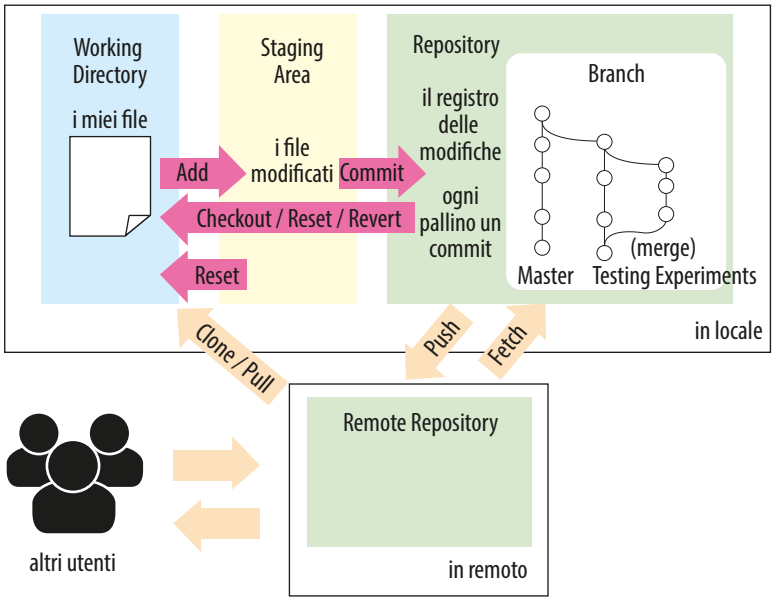
1 ESERCIZI IN LABORATORIO

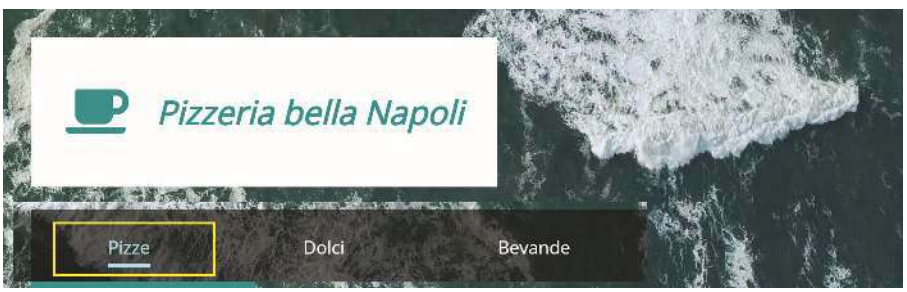
Un esempio di sviluppo collaborativo con GitHub

Prima di presentare uno schema operativo per effettuare lavori di gruppo riassumiamo con una grafica i principali comandi di **Git** che utilizzeremo nella nostra esercitazione; li possiamo separare in due gruppi di operazioni:

- **in locale**: con Git;
- **in remoto**: con GitHub, anche mediante il software **GitHub Desktop**.

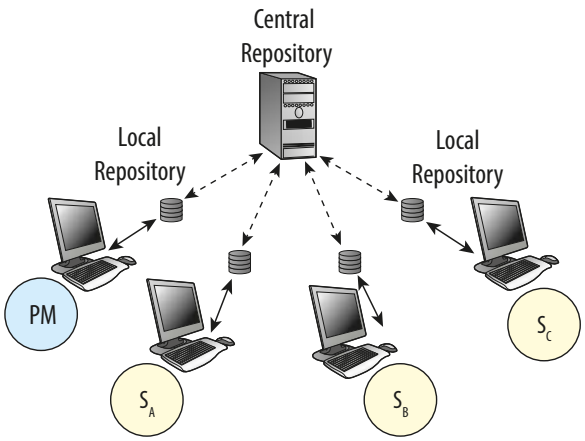


Il progetto che affronteremo è quello di un'applicazione web per una pizzeria. In particolare, realizzeremo un menù composto da tre sezioni, come riportato nella seguente figura:



Il nostro gruppo di lavoro è composto da quattro persone: il **project manager** e tre **sviluppatori**. A ogni sviluppatore viene assegnato il compito di realizzare una sezione del menù, e precisamente:

- allo **sviluppatore A** viene assegnato il *menù Pizze*;
- allo **sviluppatore B** viene assegnato il *menù Dolci*;
- allo **sviluppatore C** viene assegnato il *menù Bevande*.



ESERCIZI IN LABORATORIO 1

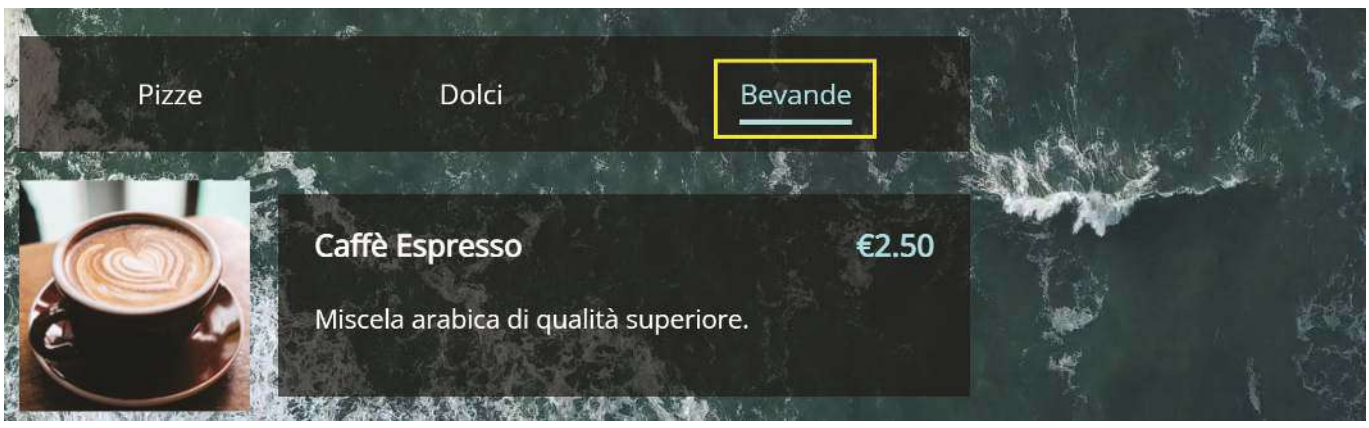
Il template che abbiamo scelto per le pagine web è molto semplice da comprendere e utilizzare: le tre sezioni sono da "inserire" nei tre `<div>` che abbiamo di seguito evidenziato all'interno del file `index.html`; quest'ultimo sarà il file che verrà modificato in contemporanea dagli sviluppatori (e sarà causa di conflitto nell'aggiornamento).

```
53          </nav>
54          <!--  elenco pizze -->
55          <div id="Pizze" class="tm-tab-content">
56            <div class="tm-list">
57
58            </div>
59          </div>
60          <!--  elenco dolci -->
61          <div id="Dolci" class="tm-tab-content">
62            <div class="tm-list">
63
64            </div>
65          </div>
66          <!--  elenco bevande -->
67          <div id="Bevande" class="tm-tab-content">
68            <div class="tm-list">
69
70              <div class="tm-list-item">
71                <img src="img/espresso.png" alt="Image" class="tm-list-item-img">
72                <div class="tm-black-bg tm-list-item-text">
73                  <h3 class="tm-list-item-name">Caff&egrave; Espresso <span class="tm-list-item-price">€2.50</span></h3>
74                  <p class="tm-list-item-description">Miscela arabica di qualit&agrave; superiore.  </p>
75                </div>
76              </div>
77
78            </div>
79          </div>
80          <!-- end Menu Page -->
81        </div>
```

Come si può vedere dall'immagine precedente, nel template il project manager ha lasciato un esempio di "riga di menù" per gli sviluppatori, utile per vedere come aggiungere un **item**; in particolare è stato aggiunto nell'elenco delle Bevande il *Caffè Espresso*:

```
66          <!--  elenco bevande -->
67          <div id="Bevande" class="tm-tab-content">
68            <div class="tm-list">
69
70              <div class="tm-list-item">
71                <img src="img/espresso.png" alt="Image" class="tm-list-item-img">
72                <div class="tm-black-bg tm-list-item-text">
73                  <h3 class="tm-list-item-name">Caff&egrave; Espresso <span class="tm-list-item-price">€2.50</span></h3>
74                  <p class="tm-list-item-description">Miscela arabica di qualit&agrave; superiore.  </p>
75                </div>
76              </div>
77
78            </div>
79          </div>
```

Visualizziamo il **menù Bevande** in modo da comprendere il significato grafico di ogni istruzione.



1 ESERCIZI IN LABORATORIO

Configurazione ambiente e avvio progetto

I primi compiti del **project manager**, di seguito indicato con **PM**, per avviare il lavoro sono:

1. creare un repository su Github;
2. caricare il template del progetto;
3. formare il gruppo di lavoro invitando i singoli sviluppatori/programmatori.

Nel dettaglio, verranno effettuate le operazioni da PM e dai singoli sviluppatori S_i mostrate a destra:

PM Inizializzare il repository web su GitHub [`New`]

PM creare il gruppo di lavoro su GitHub [`xxx`]

PM creare il **repository locale** e collegarlo al **repository remoto** [`init`]

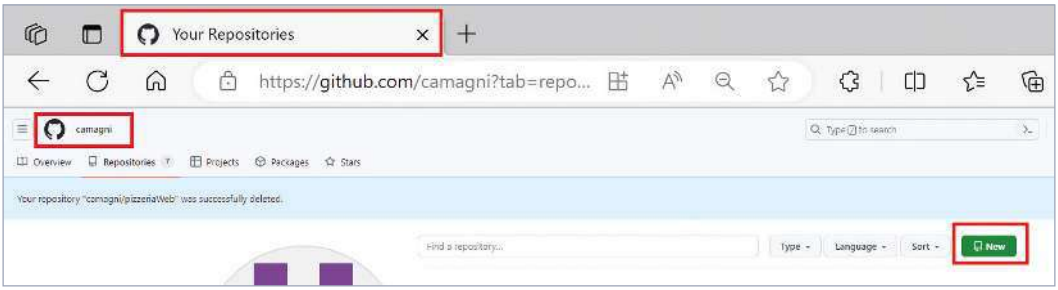
PM salvare il **template del progetto** nel repository locale [`status`][`add`][`commit`]

PM copiare il **repository locale** nel repository remoto [`push`]

S_i clonazione del repository da remoto a locale [`clone`]

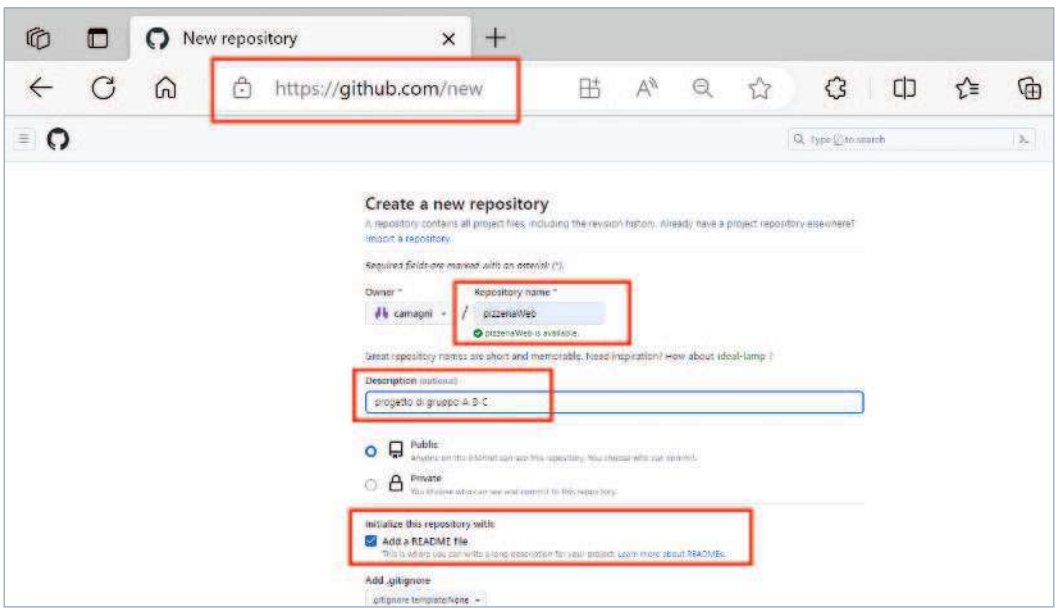
Inizializzazione del repository web

Il PM, come prima operazione, si posiziona sul proprio ambiente GitHub e crea un nuovo repository remoto:



Dopo aver cliccato su **New** nella scheda dei **Repository**, inserisce i dati del progetto, indicando il nome del repository, che sarà anche l'indirizzo remoto al quale dovranno connettersi gli sviluppatori: per comodità nella trattazione didattica, noi lasciamo come tipologia **Public**; nel caso di progetti "veri" consigliamo invece di settare **Private** e permettere l'accesso solo ai membri autorizzati e/o invitati.

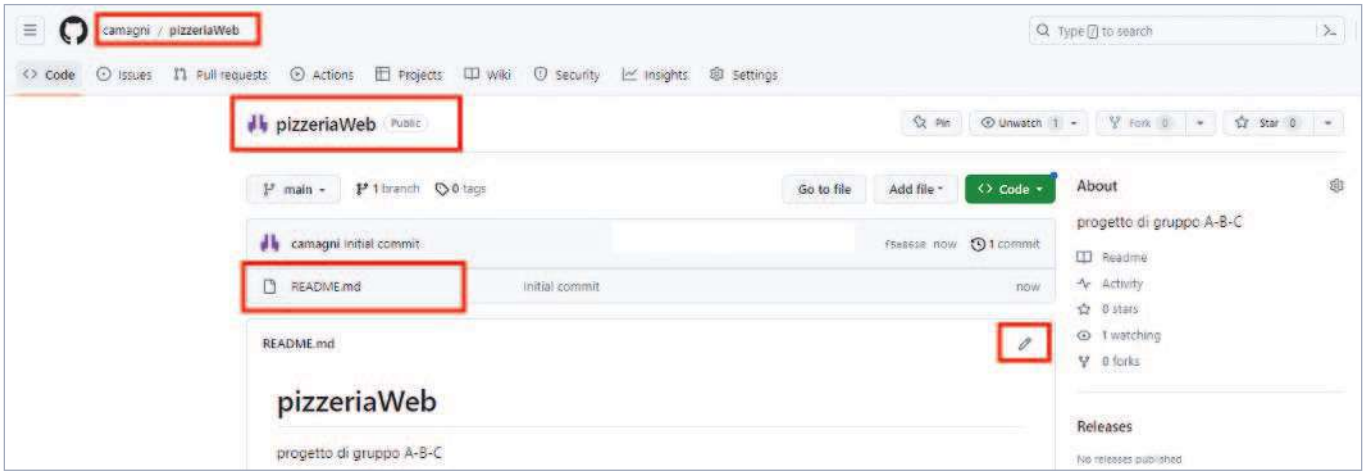
Nei progetti pubblici non è necessario invitare i partecipanti, ma noi faremo lo stesso, a titolo di esempio.



