'interno del glossario viene utilizzato \glossDef.
La scelta di creare un comando apposito per una operazione "elementare" è scaturita dall'agevolazione che porta alla stesura della documentazione: avendo un modo univoco di riconoscere i termini all'interno del glossario è possibile automatizzare la verifica del soddisfacimento delle norme all'interno dei documenti.

3.1.9 Strumenti a supporto della documentazione

La stesura dei documenti deve essere effettuata utilizzando il linguaggio di markup LaTeX utilizzando l'ambiente TeXstudio con dizionario italiano ed inglese installati.

3.2 Qualità

3.2.1 Descrizione

3.2.2 Metriche

- **Metriche per processi:** ogni processo dovrà avere uno standard di qualità elevato definito come unione di:
  - Tempo: richiesto per completamento
  - Risorse: uomo o software richieste
  - Occorrenze: ossia il numero di volte in cui si presenterà un particolare evento come il numero di difetti di una caratteristica di prodotto.
- **Metriche per i documenti:** per la verifica dei documenti è stata scelto l'indice di leggibilità secondo l'indice Gulpease che viene calcolato tramite funzione Latex.
  I risultati sono compresi tra 0 e 100 dove il valore 100 indica una leggibilità massima, si è scelto di considerare un documento sufficientemente leggibile all'ottenimento del valore X;
- **Metriche per il software:** al fine di perseguire gli obiettivi qualitativi prefissati nel Piano Di Qualifica è doveroso definire delle metriche come copertura del codice tramite test automatici;
- **Metriche per i feedback di miglioramento:** i feedback gestiti come spiegato nelle sezioni successive presentano un indice di occorrenze che viene incrementato in automatico ad ogni nuova segnalazione.

3.3 Configurazione

3.3.1 Controllo di versione

3.3.1.1 Descrizione

Per le parti versionabili del progetto e per i documenti ufficiali si è scelto l'utilizzo della tecnologia Git, usando il servizio di hosting di repository$_G$ di GitHub$_G$. La condivisione dei documenti informali e delle parti non versionabili è invece effettuata tramite l'uso di una cartella Google Drive condivisa.

3.3.1.2 Struttura delle repository

E' stata realizzata solo una repository$_G$ durante la fase di RR ossia quella relativa alla documentazione denominata "Documentazione 353", si prevede inoltre di creare ulteriori repository$_G$ per suddividere lo sviluppo e la codifica dell'applicazione del progetto.

I file interni al repository$_G$ "Documentazione 353" sono organizzati secondo questa struttura:

- **Esterni**
  - **AnalisiDeiRequisiti**
  - **Glossario**
  - **PianoDiProgetto**
  - **PianoDiQualifica**
  - **Verbali esterni**
- **Interni**
  - **NormeDiProgetto**
  - **Glossario**
  - **StudioDiFattibilita**
  - **Verbali interni**

All'interno di ogni cartella è stato definito un file LaTex principale che assume il nome del documento stesso ed il file diariomodifiche.tex per mantenere traccia delle modifiche effettuate.
Nella root della repository$_G$ è stato messo invece un file .sh gestito tramite Travis CI che controlla automaticamente gli errori in tutti i documenti presenti e compila un log con i risultati ottenuti.

3.3.1.3 Ciclo di vita dei branch

Per sfruttare il parallelismo nello sviluppo di uno stesso documento sono stati creati appositamente dei branch denominati con il nome del membro o dei membri del gruppo che devono lavorare su una determinata parte, mentre i documenti baseline saranno invece contenuti solamente nel master branch.
Il merge col master avviene quindi solamente quando un documento si trova in stato di "Approvato".

3.3.1.4 Aggiornamento della repository

Per l'aggiornamento della repository$_G$ è previsto il seguente sotto processo motivato dalla sezione precedente "ciclo di vita":

- Verificare di trovarsi sul branch personale con "git branch" (quella selezionata presenta un asterisco *) e in caso di riscontro negativo cambiare branch con "git checkout"
- Dare il comando "git pull". Nel caso in cui si verifichino dei conflitti:
  - Dare il comando "git stash" per accantonare momentaneamente le modifiche apportate;
  - Dare il comando "git pull" per ottenere ed applicare i commit mancanti;
  - Dare il comando "git stash apply" per ripristinare le modifiche.

