Guida_allo_sviluppo_di_progetti

December 3, 2024

1 Condizioni richieste perché il progetto possa essere concluso e discusso con i docenti con esito positivo

- Il progetto deve essere sviluppato da un gruppo costituito di norma da tre studenti.
- **Almeno 15 giorni prima della data prevista per la discussione del progetto**, un membro del gruppo deve comunicare **a tutti i docenti** il repository GitHub che verrà usato per lo sviluppo del progetto stesso.
- Il membro del gruppo proprietario del repository **deve attivare le notifiche sui push al docente incaricato di monitorare il progetto**. Per farlo deve:
  - entrare nella sezione `Settings` (elenco orizzontale in alto a destra)
  - entrare nella sottosezione `Email notifications` (elenco verticale a sinistra in fondo)
  - inserire la mail del docente nella casella di testo `Address`
  - verificare che le notifiche siano attive (check box `Active` in basso)
- Il progetto deve essere sviluppato **gradualmente** e completato **entro il giorno precedente alla data prevista per la discussione** in modo che il docente incaricato di monitorarlo possa verificare che tutte le condizioni richieste siano state verificate. **L'esito positivo di tale verifica è condizione necessaria per la discussione**. Per sviluppo *graduale* si intende che vengano effettuati un numero adeguato di commit da parte di tutti i membri del gruppo distribuiti in un periodo di almeno 15 giorni precedenti alla discussione.
- Il progetto deve soddisfare tutte le specifiche fornite e deve includere i dati necessari per effettuare test finalizzati alla verifica di correttezza del codice. In particolare, è richiesto che esso includa il codice corrispondente alla verifica automatica di almeno 3 test di correttezza utilizzando il modulo *unittest*.
- Il progetto deve utilizzare le classi come strumento di modularizzazione del codice e deve usare design pattern nei casi in cui il loro uso consenta una generalizzazione dei diversi componenti software inclusi nel progetto. In particolare, si suggerisce di valutare l'impiego del pattern *strategy* e del pattern *factory* predisponendo in tal modo il progetto alla possibilità di prevedere soluzioni alternative all'implementazione di alcune funzioni *anche se di fatto verrà inclusa nel progetto finale una sola soluzione impelementativa*.
- Nello sviluppo è consentito l'uso dei package e moduli standard, nonché l'uso di *numpy*, *pandas*, e *matplotlib*. Nel caso il gruppo ritenga opportuno usare altri package o moduli deve essere autorizzato dal docente che effettua il monitoraggio del progetto.

- Il progetto deve essere completato con le istruzioni e i file necessari per create un immagine docker dell'intera applicazione. L'immagine deve essere configuirata in modo che sia possibile passare ai container generati a partire da essa il file contenente il dataset di partenza e di salvare sul file sistem della macchina ospite i file con i risultati.
- Nella discussione i componenti del gruppo dovranno:
  - discrivere la loro esperienza nello sviluppo del progetto;
  - discutere possibili alternative alle soluizioni implementative adottate con riferimento alla parte di progetto svolta personalmente da ciascun membro, inclusi i casi di testi impiegati;
  - rispondere a una domanda sulla parte di programma relativa all predisposizione del dataset e alla valutazione dei risultati di una pipeline di classificazione.

2 Guida allo sviluppo del progetto

Per svolgere un progetto Python in modo che sia facile configurarlo e manutenerlo e che possa essere monitorato efficacemente dal docente che segue il progetto procedere nel modo seguente.

2.1 Specifica del progetto

Creare il file README.md e mantenerlo aggiornato con la specifica del progetto. Man mano che si rilasciano versioni funzionanti del progetto descrivere nel file README.md cosa deve fare l'utente per eseguirlo. Se questa documentazione è troppo estesa per essere contenuta nel file README.md aggiungere uno o più file aggiuntivi, elencarli in README.md con una breve illustrazione del loro contenuto.

In generale, tutti questi file devono essere file di testo. Usare un linguaggio di *markup* per includere elementi di formattazione nel testo (per es. il linguaggio **markdown**).