ESERCIZI IN LABORATORIO 1

Avendo indicato di inserire il file `README` nel progetto, alla conferma della creazione:

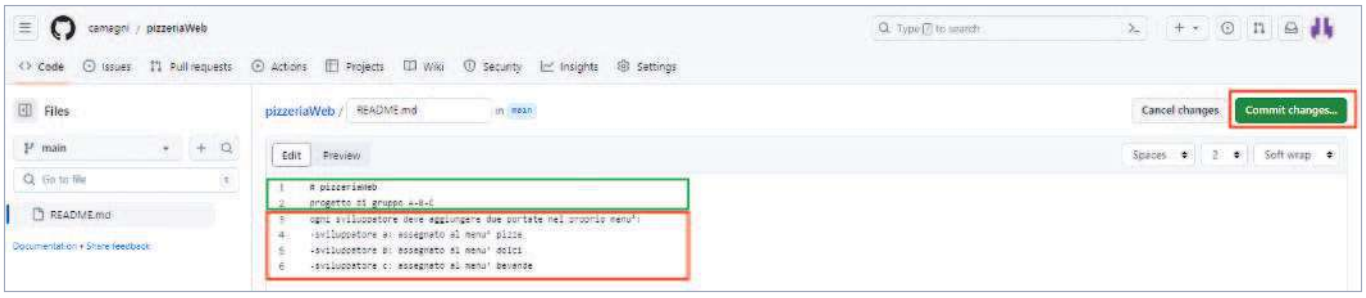
ci si trova in questa situazione:



Modifichiamo il file `README.md`. Le prime due righe sono create in automatico: completiamo il file con le altre indicazioni che il PM desidera condividere con gli sviluppatori:

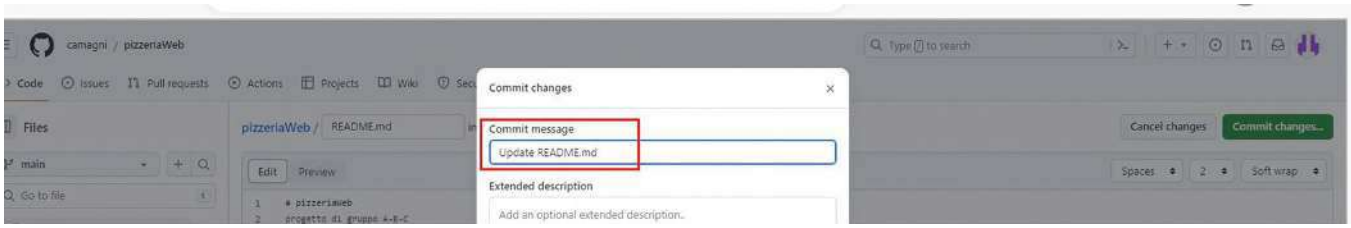
```
ogni sviluppatore deve aggiungere due portate nel proprio menu':
-sviluppatore a: assegnato al menu' pizze
-sviluppatore b: assegnato al menu' dolci
-sviluppatore c: assegnato al menu' bevande
```

Per poter "abbellire i file di testo" inserendo comandi di formattazione come intestazioni, grassetto, italico ecc., GitHub di default propone un file di tipo `.md`, cioè di "Markdown Documentation".

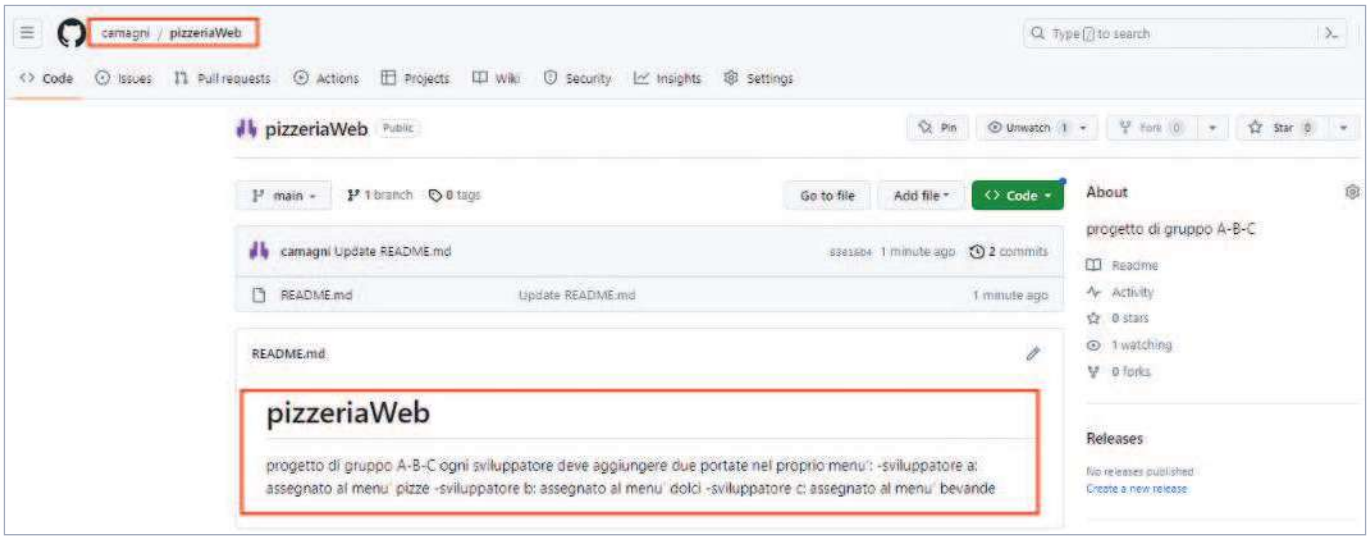


1 ESERCIZI IN LABORATORIO

La pagina di **Edit** ci permette di scrivere il file direttamente in GitHub: al termine della digitazione effettuiamo il **commit** in quanto anche la semplice modifica del file `README.md` deve seguire la prassi di tutti i file modificati:

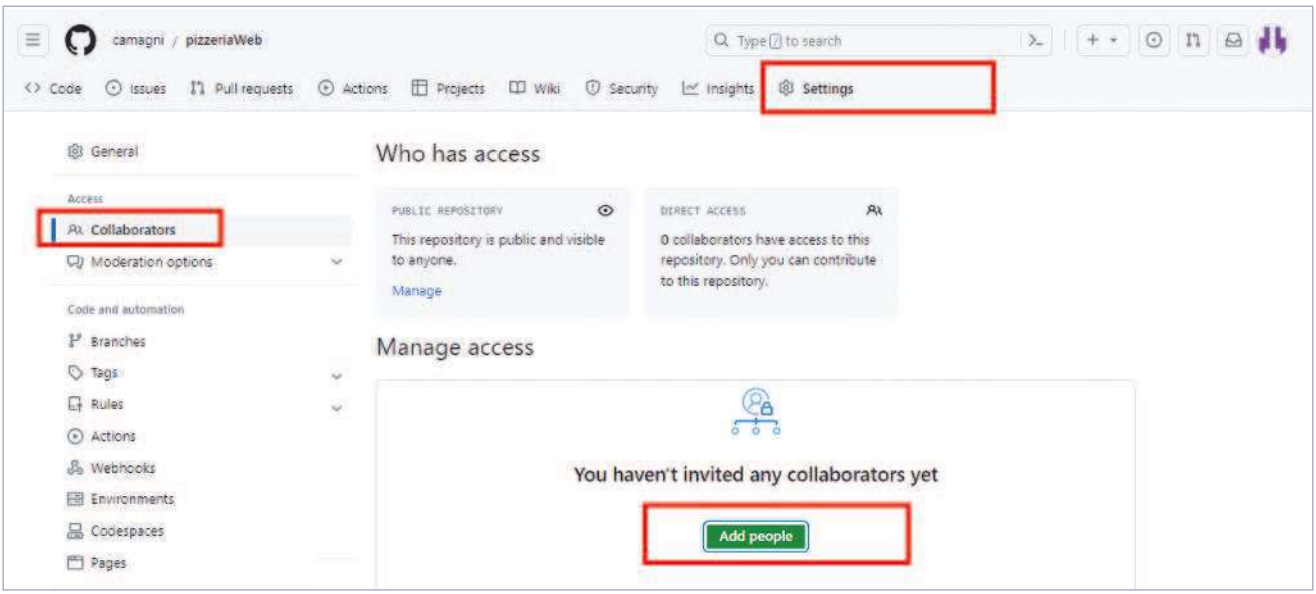


La nuova situazione "di partenza" è quella presentata in figura.



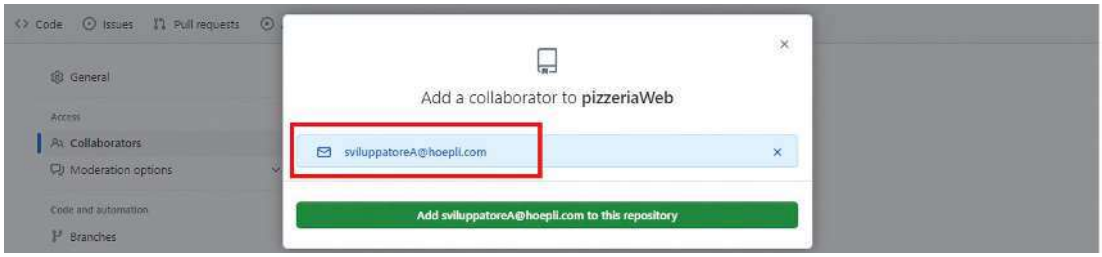
Formazione del gruppo di lavoro

A questo punto il PM invita gli sviluppatori a partecipare al progetto: dal menu **Settings** seleziona la prima opzione della colonna di sinistra e procede aggiungendo i membri del team:

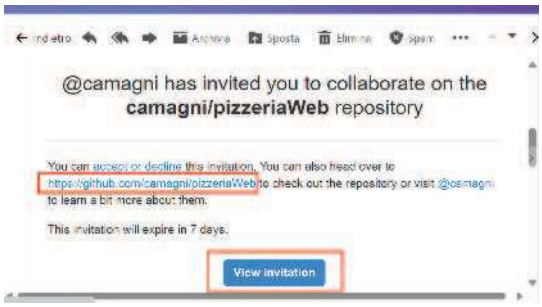


ESERCIZI IN LABORATORIO 1

Selezioniamo tra gli indirizzi e-mail proposti quello degli sviluppatori prescelti oppure inseriamo un nuovo indirizzo di posta:

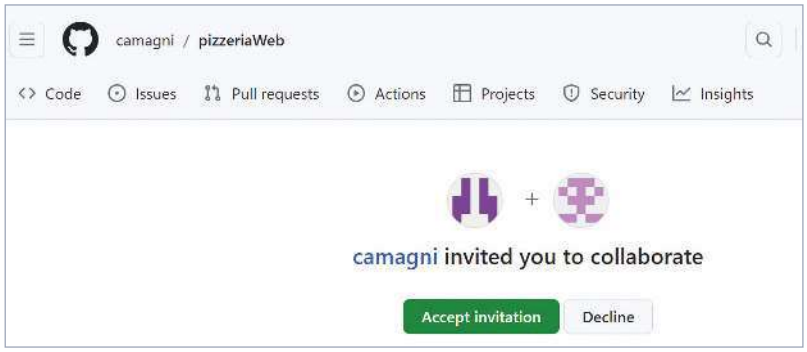


Con questa operazione spediamo una mail di invito agli sviluppatori simile alla seguente:

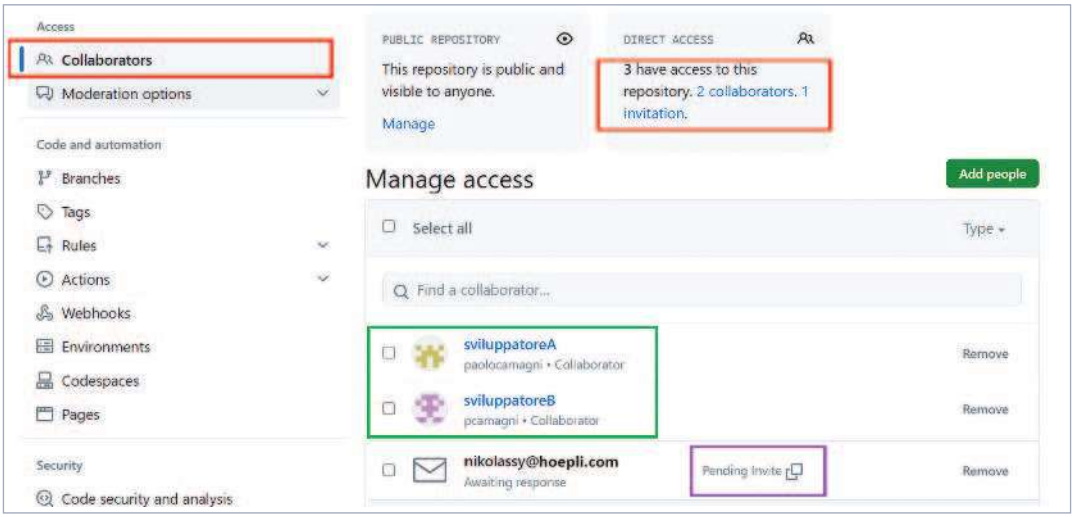


Osserviamo che viene indicato l'indirizzo del repository remoto dove il programmatore dovrà collegarsi.