  In questo modo il repository$_G$ locale risulta aggiornato rispetto il repository$_G$ remoto, mantenendo le modifiche personali apportate;
- Dare il comando "git add $*$" , che aggiungerà i file modificati e quelli nuovi;
- Dare il comando "git commit" e successivamente riassumere le modifiche effettuate, in caso sia utile si può aggiungere un messaggio esteso di descrizione;
- Dare il comando "git push" per completare l'operazione e fornire le modifiche agli altri membri del gruppo.

3.4 Verifica

3.4.1 Descrizione

Un processo fondamentale per il proseguimento e l'evoluzione di un progetto è la verifica su ogni sottoprodotto del progetto che porta alla creazione di un singolo prodotto.
In questa sezione si descriveranno gli strumenti e i metodi che verranno usati per la verifica del codice durante la progettazione.
Per quanto riguarda questa prima parte di progetto la verificati si è concentrata essenzialmente sui documenti e sui diagrammi.

3.4.2 Analisi statica

L'analisi statica è una tecnica di analisi applicabile sia alla documentazione che al codice e permette di effettuare la verifica di quanto prodotto individuando errori ed anomalie.
Essa può essere svolta in due modi diversi:

- **Walkthrough:** tecnica applicata quando non si sanno le tipologie di errori o di problemi che si stanno cercando e quindi prevede una lettura da cima a fondo del codice o documento per trovare anomalie di qualsiasi tipo.
- **Inspection:** tecnica da applicare quando si ha idea delle possibili problematiche che si stanno cercando e si attua leggendo in modo mirato il documento/codice sulla base di una lista di possibili errori precedentemente stilata.

Sono stati utilizzati degli strumenti e sotto processi per velocizzare e rendere più veloce questa analisi con l'utilizzo di funzionalità integrate nell'editor TexStudio oltre all'utilizzo di un sistema di checklist online personale per garantire esecuzione di ogni controllo da parte del Verificatore.

3.4.3 Analisi dinamica

Il processo di analisi dinamica consiste nella realizzazione ed esecuzione di una serie di test sul codice del software.

3.4.4 Verifica Diagrammi UML

I Verificatori devono controllare tutti i diagrammi UML prodotti rispettino lo standard UML e che siano corretti semanticamente.

3.4.5 Strumenti usati per la verifica

- **Software:** verranno usati per la verifica Mocha ed Enzyme per React, Jest per la verifica di Redux mentre per i contratti Solidity useremo Mocha e Chai;
- **Documenti:** Per il controllo dei documenti prodotti utilizzeremo le funzionalità di Texstudio assieme a script bash eseguiti automaticamente da un Bot_{G} per controllare che non ci siano errori nei file LaTex, per assicurarsi che il Glossario non presenti mancanze o duplicati e verificare l'assenza di errori ortografici;

- **Gestione processi e feedback:** è stato scelto di utilizzare il sistema integrato di issues presente su GitHub$_G$ per permettere un dialogo maggiore per ogni singola issue.

Il comando IL (Indice di Leggibilità) ha tre argomenti, nell'ordine: 1) numero lettere 2) numero frasi 3) numero parole

3.5 Validazione

2292.89099
305.18423
45.15555

4. Processi organizzativi

4.1 Processi di cordinamento

4.1.1 Comunicazione

In questa sezione vengono illustrate le norme che regolano la comunicazione sia tra i membri del gruppo 353 che con entità esterne, come i Committenti e i Proponenti.

4.1.1.1 Comunicazioni interne

Le comunicazioni interne al gruppo vengono effettuate tramite Slack$_G$, un'applicazione di messaggistica multi piattaforma con funzionalità specifiche per gruppi di lavoro. La decisione di utilizzare questo strumento è stata supportata dalle funzionalità di integrazione di Bot$_G$ e di creazione di canali tematici, utili per poter classificare le conversazioni e permettere una comunicazione più efficace e mirata. Inoltre è stato deciso di affiancare Skype$_G$ per effettuare video chiamate in caso di impossibilità di riunirsi personalmente. É stato scelto questo metodo poiché immediato e di facile utilizzo.
All'interno di Slack$_G$ sono stati predisposti dei canali tematici, suddivisi per argomento in questo modo:

- **#General**: Per discutere tutto ciò che riguarda l'organizzazione generale del progetto, la scelta degli strumenti di lavoro e per prendere decisioni in modo rapido. Inoltre qui vengono decisi gli argomenti principali da discutere nelle riunioni;
- **#Calendario_meeting**: Il canale riservato nel quale un Bot$_G$ notifica giornalmente gli eventi facenti parte del calendario comune, come riunioni interne, eventi di formazione o riunioni esterne;
- **#Daily_standup**: Il canale dedicato a un Bot$_G$ che giornalmente chiede ai partecipanti notizie sulle attività svolte nel giorno precedente e quelle che sono programmate per il giorno corrente. Inoltre viene chiesto se si

sono verificati problemi nelle attività in corso, così da aiutare il gruppo a organizzarsi e a concentrare gli sforzi su particolari attività;

- **#Github_notifications**: Il canale creato affinché un Bot_G invii una notifica ogni qualvolta un membro del gruppo effettua un operazione su Github;
- **#Gmail353**: Il canale creato affinché un Bot_G invii una notifica ogni qualvolta la casella email del team riceva un messaggio;
- **#Random**: Un canale generico, dove il gruppo discute liberamente di questioni non riguardanti il progetto e il suo sviluppo.

Sono stati creati anche dei canali appositi per gestire meglio la stesura dei documenti, nello specifico:

- **#Analisi_requisiti**: Per commentare i casi d'uso e i requisiti necessari per la stesura del documento *Analisi dei requisiti*;
- **#Piano_progetto**: Per confrontarsi sulla ripartizione dei ruoli e del monte ore a essi associato da includere nel *Piano di progetto*;
- **#Piano_qualifica**: Per discutere riguardo gli obiettivi e le strategie da applicare per garantire qualità e gestire verifica e validazione.

4.1.1.2 Comunicazioni esterne

Questa sottosezione raccoglie le norme che regolano le comunicazioni con soggetti esterni al gruppo 353, i quali sono:

- La proponente **RedBabel** rappresentata da Alessandro Maccagnan e Milo Ertola, con i quali si intende stabilire un rapporto di collaborazione a fine di definire i $\text{requisiti}_\text{G}$ che permetteranno la realizzazione del prodotto;
- **Prof. Tullio Vardanega** e **Prof. Riccardo Cardin**, ai quali verrà fornita la documentazione richiesta in ciascuna revisione di progetto, con i quali si intende dialogare con fine il miglioramento continuo.

Comunicazioni esterne scritte Le comunicazioni esterne scritte vengono effettuate utilizzando l'indirizzo email del gruppo:

353swe@gmail.com

a cui ogni membro del gruppo ha accesso.
Per comunicare con i Proponenti vengono utilizzati gli indirizzi email forniti durante la presentazione dei capitolati, rispettivamente

alessandro@redbabel.com
e
milo@redbabel.com

Inoltre è stato creato nel workspace Slack$_G$ del Proponente RedBabel un canale apposito per la comunicazione tra il gruppo 353 e i rappresentanti del Proponente, utile anche per accordarsi riguardo eventuali incontri e specifiche di progetto.
Per comunicare con i Committenti si utilizza l'indirizzo di posta elettronica, utilizzando un oggetto specifico e conciso per descrivere il contenuto del messaggio. Ci si rivolge ai committenti dando loro del Voi o del Lei.

4.1.2 Riunioni

Questa sotto-sezione definisce le regole che normano le riunioni, interne o esterne, e il loro svolgimento. Durante lo svolgimento di ogni riunione verrà nominato un segretario tra i membri del gruppo 353, il cui compito verterà nel far rispettare l'ordine del giorno, annotare gli argomenti discussi e redarre il Verbale di Riunione.

4.1.2.1 Verbale di riunione

Redigere il Verbale di Riunione è il compito del segretario, il quale sarà strutturato secondo questo scheletro:

- **Verbale TIPO del DATA:** costituirà il frontespizio;
  - **TIPO:** indica la tipologia d'incontro ossia se Interno o Esterno;
  - **DATA:** indica il giorno in cui si è svolta la riunione mentre per
- **Informazioni sulla riunione**: questa sezione contiene:
  - **Motivo della riunione**: sotto forma di paragrafo, che illustra i motivi generali della riunione;
  - **Luogo e Data**: ad esempio, Padova 05 Dicembre 2017;
  - **Ora di inizio**: nel formato ventiquattro ore, ad esempio 13:15;
  - **Ora di fine**: nel formato ventiquattro ore, ad esempio 15:30;
  - **Partecipanti**: elencati iniziando da Proponente/Committenti se la riunione è esterna, seguiti dai membri del gruppo partecipanti;
- **Ordine del Giorno**: indicato come elenco puntato degli argomenti da discutere;
- **Resoconto**: contiene il riassunto redatto dal segretario secondo i punti dell'ordine del giorno, precisando se sono stati discussi e le loro relative discussioni.