2.2 Creazione del repository

Se si sta svilupando una nuova applicazione creare una directory all'interno della quale inserire tutti i componenti che verranno via via generati. Configurare la directory come repository locale usando il comando:

```
git init
```

N.B. Se si vuole che il progetto abbia un repository su GitHub effettuare i seguenti passi:

- creare un repository vuoto (senza i file README.md, di licenza o .gitignore)
- copiare la REMOTE_URL dalla sezione Quick setup
- da terminale, dalla directory del repository locale dare i comandi:

```
git remote add origin <REMOTE_URL>
git remote -v
git push -u origin main
```

Se in alternativa si parte da un progetto già presente su GitHub, effettuarne il clone con il comando:

```
git clone <REMOTE_URL>
```

dove REMOTE_URL è l'URL copiata da GitHub.

In ogni caso si lavora all'interno di una directory indicata nel seguito `root_dir`.

WARNING: Creare .gitignore (o aggiornarlo se lo si è già creato) per escludere i file e le directory che non si vogliono tracciare. In particolare se si una un IDE, nella `root_dir` possono esserci file di configurazione da escludere (per esempio l'ambiente pycharm crea la directory nascota `.idea`). In generale, aggiungere informazioni a .gitignore quando si creano file o directory che si vogliono escludere.

Per esempio, se i file che contengono i dati di test sono molto grandi può essere conveniente escludere le directory che contengono tali dati (o alcuni file in esse contenuti).

Puramente a titolo di esempio, un contenuto minimo del file .gitignore potrebbe essere il seguente:

```
env/*
__pycache__/*
data/*
results/*
.idea/*
.DS_Store
```

che esclude, oltre ai file del virtual environment anche i file temporanei di codice intermedio creati dall'interprete, i dati per il testing (sottodirectory *data*), gli eventuali risultati (sottodirectory *results*), i metadati di configurazione del progetto PyCharm, il file nascosto *.DS_Store* creato da Mac OS.

2.3 Sviluppo collaborativo da parte di tutti i componenti di un gruppo

- Ciascun membro del gruppo crea un clone locale su cui lavora.
- Concordare tra i membri del gruppo come gestire il lavoro in collaborazione: tutti i membri del gruppo devono risultare attivi sul repository GitHub; si può lavorare su un solo branch o su più branch, **purché sul branch principale sia presente sempre una versione funzionante dell'applicazione**.
- Avvertire il docente che il progetto è attivo su GitHub e verificare che la possibilità di inviare *issues* sia abilitata,
- Abilitare il docente a ricevere notifiche sui push effettuati.
- Gestire tutte le interazioni con il docente tramite le *issues* non inviare mail su questioni riguardanti il progetto, a meno di situazioni particolari che non possono essere gestite tramite le *issues*.
- Il docente effettua un frequente monitoraggio del progetto sul branch principale e comunica le proprie osservazioni tramite le *issues*.
- Rispondere puntualmente a tutte le *issues* inviate dal docente; se si ritiene necessario interagire direttamente con il docente comunicarlo rispondendo alla *issue* corrispondente.
- Se si desidera che il docente faccia delle verifiche su branch secondari informarlo rispondendo a una *issue*.

- Conviene rilasciare versioni intermedie anche con funzionalità molto parziali, purché funzionanti sui casi di test forniti.

2.4 Casi di test

Preparare un set di casi di test sufficienti per verificare le funzionalità dell'applicazione. Se non ovvio, specificare per ciascun caso di test quale è l'output atteso.

Se richiesti, i file relativi ai casi di test vanno caricati manualmente sul repository remoto pubblico in modo che chi effettua un clone del progetto possa scaricare anche i dati richiesti per effettuare i test.

2.5 Commenti

È importante inserire commenti man mano che si sviluppa codice. I commenti devono essere redatti tenendo presenti le seguenti linee guida:

- Deve essere inserito un commento subito prima o, meglio, subito dopo la prima riga di ogni funzione, inclusi i metodi di una classe. Tali commenti devono includere i seguenti elementi: (i) una breve descrizione della funzione svolta; (ii) la descrizione del ruolo svolto da ciascun parametro della funzione/metodo; (iii) la descrizione dei risultati restituiti (o un messaggio che la funzione/metodo non restituisce risultati).
- Per ogni classe deve essere inserito un commento che descrive il ruolo della classe e la sua rappresentazione interna (cioè quali attributi utilizza e quale è il loro ruolo). Questo commento può essere distinto da quelli relativi ai metodi, o essere incluso in uno di essi (per esempio in quello del costruttore).
- Nel corpo di funzioni e metodi, così come negli script, i commenti devono essere brevi e possono avere due funzioni:
  - (i) descrivere il ruolo di una istruzione o di un gruppo di istruzioni (evitando di ripetere ciò che risulta già evidente dal codice); questi commenti sono particolarmente necessari quando il codice fa molto uso di meccanismi linguistici molto potenti che compattano in poche righe di codice operazioni di una certa complessità, o quando i dati non vengono rappresentati in modo naturale, ma ricorrendo a forme di codifica (per es. usando numeri interi, indici, ecc.).
  - (ii) esplicitare una proprietà che è sempre verificata prima o dopo un'istruzione o un gruppo di istruzioni.

Si raccomanda di studiare il codice distribuito con il materiale del corso per avere esempi di come applicare in pratica i prinicipi sopra elencati.

2.6 Effettuare commit frequenti e mantenere aggiornata la documentazione

Effettuare frequentemente i *commit* e se il progetto è anche su repository remoto effettuare i corrispondenti push.

Utilizzare branch secondari per lo sviluppo.

Prima di rilasciare una versione sul branch principale, verificare sempre che l'applicazione soddisfi i requisiti speficati nel file README.md effettuando i test opportuni.

2.7 Installazione di moduli non standard

Man mano che si sviluppa l'applicazione installare i moduli non standard richiesti dall'applicazione (denominati solitamente *requirements*).

Se si sta usando un virtual environment, mantenere sempre aggiornato il file `requirements.txt`} che contiene la lista dei moduli installati con il comando:

```
pip freeze > requirements.txt
```

Se, al contrario, non si sta usando un virtual environment, creare il `requirements.txt` in questo modo include nella lista tutti i moduli installati per l'interprete in uso, cose che corrisponde normalmente a includere molti moduli superflui rispetto al progetto in esame.

2.8 Creazione di un virtual environment

Prima di iniziare a scrivere codice, creare nella `root_dir` un virtual environment minimale (da terminale o dall'ambiente di sviluppo).

Se si crea il virtual environment da terminale nella directory `root_dir` si può usare il comando:

```
python3.X -m venv PATH_RELATIVO_VIRTUAL_ENVIRONMENT
```

dove ovviamente la `X` va sostituita con la versione di Python che si vuole usare.

Se si lavora da terminale, attivare il virtual environment con il comando:

```
source ./PATH_RELATIVO_VIRTUAL_ENVIRONMENT/bin/activate
```

nel caso di sistema operativo Linux o MacOS, oppure:

```
./PATH_RELATIVO_VIRTUAL_ENVIRONMENT/Scripts/activate
```

nel caso di sistema operativo Windows.

Per deattivare l'environment usare il comando:

```
deactivate
```

WARNING: Aggiornare .gitignore (o crearlo se non lo si è già creato) per escludere la directory del virtual environment se è stata creata come sottodirectory di `root_dir`.

Aggiornare pip nel caso la versione di Python usata per generare il virtual environment non contenga una versione di pip aggiornata. Per esempio si può usare il comando:

```
pip install --upgrade pip
```

2.9 Sviluppo con IDE

Se si lavora utilizzando un *Integrated Development Environment* (IDE) verificare se l'IDE permette di configurare come interprete di lavoro quello di un virtual environment.

Per esempio se si usa PyCharm:

- creare un nuovo progetto nella directory del progetto git;
- configurare l'interprete del virtual environment appena creato come interprete di lavoro.

Se si usa un altro IDE, consultare la relativa documentazione per effettuare le configurazioni analoghe.

2.10 Clone dell'applicazione in un nuovo contesto

Per installare l'applicazione in un nuovo contesto, dopo aver effettuato il colone del progetto e aver creato un virtual environment minimale, usare il comando:

```
pip install -r requirements.txt
```

oppure creare l'immagine docker dell'applicazione partendo da un'immagine configurata con la versione di python desiderata.

2.11 Progetti di esempio

Sulla pagine pubbliche GitHub dell'utente `iannellog` sono disponibili alcuni progetti che esemplificano quanto illustrato nei paragrafi precedenti. Link ad alcuni di questi progetti sono pubblicati anche sul sito del corso.