Dopo aver visionato l'invito, lo sviluppatore conferma (o meno) la sua partecipazione:

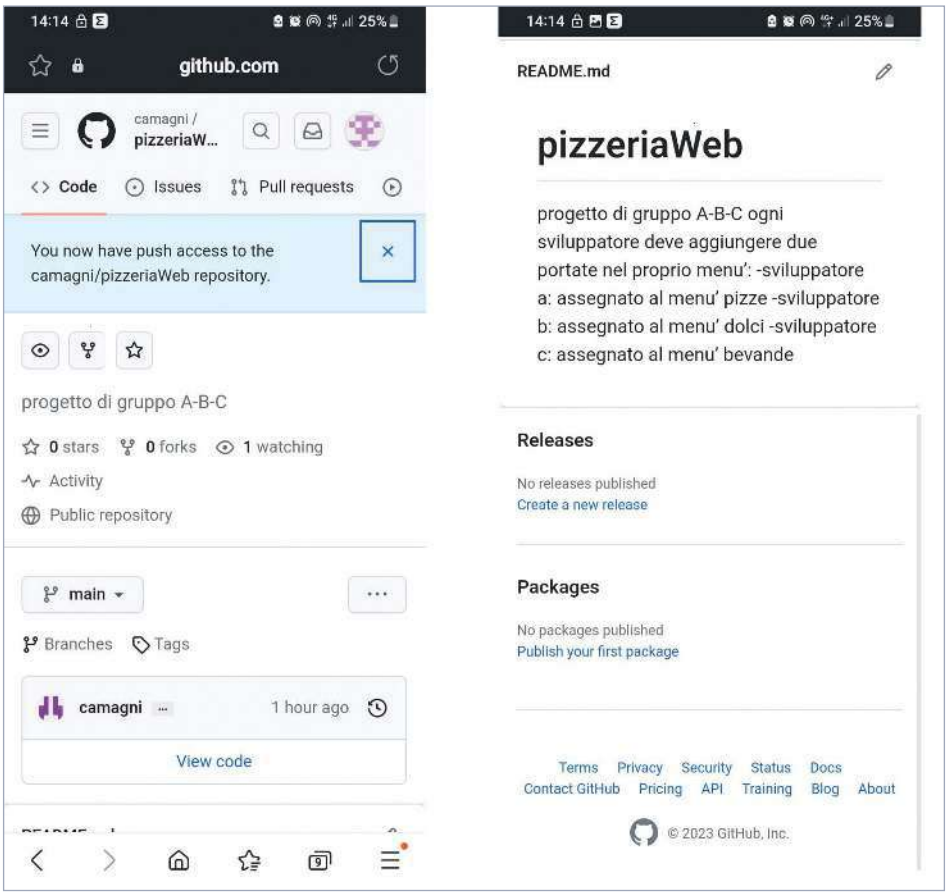


Successivamente all'inoltro degli inviti, in qualità di PM aspettiamo le risposte che, una alla volta, iniziano ad arrivare: in questo esempio notiamo che due collaboratori hanno risposto positivamente:



1 ESERCIZI IN LABORATORIO

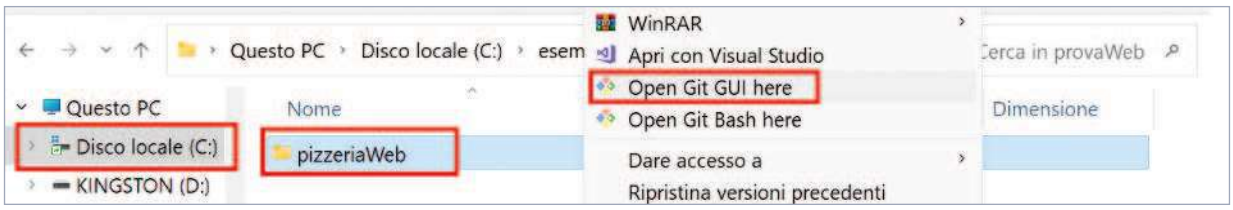
Possiamo anche visualizzare dallo smartphone l'avanzamento del nostro lavoro:



Inizializzazione del repository locale del Project Manager

Il Project Manager crea un repository locale sul proprio PC dove saranno memorizzati i file del template del progetto che dovranno essere condivisi con gli sviluppatori.
Come primo passo crea un **repository locale**:

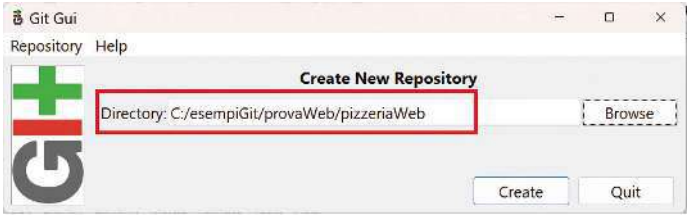
1. crea una directory, nel nostro caso `pizzeriaWeb`;
2. con il tasto destro del mouse clicca sulla cartella e apre **Git GUI**:



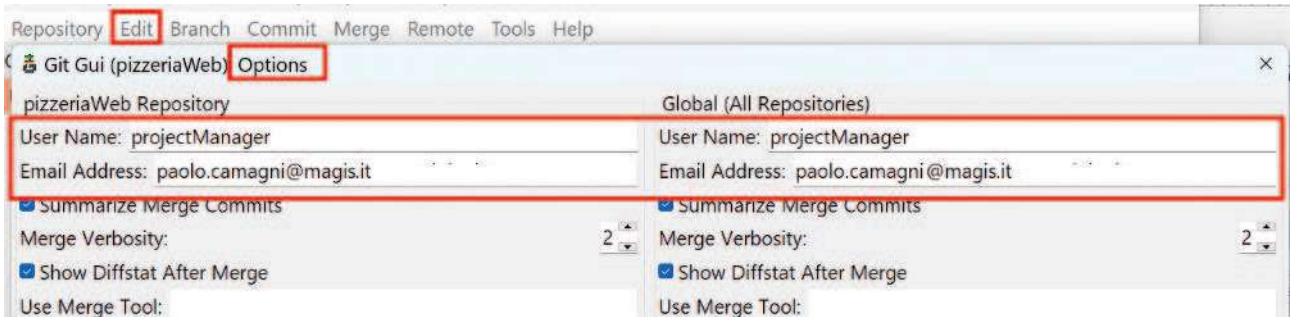
3. nella videata che si presenta procede con la creazione del nuovo repository: questa operazione effettua l'`init` della cartella, creando la directory `.git`.

ESERCIZI IN LABORATORIO 1

4. Come directory prescelta seleziona la cartella appena creata, che conterrà il template:



Si procede quindi a configurare l'ambiente di lavoro, indicando il nome del programmatore, che servirà in seguito per identificare il suo lavoro nel gruppo, e il suo indirizzo di posta, dove riceverà le comunicazioni sull'evoluzione del progetto.



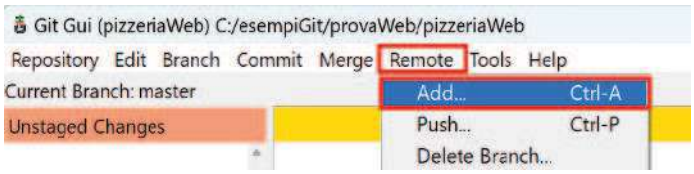
Download della prima versione del repository remoto

Come test di connessione al **repository remoto** effettuiamo il download del file `README.md`.
Sono molteplici le operazioni da effettuare, di seguito indicate cronologicamente:

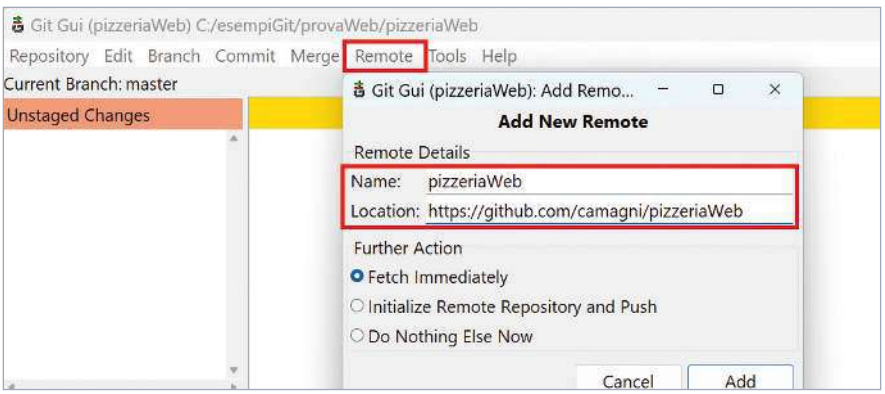
1. inizializzare nel repository locale il collegamento con il repository remoto;
2. dato che noi utilizziamo **GitHub Desktop**, creare in esso la connessione con il **Git locale**;
3. verificare che la "catena di connessioni" sia corretta;
4. effettuare il primo allineamento scaricando il file `README.md` in locale.

Procediamo passo per passo.

1. Ci posizioniamo sull'opzione **Add** del menu **Remote** per connettere il repository locale al repository remoto:

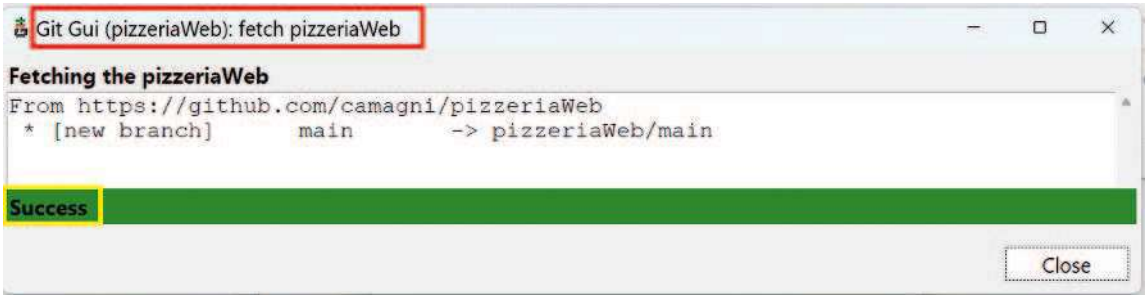


Scriviamo il nome del progetto e riportiamo il suo indirizzo completo:



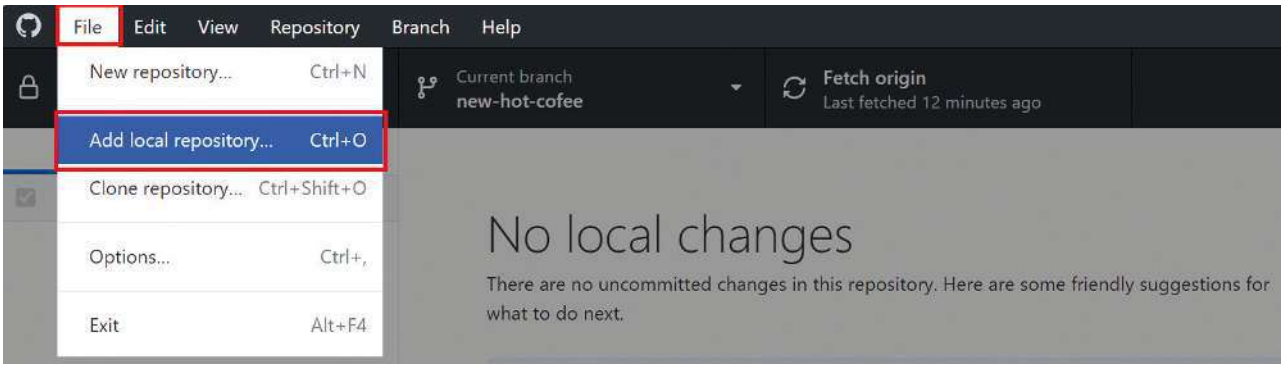
1 ESERCIZI IN LABORATORIO

Se tutto è andato a buon fine ci viene presentata la seguente schermata:

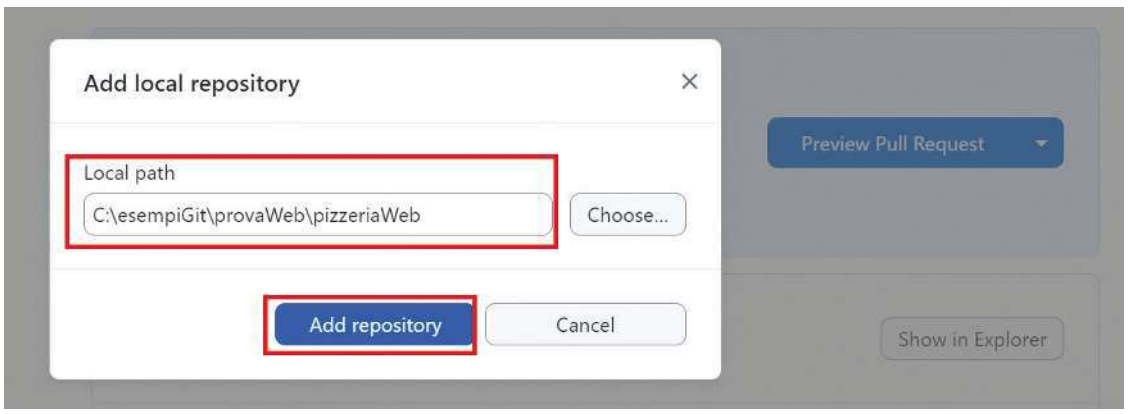


dove la barra verde ci segnala il successo dell'operazione.

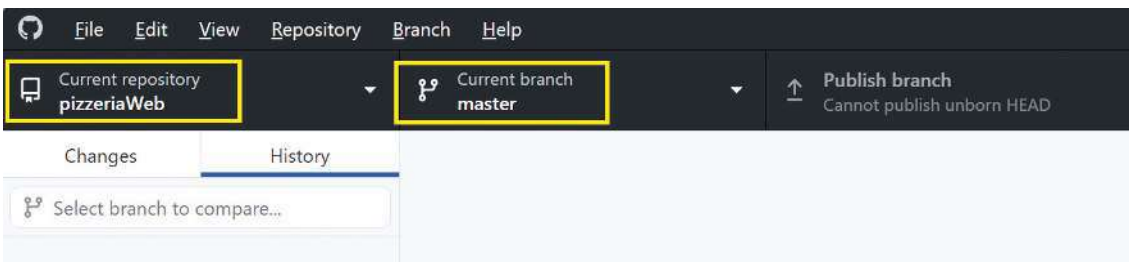
2. Creiamo ora la connessione tra **GitHub Desktop** e il nostro repository presenti in locale: da GitHub Desktop selezioniamo nel menu **File** l'opzione apposita:



e come **local path** selezioniamo la directory che abbiamo indicato come **repository locale**.