Nomenclatura e Conservazione: i file relativi ai verbali dovranno essere nominati come: **VER-DATA**.
Ad esempio VER-2017-12-05, per permettere una facile organizzazione dei documenti. Essendo documenti ufficiali, devono essere redatti in LaTeXe inclusi nella repository$_G$ omonima.

4.1.2.2 Riunioni interne

La partecipazione alle riunioni interne è permessa ai soli membri del gruppo 353. Il Responsabile di progetto ha il compito di stilare l'ordine del giorno, fissare la data e inserirla nel calendario e approvare il verbale redatto dal Segretario.
Di contro, i **partecipanti** devono presentarsi puntualmente alle riunioni, comunicare eventuali ritardi e partecipare attivamente alle discussioni.
Affinché una riunione sia ritenuta valida, devono essere presenti almeno cinque membri del gruppo 353.

4.1.2.3 Riunioni esterne

Le riunioni esterne vedono coinvolti sia i membri del gruppo 353 che alcuni soggetti esterni.
Principalmente le riunioni esterne vengono svolte in Torre Tullio Levi Civita, previa disponibilità dei locali.
A causa della locazione dell'azienda proponente, con sede a Amsterdam, le riunioni verranno principalmente effettuate tramite Skype$_G$, ovviando così al problema della distanza fisica.

4.2 Processi di pianificazione

4.2.1 Ruoli di progetto

La realizzazione del progetto è il risultato di un'attività collaborativa tra i membri del gruppo. Ad ogni persona infatti sarà attributo un ruolo, corrispondente a una figura aziendale, per un certo periodo di tempo. A rotazione, ogni membro del gruppo ricoprirà tutti i ruoli di seguito elencati:

- **Responsabile di progetto**;
- **Amministratore**;
- **Analista**;
- **Progettista**;

- **Programmatore**;
- **Verificatore**.

Il cambio del ruolo sarà effettuato in modo da garantire continuità alle attività in corso e far sì che ognuno ricopra ogni ruolo per un tempo omogeneo.
L'assegnazione dei ruoli verrà effettuata in modo da non creare condizioni contraddittorie, come ad esempio essere verificatori di documenti prodotti da se stessi, in modo da non compromettere la qualità del lavoro effettuato. Spetterà al verificatore il compito di individuare possibili accavallamenti di ruoli.

4.2.1.1 Responsabile di progetto

Il Responsabile di progetto, o "Project Manager", è il responsabile ultimo, per conto del suo gruppo, dei risultati del progetto. Partecipa al progetto per tutta la sua durata e accentra le responsabilità di scelta e di approvazione. Inoltre rappresenta il gruppo di lavoro nei confronti dei Committenti e dei Proponenti. Egli:

- Elabora ed emana piani e scadenze;
- Approva l'emissione di documenti;
- Coordina le attività del gruppo;
- Si relaziona con il controllo di qualità interno al progetto;
- Approva l'Offerta e i relativi allegati.

4.2.1.2 Amministratore

La figura dell'amministratore è indispensabile per permettere una buona produttività ed efficienza del gruppo, fornendo gli strumenti per adempiere ai propri compiti in maniera regolamentata. Deve gestire l'ambiente di lavoro redigendo i documenti che normano l'attività lavorativa e la loro verifica. Infatti:

- È responsabile della redazione e attuazione di piani e procedure di Gestione per la Qualità;
- Controlla il versionamento e le configurazioni dei prodotti;
- Collabora alla redazione del *Piano di progetto*;
- Redige le Norme di Progetto;
- Si assicura che la documentazione sia corretta, verificata, approvata e facilmente accessibile.

4.2.1.3 Analista

L'attività dell'Analista è necessaria e fondamentale affinché il progetto possa essere realizzato. Il suo compito è quello di analizzare il dominio del problema per comprenderlo a pieno, abbassando le probabilità che vengano effettuati gravi problemi di progettazione. I suoi compiti comprendono:

- Lo studio e la definizione del problema da risolvere, per capire cosa deve essere realizzato e definire quindi gli accordi contrattuali in base ai requisiti$_G$ richiesti;
- La verifica delle implicazioni di costo e qualità;
- La modellazione concettuale del sistema e la ripartizione dei requisiti$_G$;
- La realizzazione dello Studio di Fattibilità e dell'Analisi dei requisiti$_G$.