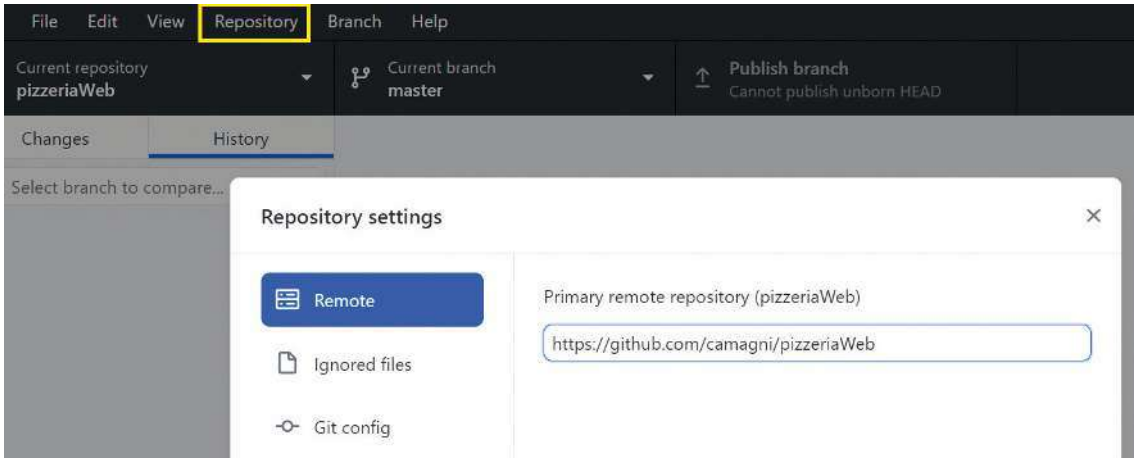


Osserviamo ora la presenza del nostro direttorio indicato come repository corrente:

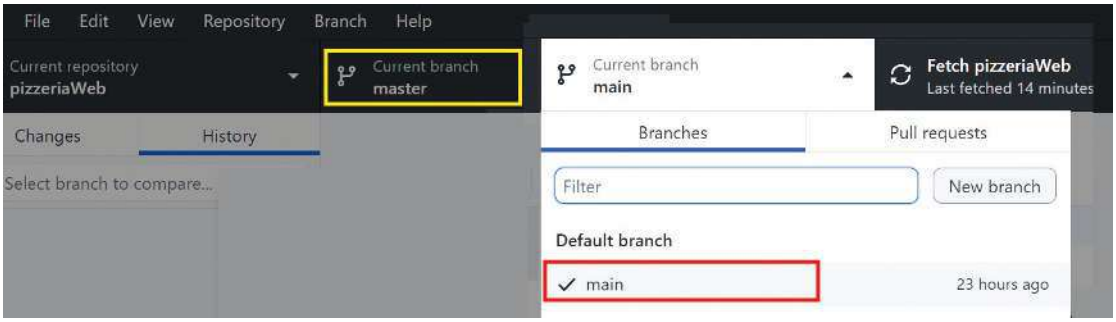


ESERCIZI IN LABORATORIO 1

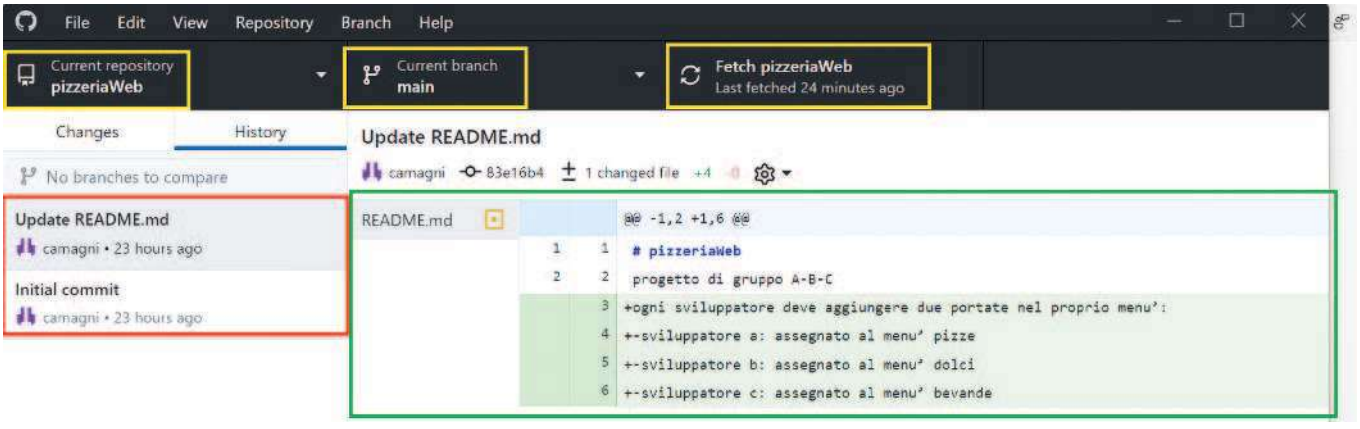
3. Verifichiamo che la "catena di connessioni" sia corretta andando a scaricare in locale il file `README.md` dal repository web.
Innanzitutto verifichiamo se l'indirizzo del **repository remoto** è presente ed è corretto:



Clicchiamo su **Current branch** e selezioniamo **main**:

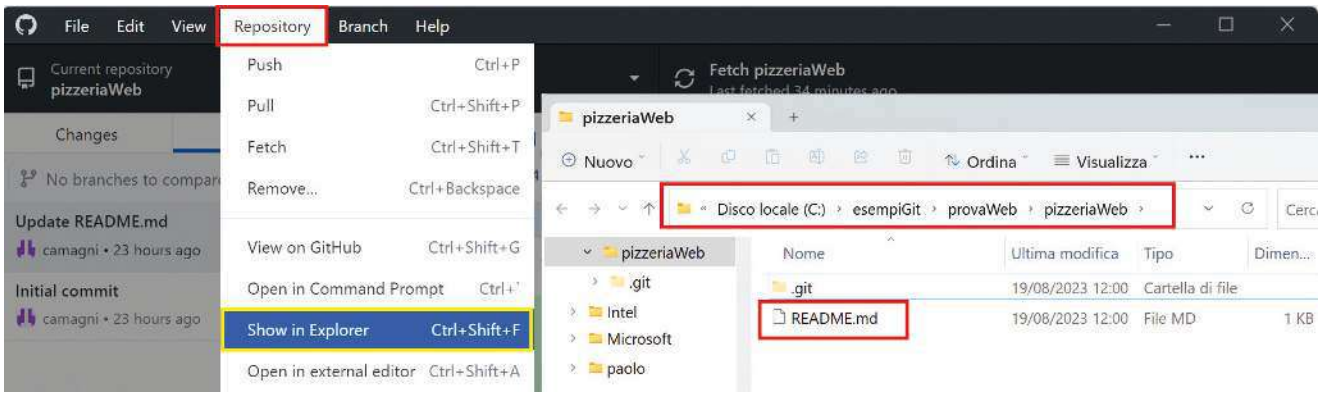


Nella prima tendina possiamo vedere l'elenco delle operazioni fatte sul repository web; di default ci viene visualizzata l'ultima operazione, che nel nostro caso è l'editing del file `README.md`; nella barra del menu ci viene anche indicato quando è stato fatto l'ultimo "allineamento" (**Fetch**).



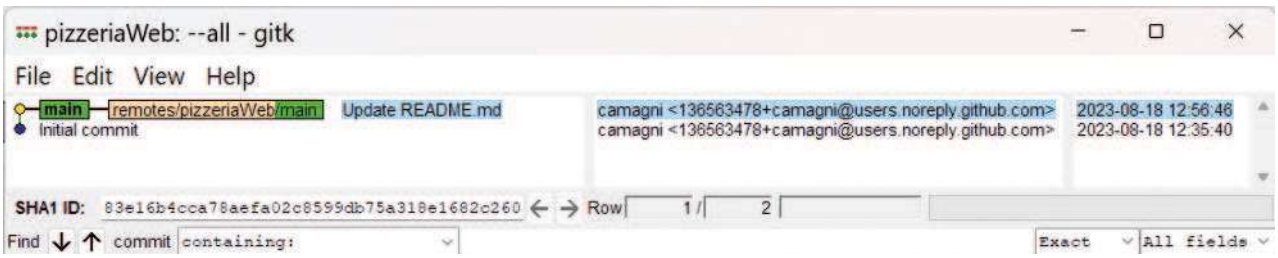
1 ESERCIZI IN LABORATORIO

4. Selezioniamo ora dalla tendina del menu **Repository** l'opzione indicata:



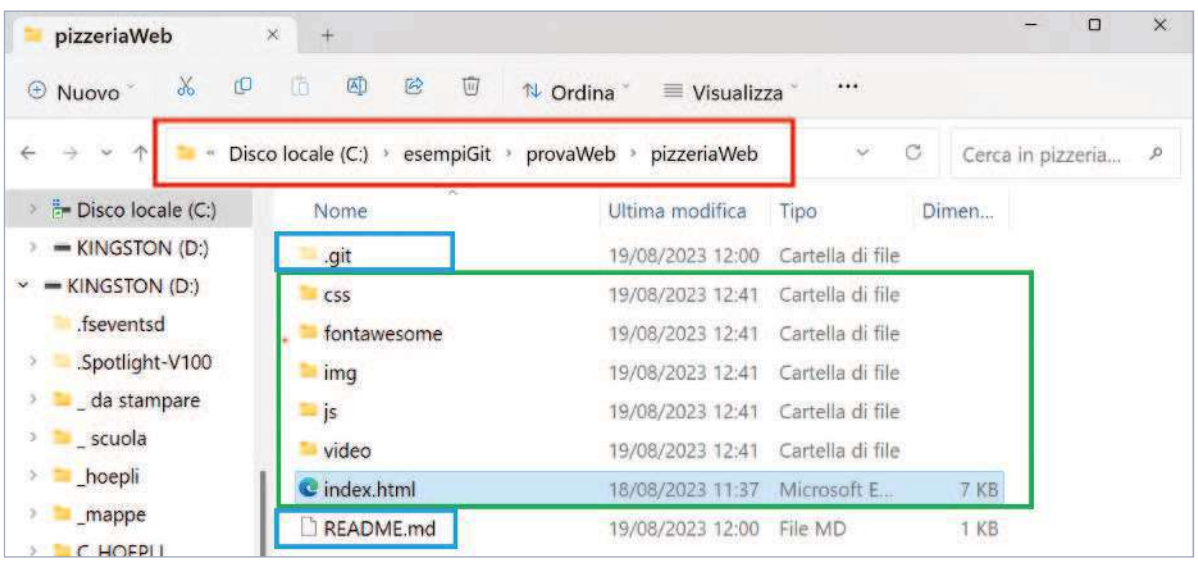
Possiamo così constatare che "l'allineamento" con il repository remoto è stato fatto automaticamente in quanto nel nostro repository locale troviamo il file `README.md`.

Se passiamo ora alla visualizzazione grafica in locale dei **branch** otteniamo qualcosa di simile a quanto mostrato nella seguente videata.



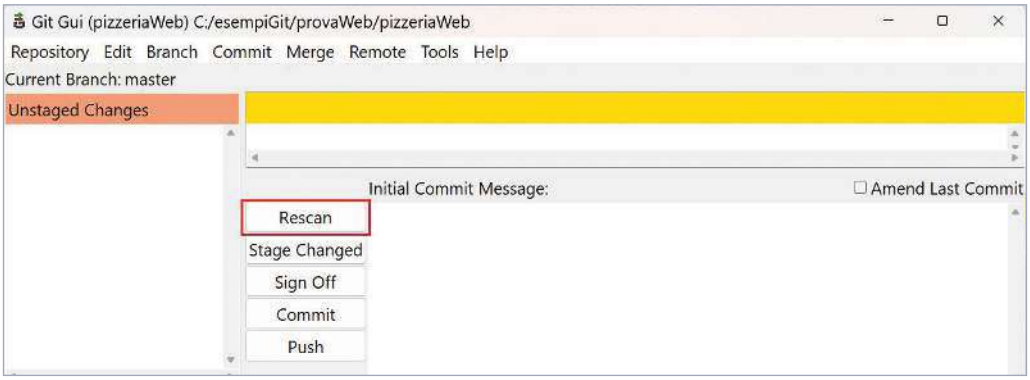
Salvataggio della prima versione del repository locale

Ora noi, come Project Manager, copiamo nella directory locale tutti i file relativi al template del sito web che dobbiamo realizzare ricordando che, per semplicità didattica e senza perdere di genericità, le modifiche saranno portate solo al file `index.html` che contiene il codice da "completare in condivisione" per mano degli sviluppatori (**Si**):

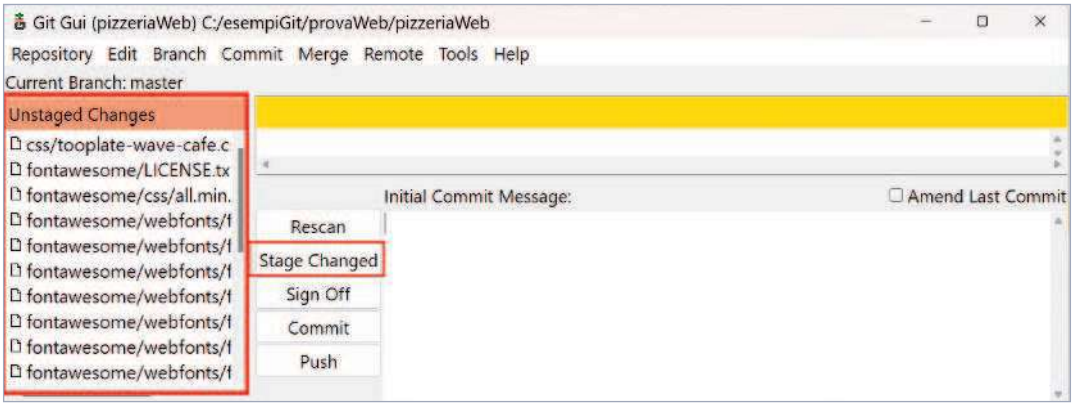


ESERCIZI IN LABORATORIO 1

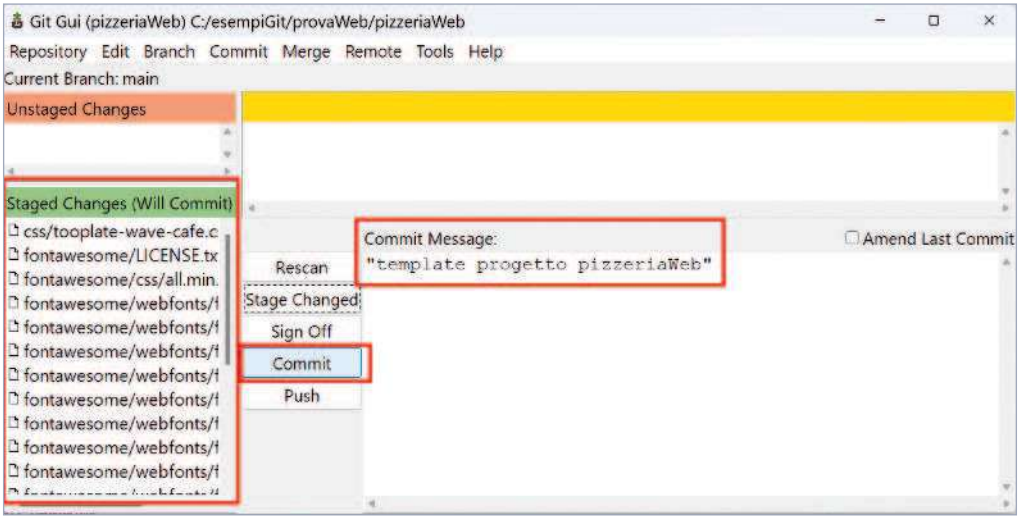
Nella nostra directory locale, oltre alla cartella `.git` e al file `README.md`, abbiamo le **cartelle del progetto** e il file `index.html`.
Rientriamo in **Git GUI** e come prima operazione effettuiamo un refresh cliccando su **Rescan**:



I nuovi file aggiunti al **repository locale** vengono individuati ed evidenziati nella sezione in alto a sinistra della schermata, sotto la voce **Unstaged Changes**.
Procediamo aggiungendoli allo stage cliccando su **Stage Changed**:

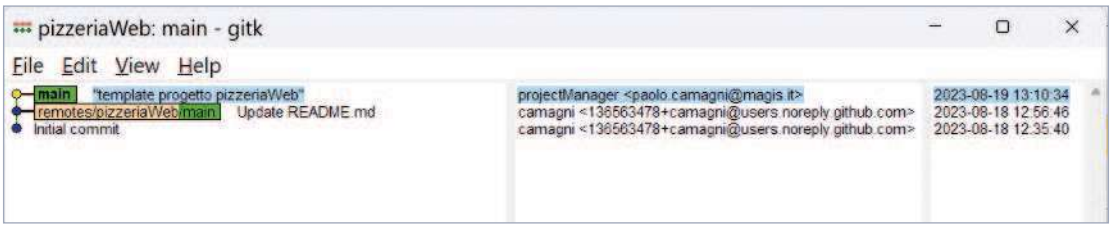


Osserviamo che i file sono stati "trasferiti" nella parte inferiore, pronti per essere "committati": effettuiamo quindi il primo **commit** di tutti i file del template "dandogli un nome" nell'apposita area per poterli poi individuare in tutto il progetto.

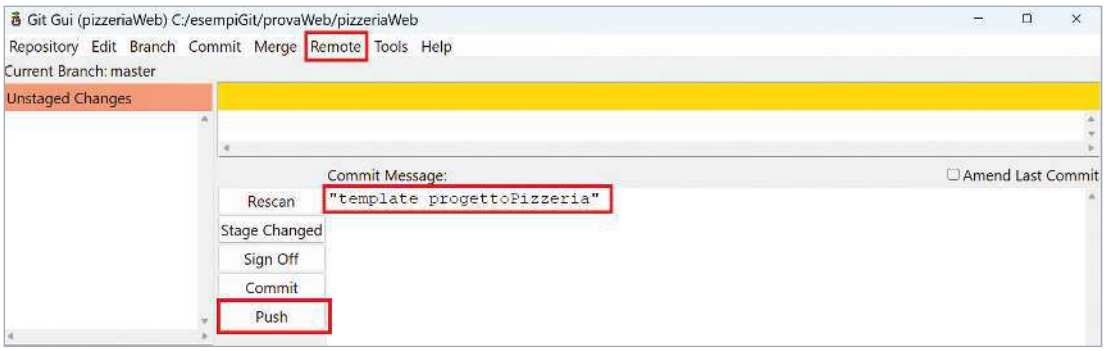


1 ESERCIZI IN LABORATORIO

Siamo nella seguente situazione, visualizzabile con il comando **Repository | Visualize...**:



Come ultimo passaggio effettuiamo il **Push** per trasferire tutti i file nel Web lasciando come etichetta di riferimento `"template progettoPizzeria"`:

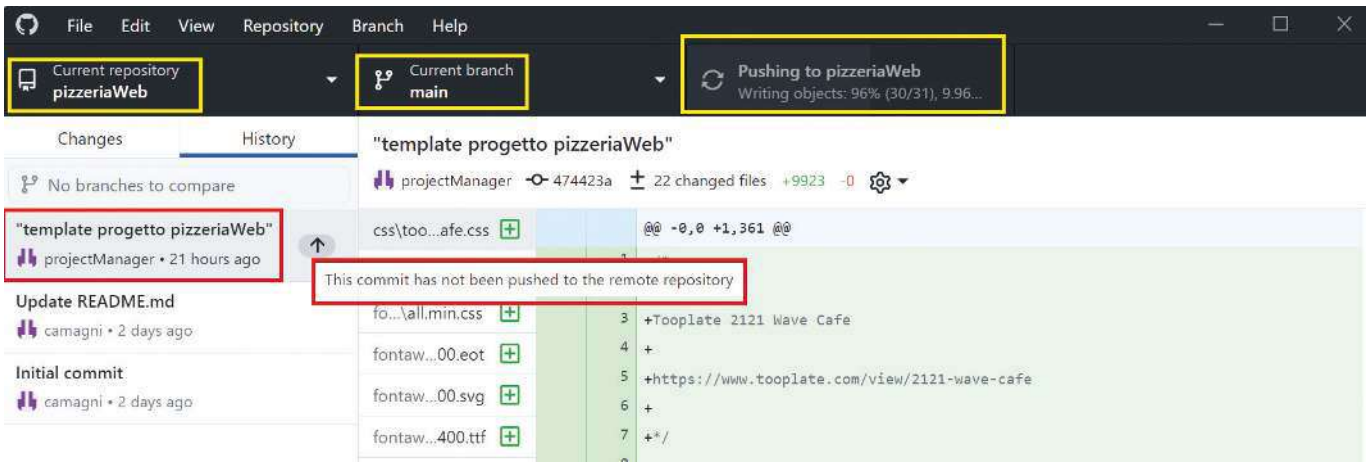


Se tutto va a buon fine ci viene restituita una schermata simile a quella mostrata a fianco, con barra verde che indica l'esito positivo delle operazioni.

Ora spostiamoci sul Web per verificare il **repository remoto**: possiamo effettuare il controllo in due modalità differenti.

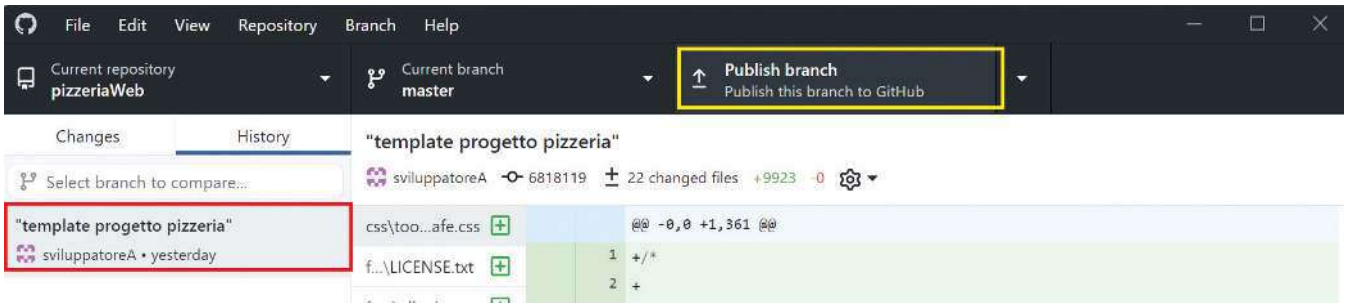
1) Modalità Desktop

Spostandoci su **GitHub Desktop** troviamo le nuove aggiunte che, però, non sono ancora pubblicate sul repository web: a tal fine clicchiamo sull'ultimo pulsante che effettua il **push**:

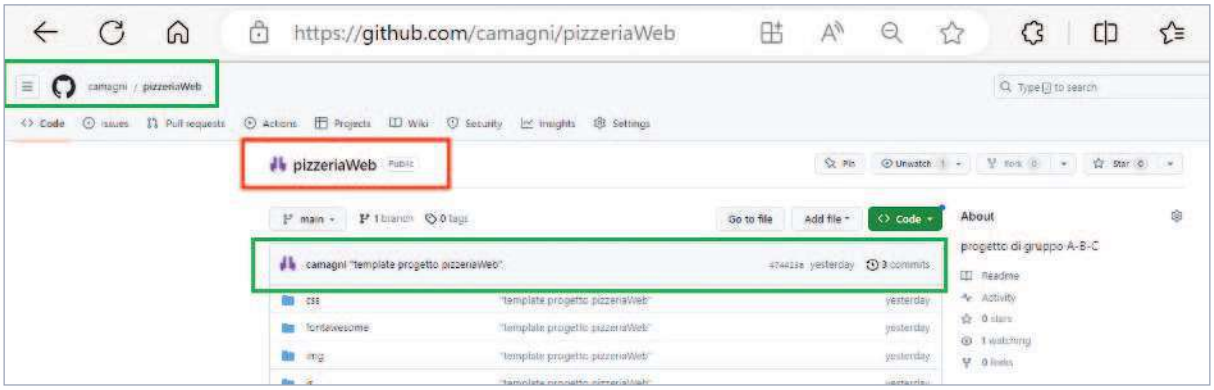


ESERCIZI IN LABORATORIO 1

e ci portiamo in questa situazione:



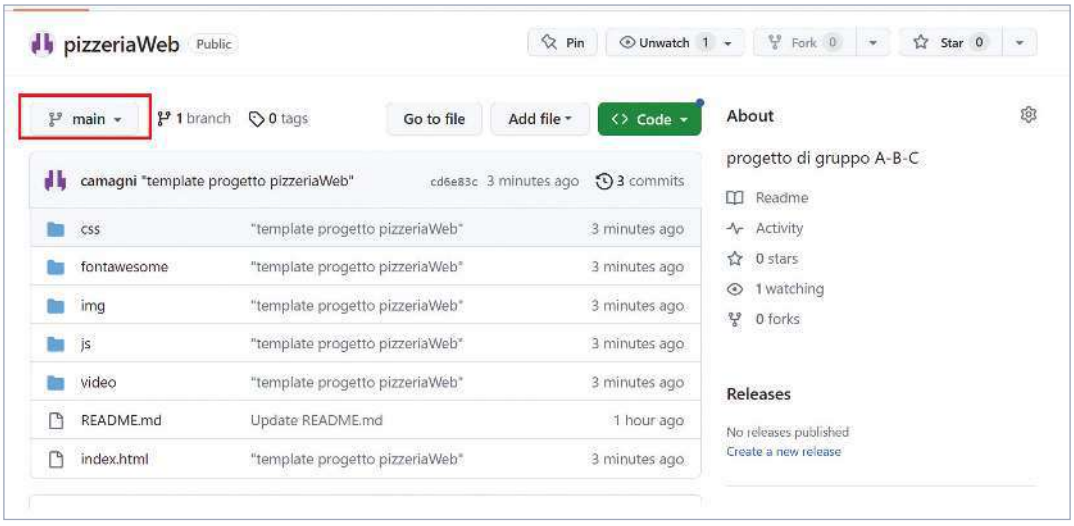
Ora nel browser possiamo vedere la nuova situazione presente in rete:



In alcune versioni questo primo "allineamento" avviene automaticamente.

2) Modalità browser

Se invece andiamo direttamente sul browser, l'aggiornamento avviene in automatico in quanto a ogni accesso il browser provvede a verificare se ci sono operazioni in sospeso.
Naturalmente ci troviamo nella medesima situazione vista con la modalità Desktop.



Questa linea di sviluppo, che è la principale (**main** o **master branch**), è quella che contiene la "versione pubblicabile" (**deployable**), cioè quella che viene data al cliente/utente.

1 ESERCIZI IN LABORATORIO

A partire da essa, il gruppo di sviluppo può ora iniziare a lavorare per realizzare il progetto.

> Il **primo principio dello sviluppo condiviso**, o prima regola del GitHub Flow, riporta che:
> #1 - anything in the master branch is deployable

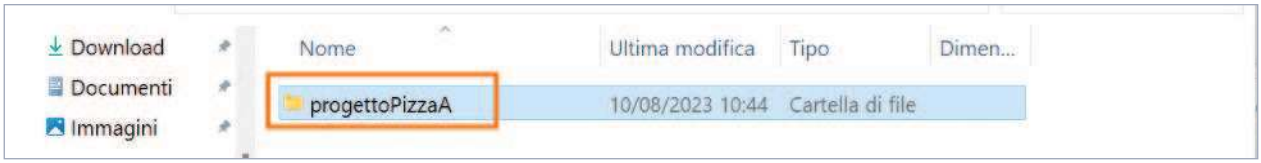
Il PM da ora in avanti è responsabile di tutto ciò che finisce su questa versione, perché andrà in produzione, cioè sarà messa online e sarà usufruibile dagli utilizzatori/consumatori/clienti.

Impostazione Git locale degli sviluppatori

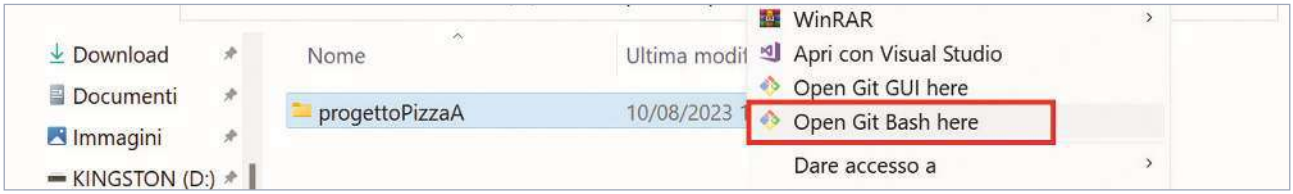
Abbiamo detto che con l'invito il **project manager** condivide con il gruppo di sviluppo l'indirizzo del repository remoto, nel nostro caso:

```
https://github.com/camagni/pizzeriaWeb
```

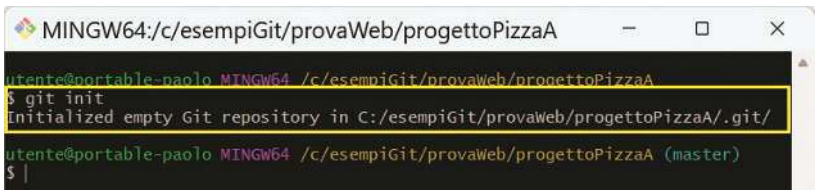
Gli sviluppatori possono ora iniziare a realizzare i propri menù.
Come prima operazione ogni sviluppatore crea una directory di lavoro sul proprio PC, che per noi che ora "ci immedesimiamo" nello **sviluppatoreA,** è la seguente:



> A scopo didattico, per lo **sviluppatoreA** eseguiremo le stesse operazioni prima effettuate per il PM con il supporto dell'interfaccia GUI, digitandole ora dalla linea di comando, cioè in modalità **Bash**, così da avere un esempio completo delle due possibili modalità operative offerte da Git.