4.2.1.4 Progettista

Il Progettista è responsabile delle attività di progettazione, cioè della definizione di una soluzione soddisfacente per tutti gli stakeholder. Gli obiettivi del Progettista comprendono:

- La soddisfazione dei requisiti$_G$ con un sistema di qualità;
- La definizione dell'architettura logica del prodotto in modo che sia facile da mantenere, applicando soluzioni note e ottimizzate;
- La suddivisione del sistema in parti di complessità trattabile, per rendere il lavoro di codifica facilmente realizzabile e verificabile;

4.2.1.5 Programmatore

Il programmatore è responsabile delle attività di codifica che portano alla realizzazione del prodotto. Affinché questo avvenga, il suo compito consta solamente di implementare l'architettura definita dal Progettista. Per fare ciò:

- Scrive codice documentato, versionato e manutenibile secondo norme fissate;
- Crea le componenti necessarie alla verifica e alla validazione del codice;
- Si occupa della stesura del Manuale Utente.

4.2.1.6 Verificatore

Il Verificatore è una figura presente per tutta la durata del progetto. Il compito principale è quello di responsabile delle attività di verifica, che comprendono:

- L'accertamento che l'esecuzione delle attività di processo non abbia introdotto errori;
- La redazione del *Piano di qualifica* che illustra l'esito e la completezza delle verifiche e delle prove effettuate secondo il piano.

4.2.1.7 Rotazione dei ruoli

Ogni membro del gruppo dovrà ricoprire ciascuno dei ruoli del progetto.
La pianificazione dovrà essere eseguita con precisione rispettando le seguenti regole:

- Ogni membro del gruppo non dovrà mai ricoprire un ruolo che preveda la verifica dell'operato svolto da lui in precedenza poiché questo potrebbe portare ad un conflitto di interesse;
- Bisogna tener conto dei possibili impegni o interessi dei singoli membri del gruppo;
- Ogni ruolo, prevedendo comportamenti e attività differenti, dovrà essere trasferito tra i vari membri in modo ottimale lasciando quindi scritto su un documento informale una lista di consigli e/o procedure da parte dell'ultimo assegnatario di quel ruolo;
- Ciascun membro dovrà assicurare l'esclusivo svolgimento del ruolo a lui assegnato.

4.2.2 Ticketing

4.2.2.1 Task list

Il *Piano di progetto* prevede la suddivisione del modello di sviluppo in varie fasi. L'insieme delle attività da svolgere in una fase è contenuto in una task list.
Il Responsabile di progetto deve realizzare le task list per ogni fase del modello su Asana$_G$. Ogni task list viene quindi creata dal Responsabile del progetto avente il nome della fase alla quale si riferisce.

4.2.2.2 Task:

Ogni task viene creata dal Responsabile di progetto oppure da un membro del gruppo. Nel secondo caso è richiesta l'approvazione della task da parte della maggioranza (4) dei rimanenti componenti del gruppo. Ogni task è quindi composta da un titolo significativo e due date, la prima d'inizio mentre la seconda di termine previsto.

4.2.2.3 Ticket:

I ticket rappresentano l'unione di una task ad un membro del gruppo. L'assegnazione del ticket ad uno specifico componente del gruppo può avvenire secondo due modalità:

- **Pro-attivamente:** l'assegnatario del task è già noto al momento della creazione dello stesso. In questo caso il ticket viene assegnato direttamente allo specifico componente del gruppo da parte del Responsabile di progetto;
- **Retroattivamente:** nel caso di task a bassa priorità oppure di grandi quantità di lavoro da parte di tutti i componenti del gruppo, può non essere possibile assegnare direttamente un task ad un componente del gruppo. Questi tasks possono essere assegnati autonomamente qualora il componente finisse in anticipo i ticket a lui già assegnati.