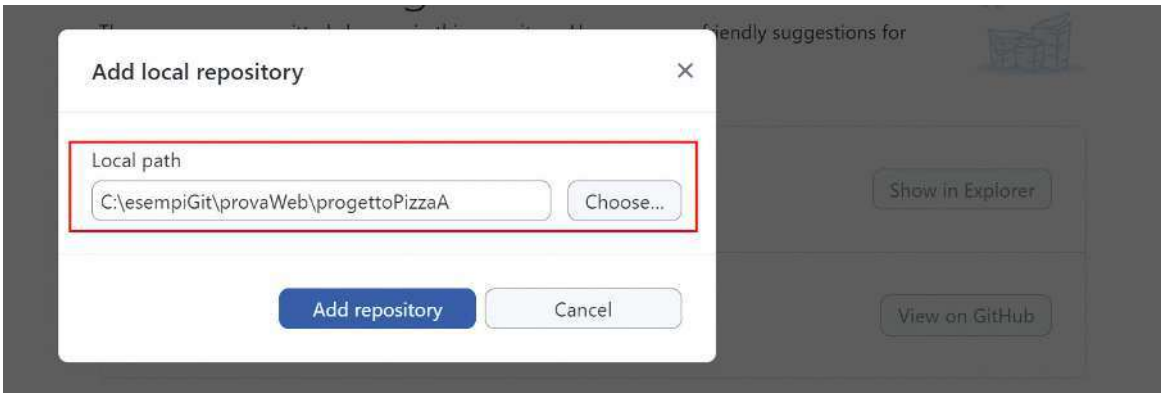
Quindi, a partire dalla directory `progettoPizzaA`, creiamo il nostro **repository locale**: dopo aver verificato che la cartella sia vuota, la inizializziamo con il comando `git init`, digitato nella finestra e osserviamo che nella directory è stata creata la "cartella nascosta" `.git`:



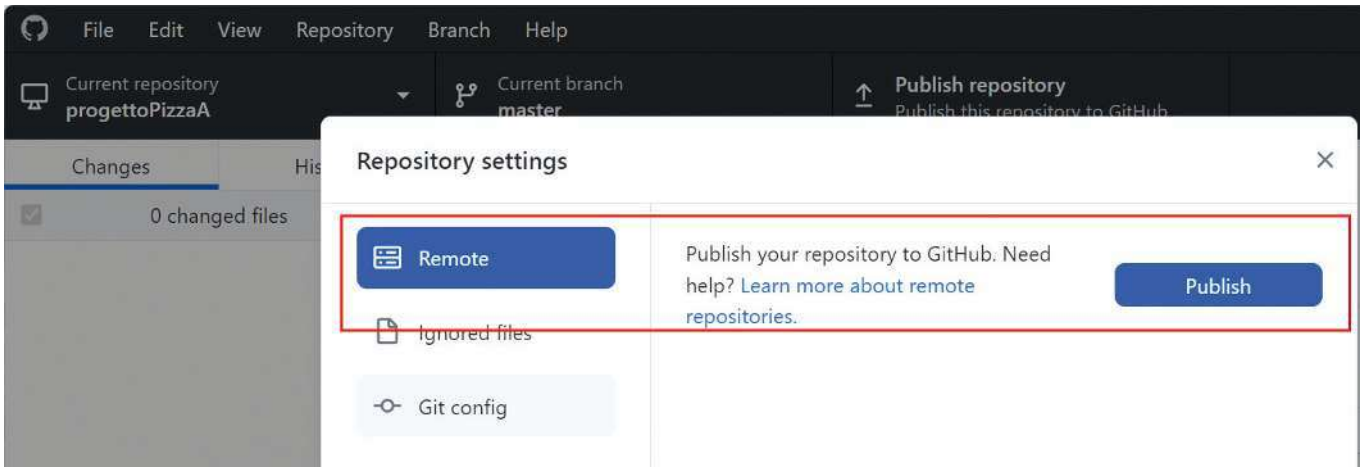
ESERCIZI IN LABORATORIO 1

Se è necessario, inizializziamo l'ambiente con il nome del programmatore:

Colleghiamolo a **GitHub Desktop**:



Naturalmente a questo punto non siamo ancora collegati al repository web, come possiamo osservare:



Non dobbiamo pubblicare il nostro repository locale ma collegarlo a quello remoto creato dal capo progetto: quindi non procediamo per questa via ma da Git locale.

1 ESERCIZI IN LABORATORIO

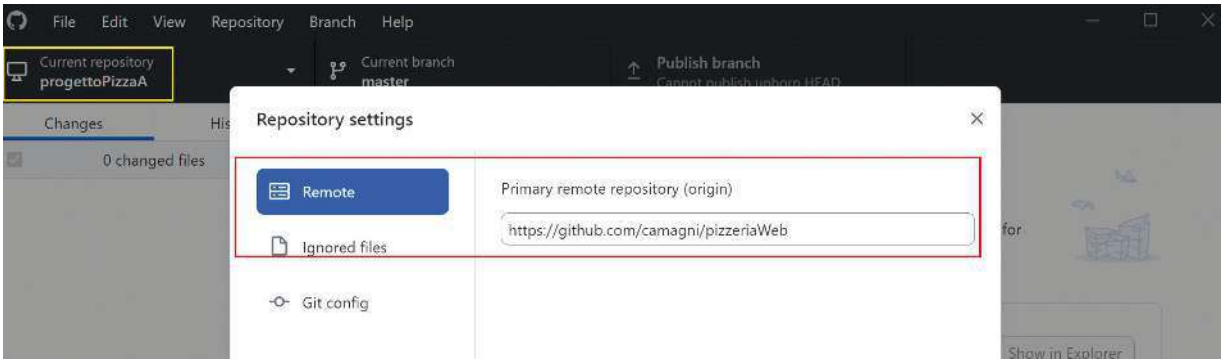
Torniamo ora alla **finestra Bash** ed eseguiamo le seguenti istruzioni da **linea di comando** per connetterlo:

```
git remote –v
```

per visualizzare l'elenco dei repository remoti configurati e i relativi URL: non viene visualizzato nulla, quindi eseguiamo il comando per **associare un repository remoto a un repository locale**:

```
git remote add origin https://github.com/camagni/pizzeriaWeb
```

Se rientriamo nella configurazione di GitHub Desktop ora troviamo correttamente inserito il percorso web:



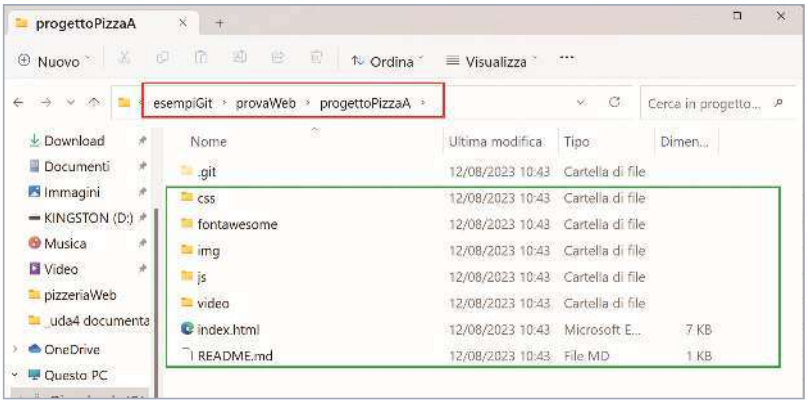
Per **allineare il repository locale con quello remoto** scaricandone tutto il suo contenuto scriviamo il comando:

```
git pull origin main
```

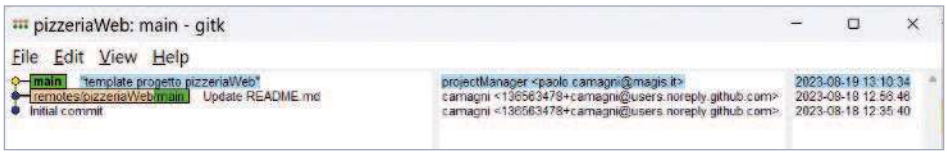
che trasferisce tutto il contenuto del repository remoto su GitHub sul PC locale.

ESERCIZI IN LABORATORIO 1

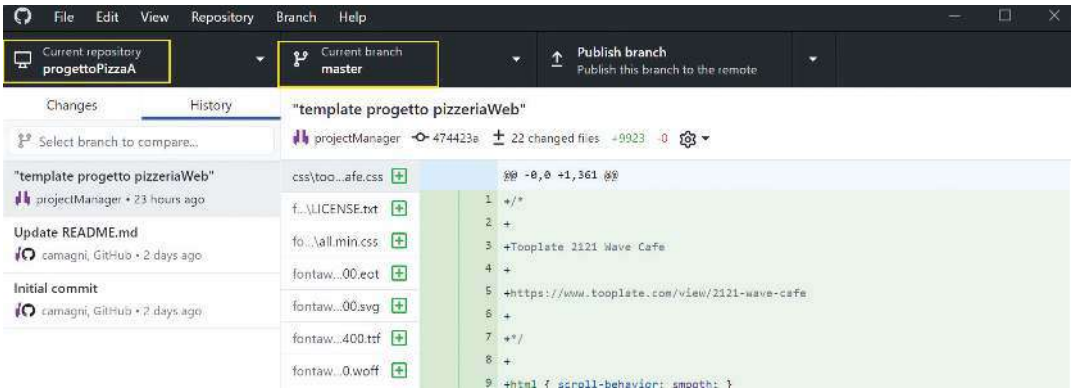
Eseguito il comando, nella nostra directory `progettoPizzaA` troviamo "scaricati" tutti i file presenti nel repository remoto:



Se andiamo a vedere la "storia", scopriamo che abbiamo la stessa situazione del **repository locale del PM**: in poche parole abbiamo effettuato una copia del suo repository locale passando attraverso il repository remoto!



Ora anche nella configurazione di **GitHub Desktop** troviamo correttamente inserita tutta la storia che abbiamo nel repository locale:



PER SAPERNE DI PIÙ

Se è la prima volta che il pc dello sviluppatore si collega a GitHub potrebbero esserci due situazioni da risolvere:

1. potrebbe venire richiesta l'autenticazione su GitHub, per esempio fornendo un OTP da inserire al link indicato nell'immagine a destra;
2. potrebbe verificarsi un problema relativo ai certificati SSL: ne disabilitiamo il controllo e procediamo con il comando mostrato nella figura seguente.

1 ESERCIZI IN LABORATORIO

Flusso di lavoro locale e remoto dello sviluppatore

Prima di procedere, è doveroso fare un'osservazione e riportare la seconda regola fondamentale dello sviluppo condiviso.

> Il **secondo principio dello sviluppo condiviso** (o seconda regola del GitHub Flow) stabilisce che:
> #2 create descriptive branches of master
> cioè ogni programmatore deve lavorare su un branch "descrittivo", e quando ha pronto un segmento/parte di lavoro lo deve pubblicare su un suo personale branch: sarà compito del PM verificare periodicamente la correttezza/qualità del lavoro e, in caso positivo, effettuare il merge con la versione pubblicata (*main*).

Ricordiamo che un **branch** è una copia separata del **repository** che consente agli sviluppatori di lavorare su modifiche specifiche senza influire sul codice principale: nel branch personale questi sviluppano le nuove funzionalità o correggono i bug in isolamento per unirle, quando pronte, al branch principale tramite una richiesta di pull (**pull request**).

Flusso di lavoro locale
(che effettua modifiche al repository locale)

- M controllare lo stato del repository locale
- M registrare versione in locale [`status`][`add`][`commit`]
- M tornare indietro nel tempo

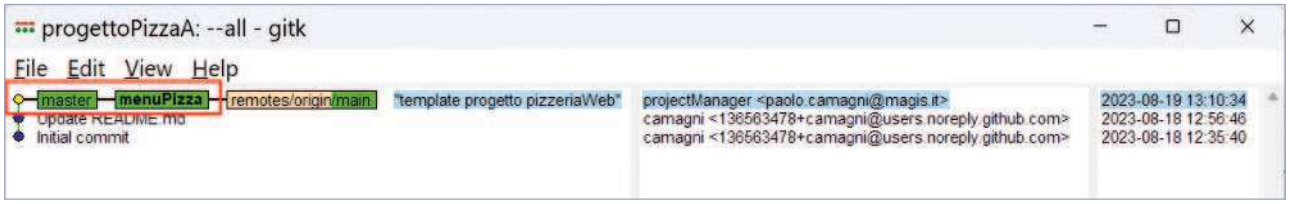
Flusso di lavoro remoto
(che effettua modifiche al repository remoto)

- M controllare lo stato del repository remoto
- M aggiorna repository remoto da locale [`push`]
- M aggiorna repository locale da remoto [`pull`]
- M risolvi conflitti [`merge`]

Come prima operazione effettuiamo quindi un branch con un nome significativo:

```
$ git checkout -b menuPizza  // sviluppatoreA
$ git checkout -b menuDolci  // sviluppatoreB
$ git checkout -b menuBibite // sviluppatoreC
```

Osserviamo come questa operazione ci viene visualizzata con **Git GUI**:



ESERCIZI IN LABORATORIO 1

Sviluppo locale

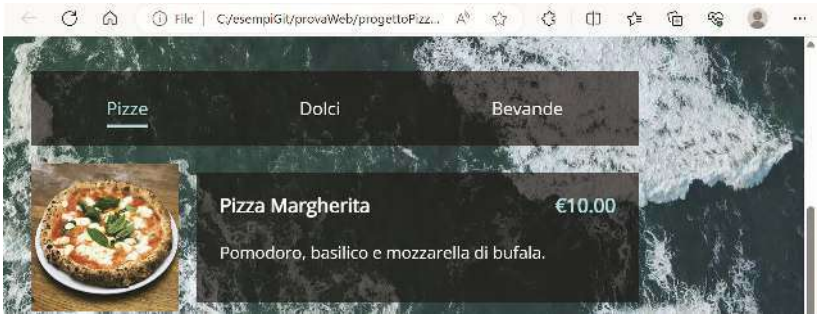
Il nostro compito come **sviluppatoreA** è quello di inserire item nel menù *pizza*; lo facciamo con due operazioni:

1. dopo aver selezionato un'immagine, per esempio quella della *pizza margherita*, la copiamo nell'apposita cartella `img` come mostrato nella seguente schermata:

2. effettuiamo le modifiche al codice (file `index.html`) per aggiungere la descrizione:

```
<!--  elenco pizze -->
<div id="Pizze" class="tm-tab-content">
  <div class="tm-list">
    <div class="tm-list-item">
      <img src="img/pizza1.jpg" alt="Image" class="tm-list-item-img">
      <div class="tm-black-bg tm-list-item-text">
        <h3 class="tm-list-item-name">Pizza Margherita <span class="tm-list-item-price">€10.00</span></h3>
        <p class="tm-list-item-description">Pomodoro, basilico e mozzarella di bufala.</p>
      </div>
    </div>
  </div>
</div>
<!--  elenco dolci -->
```

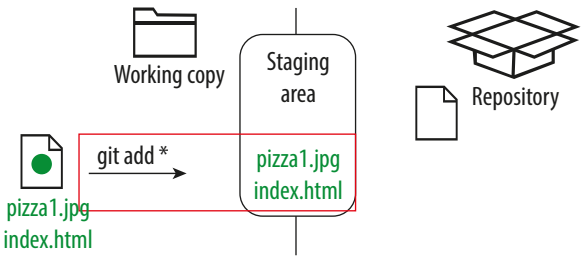
Dopo aver collaudato in locale il nostro lavoro:



siamo pronti per condividerlo con il gruppo: abbiamo aggiunto un file (`pizza1.png`) e ne abbiamo modificato un secondo (`index.html`) nella nostra **working copy**:

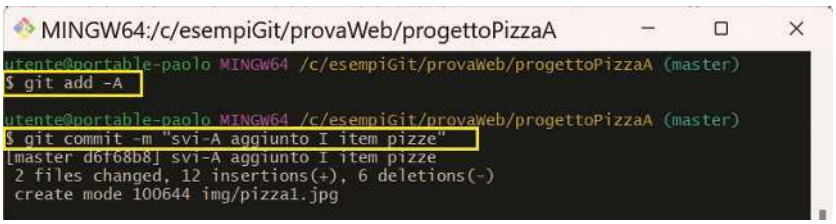
Ora effettuiamo le operazioni per condividere il nostro lavoro.