4.3 Procedure

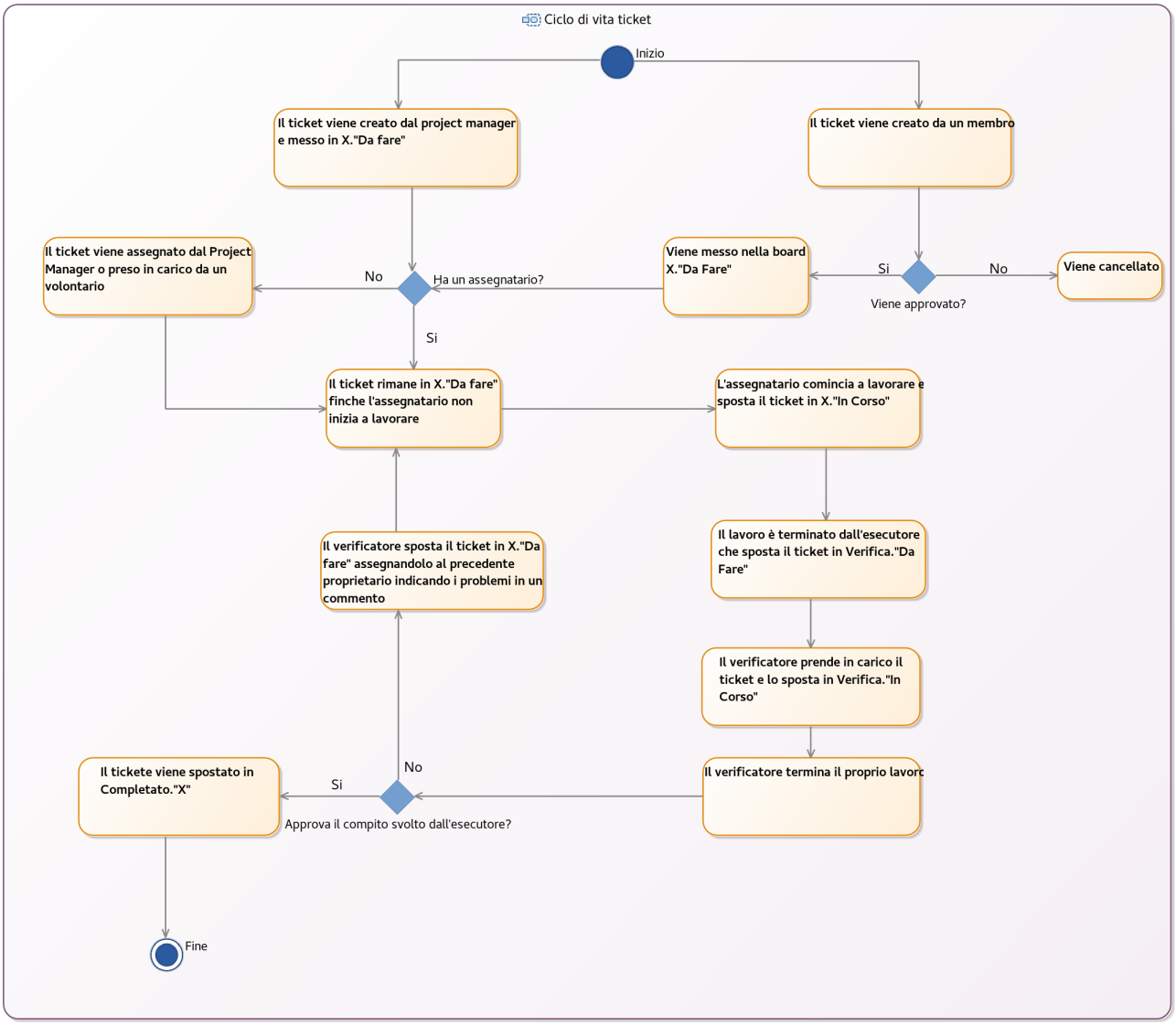
4.3.1 Creazione e gestione dei task

La procedura di creazione di un task è la seguente:

- accesso alla task list legata alla fase nella quale si vuole che l'attività sia svolta;
- creazione del nuovo task;
- se il task viene reputato troppo laborioso per essere eseguito individualmente il Responsabile di progetto può suddividerlo in sotto-task assegnate a loro volta singolarmente ai membri del gruppo.

4.3.2 Gestione dei ticket

Ogni ticket si rifà al seguente ciclo di vita a partire dalla sua creazione fino al completamento.



4.3.3 Stesura del consuntivo

L'operazione di stesura del consuntivo può essere effettuata solamente dal Responsabile di progetto della fase corrente. Le operazioni che il Responsabile di progetto esegue sono le seguenti:

1. Esporta da Asana$_G$ tramite Instagantt$_G$ un foglio di calcolo, in formato Microsoft Excel, che mostra le ore rendicontate nella fase corrente;
2. Inserisce nel foglio di calcolo precompilato relativo alla fase corrente, realizzato sulla base del template disponibile su GitHub$_G$;
3. Inserisce le ore rendicontate nelle celle previste, ottenendo quindi la differenza di ore tra il preventivo e le ore rendicontate;
4. Aggiunge all'interno del *Piano di progetto* una sezione mostrante i valori ottenuti da questo confronto;
5. Crea un a tabella nella sezione descritta punto precedente mostrante la differenza tra le ore preventivate e quelle rendicontate e il budget preventivato rispetto a quello ottenuto dalle ore rendicontate;
6. Trae infine delle conclusioni dai risultati avuti e scrive una valutazione complessiva del lavoro effettuato nella fase corrente.

Per la revisione RR sono effettuati solamente i passaggi dall' 1 al 3 senza effettuare le differenze non avendo ore rendicontate per le fasi successive. Successivamente ad ogni revisione sarà aggiornato il *Piano di progetto* con i valori ottenuti dal confronto tra ore preventivate e rendicontate.

4.4 Strumenti

4.4.1 Pianificazione

Per quanto riguarda la pianificazione, la scelta del gruppo 353 è ricaduta sul servizio cloud Asana$_G$.
Questo servizio offre la possibilità di creare dei task ed assegnarli ai membri del gruppo 353, indicando anche la data di inizio e di fine in modo da consentire una pianificazione del lavoro da svolgere.
Inoltre è possibile indicare delle dipendenze tra attività, favorendo così un controllo dei vincoli per lo svolgimento dei compiti.

Standups: per incentivare la continuità nel tempo dello sviluppo della commessa, si è scelto di utilizzare Standup Alice, un Bot$_G$ per *Slack* per la realizzazione giornaliera di standups$_G$, permettendo al gruppo di seguire costantemente l'operato degli altri componenti e di confrontarsi nel caso sorgano dubbi o domande.

4.4.2 Creazione diagrammi di Gantt

Lo strumento scelto per la realizzazione dei diagrammi di Gantt_{G} è GanttProject, in quanto è gratuito, open source, multi piattaforma ed è stato considerato adatto ai nostri bisogni.

4.4.3 Calcolo del consuntivo

Gli strumenti utilizzati dal Responsabile di progetto sono i seguenti:

- **Instagantt**[1];
- **LibreOffice Calc**[2].

4.5 Formazione

4.5.1 Formazione dei membri del gruppo

La formazione del personale è da realizzarsi in maniera autonoma. I membri del gruppo 353 sono tenuti a studiare individualmente le tecnologie che verranno utilizzate nel corso del progetto. É possibile che i membri del gruppo realizzino, in piena libertà, delle guide a carattere informale e relative ad un singolo argomento, allo scopo di facilitare la formazione ai restanti componenti del gruppo.

4.5.1.1 Ore rendicontabili e di investimento

Per distinguere se le ore svolte sono a carico del proponente (rendicontabili) e non di formazione personale (di investimento), si devono verificare le seguenti caratteristiche:

1. **Incremento:** il contenuto documento o software è stato modificato venendo ampliato;
2. **Standard:** non si sono studiati nuovi concetti, procedure o strumenti.

4.5.1.2 Guide e materiale utilizzato

La documentazione di riferimento, oltre che al materiale già citato nella sottosezione *Riferimenti Informativi*, comprende:

[1] https://instagantt.com
[2] https://it.libreoffice.org/scopri/calc/

- Per l'utilizzo di LaTeX: https://www.latex-project.org;
- Per l'utilizzo di GitHub: https://github.com;
- Per l'utilizzo di React: https://reactjs.org;
- Per l'utilizzo di Redux: https://redux.js.org;
- Per l'utilizzo di Ethereum: https://www.ethereum.org;
- Per l'utilizzo di Metamask, https://metamask.io;
- Per l'utilizzo di Solidity: https://solidity.readthedocs.io/en/develop.

Il versionamento dei prodotti servirà anche per apprendere dall'operato altrui, in modo da integrare le conoscenze personali migliorando la qualità e l'efficienza delle attività.