1. Con il comando `git add -A` si aggiungono le novità presenti nella cartella alla **staging area**:

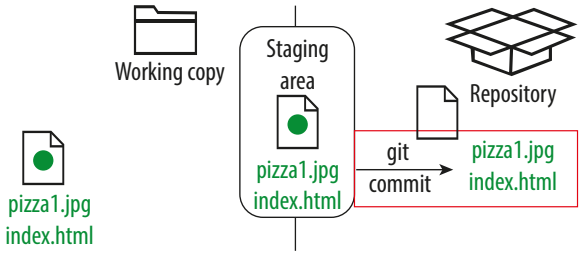


1 ESERCIZI IN LABORATORIO

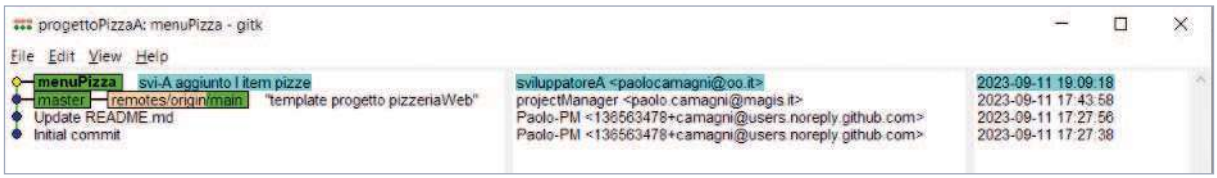
2. Con il comando `git commit -m "svi-A aggiunto I item menu pizze"` le modifiche sono aggiunte al repository locale:

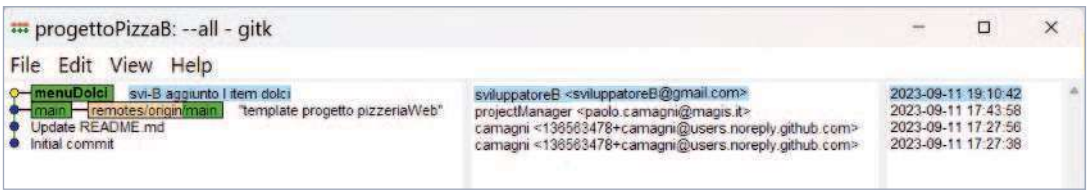


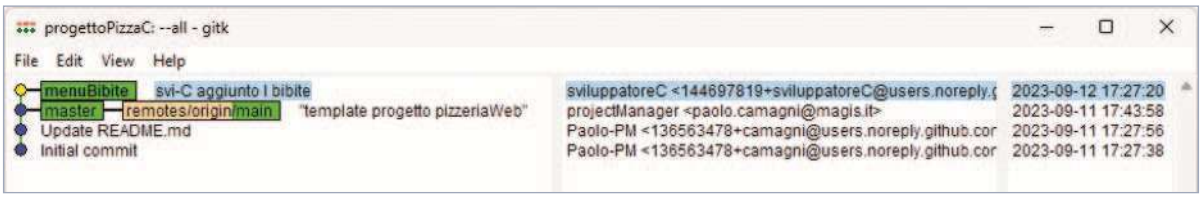
Ci troviamo in questa situazione, cioè con tre copie dei file:



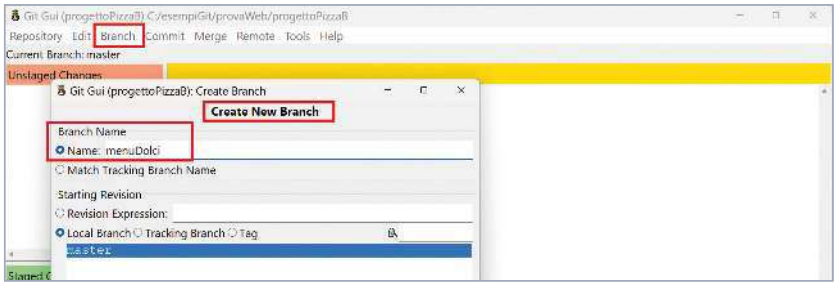
Analogo lavoro viene fatto dagli altri sviluppatori, che aggiungono un item nell'elenco dei dolci e delle bevande: riportiamo di seguito le tre viste grafiche di Git GUI.







Possiamo anche effettuare le operazioni direttamente dall'interfaccia GUI: utilizziamo questa modalità per il secondo sviluppatore, cioè nel **menuDolci**:



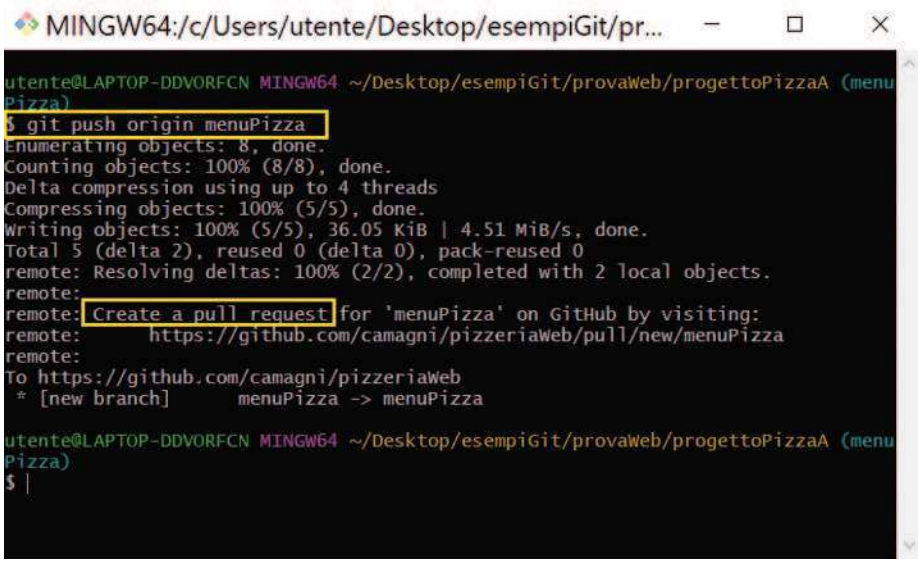
ESERCIZI IN LABORATORIO 1

Caricamento in GitHub

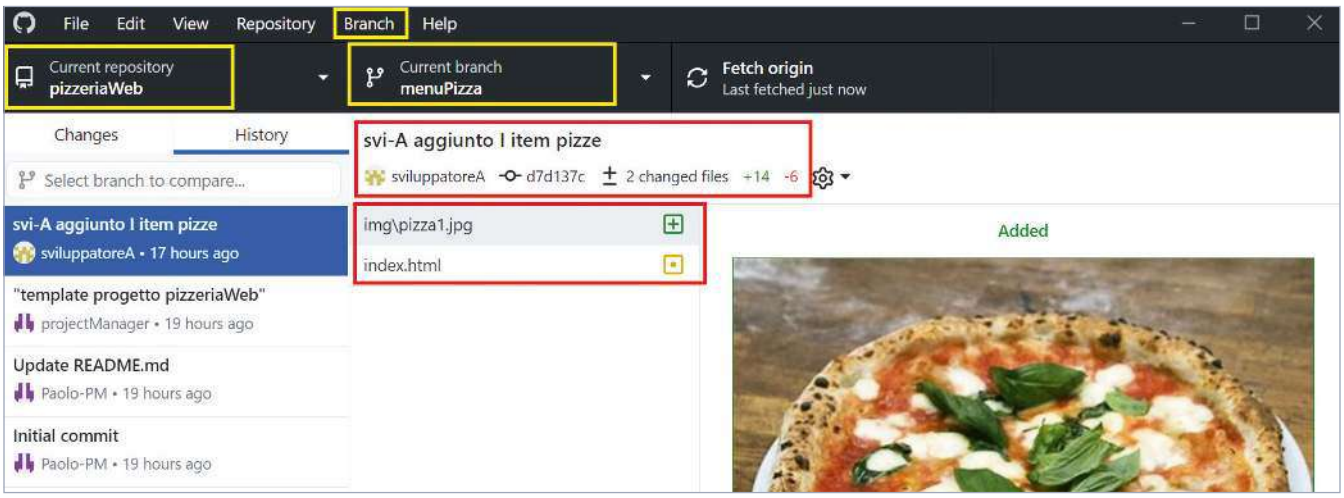
La terza regola dello sviluppo condiviso è la seguente:
#3 - push to named branches constantly

cioè "non appena hai qualcosa pronto" lo devi "pushare" in remoto per farlo approvare dal PM e pubblicarlo nella "linea principale"
Procediamo quindi pushando il nostro lavoro in remoto con:

```
$ git push origin menuPizza   // sviluppatore A
$ git push origin menuDolci   // sviluppatore B
$ git push origin menuBibite  // sviluppatore C
```

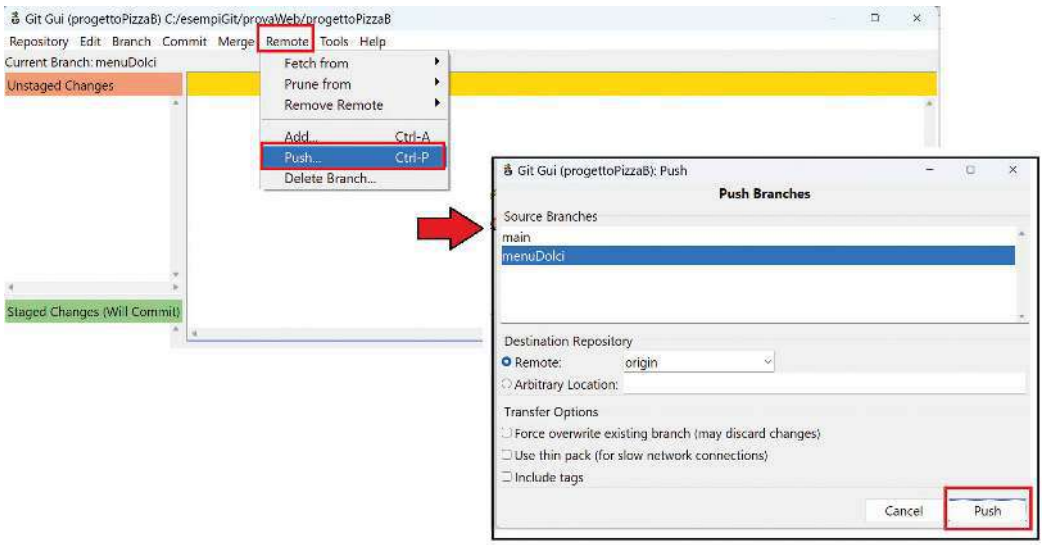


Osserviamo come viene indicato di effettuare una **pull request:** posizionandoci su GitHub Desktop osserviamo che nell'elenco delle attività troviamo quanto aggiunto dallo **sviluppatore A** (**svi-A**) con l'indicazione del file modificato e di quello aggiunto:



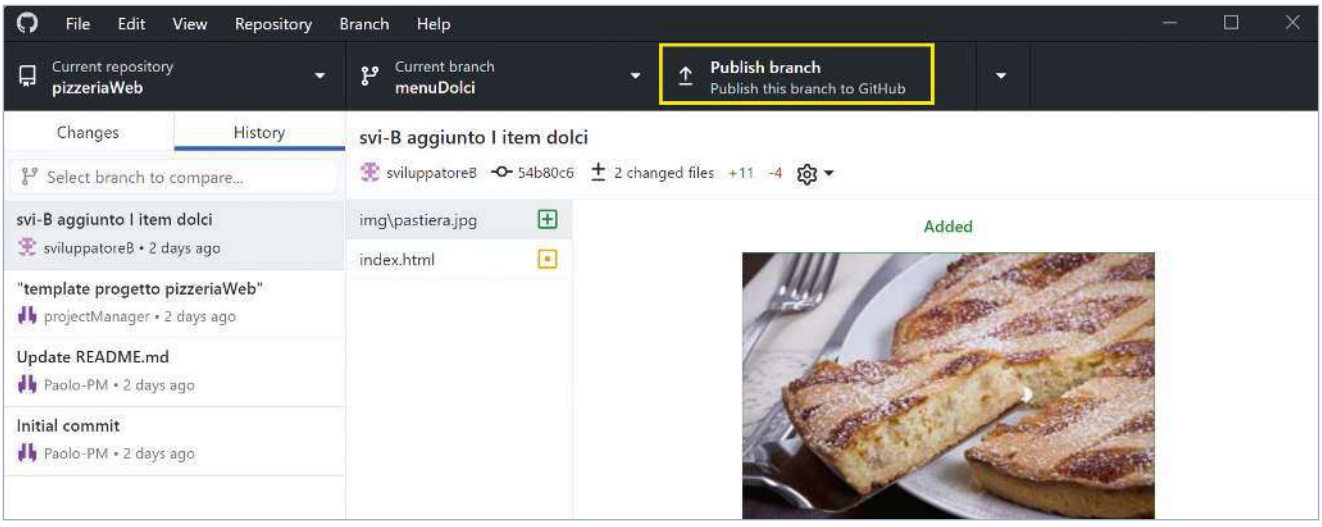
1 ESERCIZI IN LABORATORIO

Vedremo nel prossimo paragrafo come proseguire, intanto facciamo le medesime operazioni per lo **sviluppatoreB** utilizzando l'interfaccia GUI:



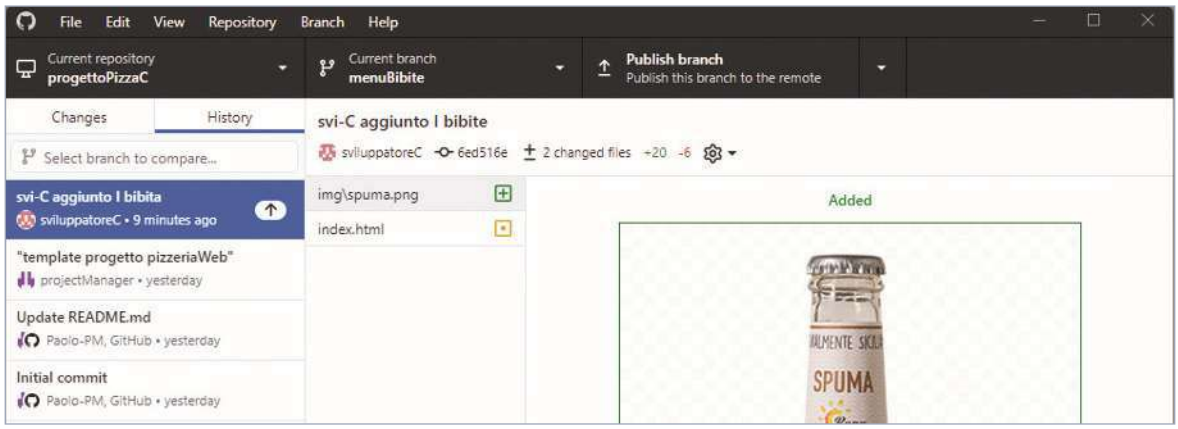
Se vanno a buon fine, si produce un output simile al seguente:

e si ottiene, naturalmente, la medesima situazione finale in GitHub Desktop:



ESERCIZI IN LABORATORIO 1

Analogamente, lo **sviluppatoreC** esegue il push del suo branch:



Aggiornamento in remoto: pull request

Abbiamo detto che gli sviluppatori **non possono posizionare direttamente il loro lavoro**; per tale fine hanno creato un **branch**: per segnalare al PM che hanno trasferito una nuova versione in attesa di revisione eseguono una particolare operazione, chiamata **Pull Request**.

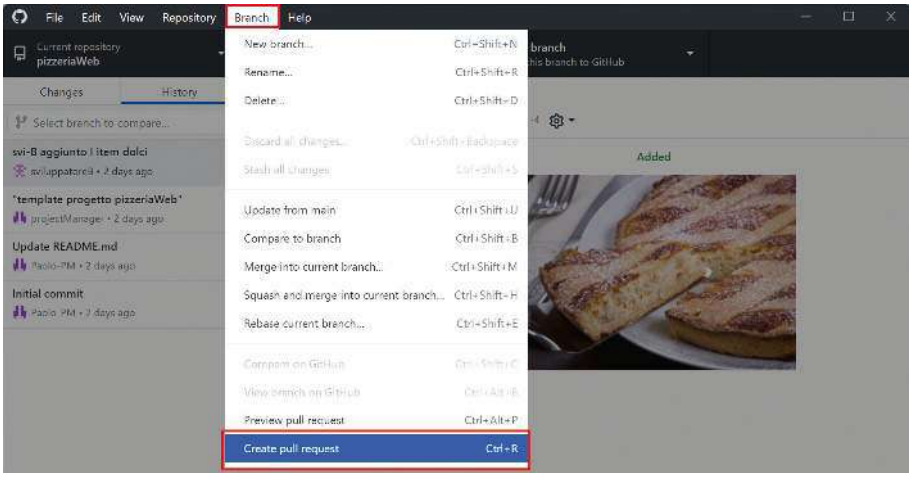
> Quarta regola del GitHub flow:
> #4 - open a pull request at any time

Quando sono completate le modifiche nel branch di uno sviluppatore, questo deve aprire una pull request (**PR**) per richiedere una revisione del codice: la PR è un meccanismo per avviare la revisione, discussione e approvazione delle modifiche prima che vengano integrate nel **branch principale.**

Per effettuare una **pull request** i passi operativi sono i seguenti:

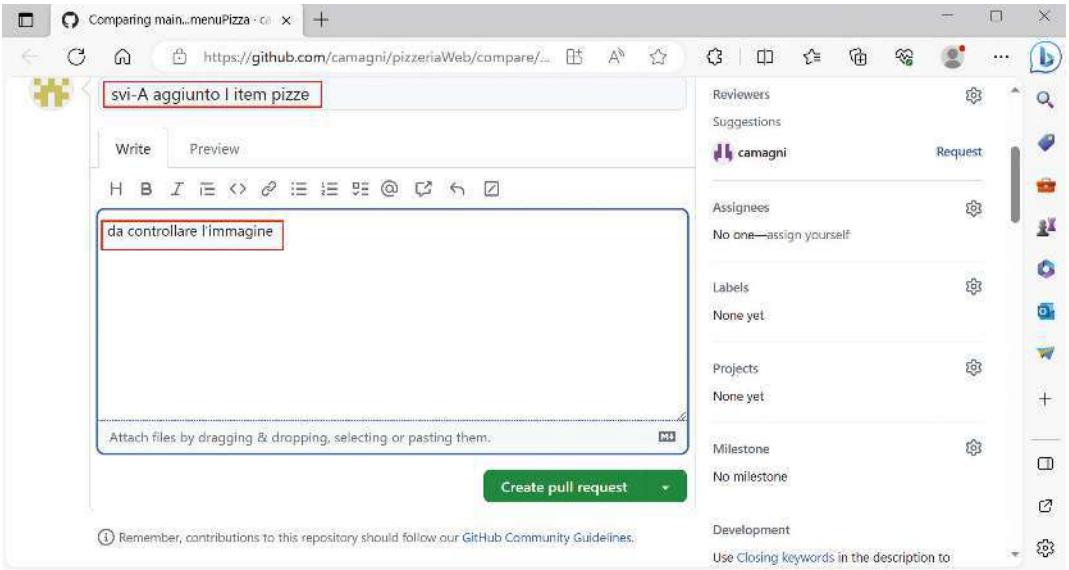
1. posizionarsi sul repository in GitHub;
2. selezionare il nuovo branch come branch corrente;
3. dal menu **Branch** selezionare **Create pull request**;
4. dalla pagina web, dopo aver compilato i campi descrittivi, cliccare sul pulsante verde di creazione.

Procediamo quindi con la creazione di una Pull Request effettuando passo per passo queste operazioni.



1 ESERCIZI IN LABORATORIO

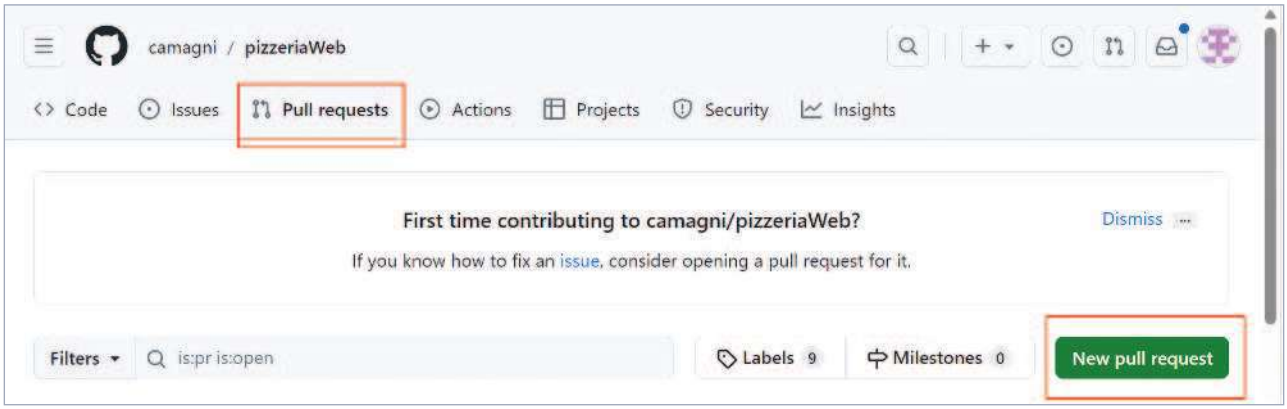
Cliccando su **Branch** / **Create pull request** veniamo indirizzati alla pagina web dove effettuare la **creazione di una pull request**: vi ritroviamo come descrizione il commento scritto al momento del commit:



Nella regione inferiore possiamo aggiungere note/osservazioni per il **PM** che valuterà il nostro lavoro: quindi clicchiamo sul pulsante di creazione e ci viene presentata una videata riepilogativa dopo che GitHub ha effettuato un insieme di controlli sulle nuove integrazioni/rettifiche che si vogliono portare, per verificare la presenza di conflitti e/o errori vari:

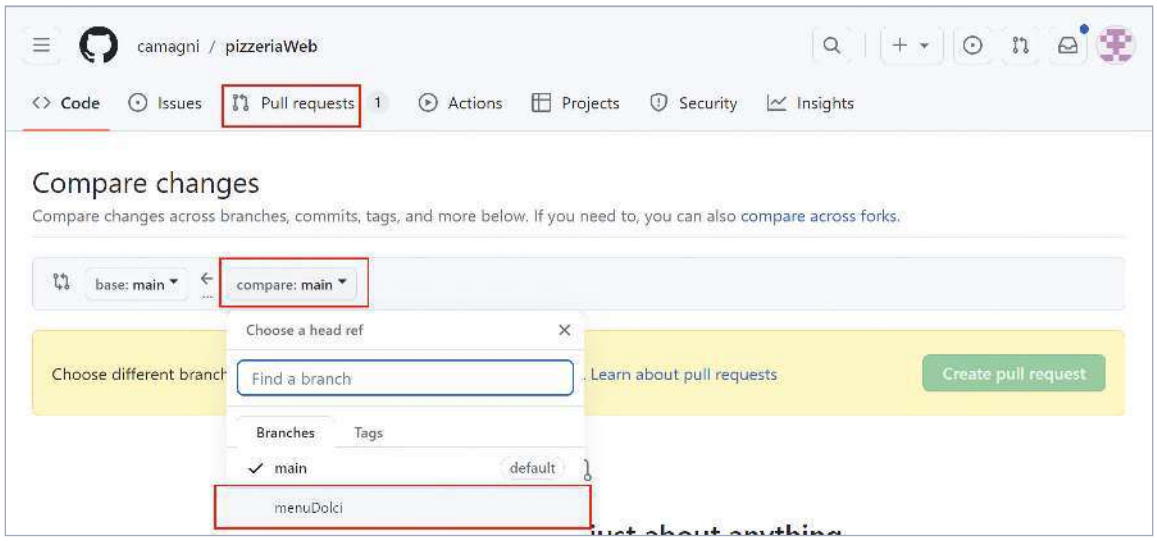
Se viene data una segnalazione come quella sopra significa che le modifiche che abbiamo fatto ai programmi non vanno in conflitto con quello presente nella linea principale.

La stessa situazione si ottiene creando una **pull request direttamente dal browser**, senza passare da GitHub Desktop: si seleziona **Pull requests** e si clicca sull'apposito pulsante verde: come esempio lo utilizziamo per completare l'iter del secondo sviluppatore.

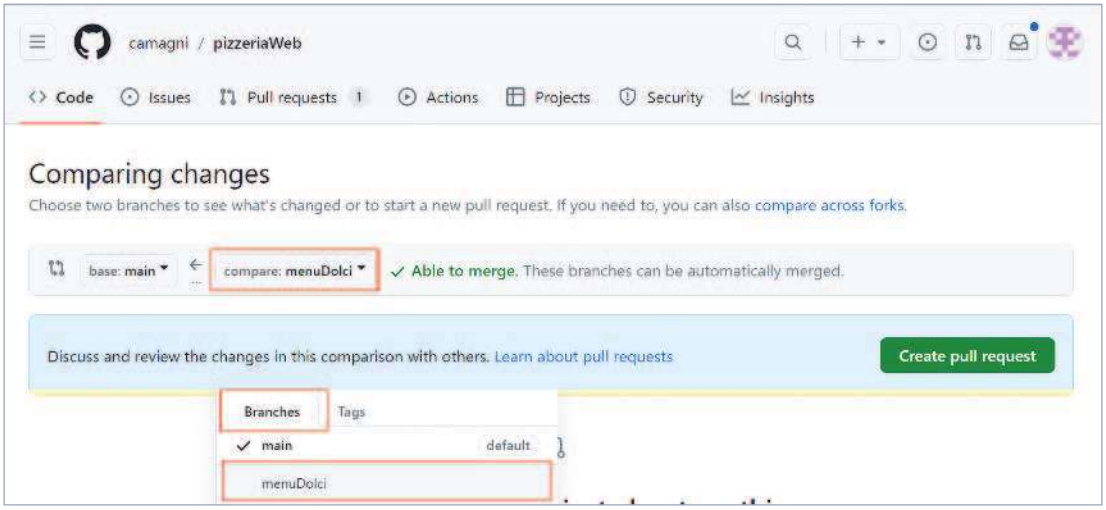


ESERCIZI IN LABORATORIO 1

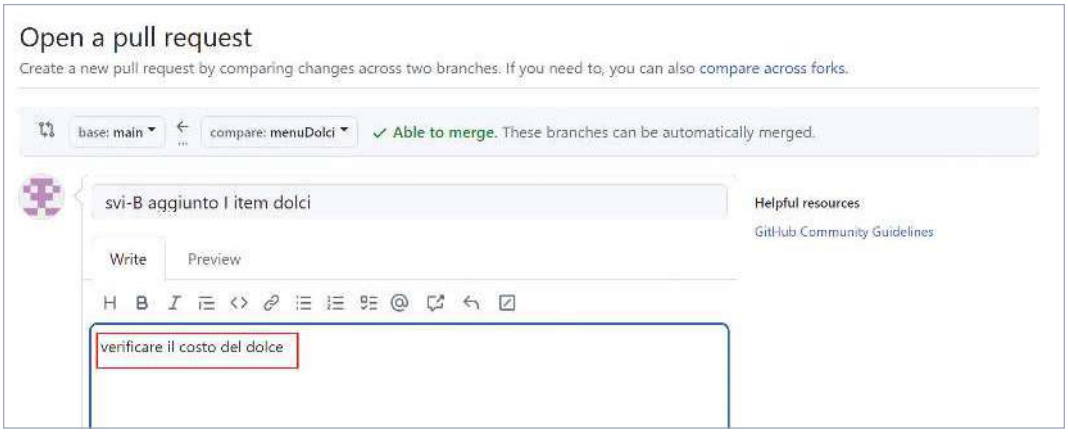
Viene richiesto di indicare il branch desiderato, e noi scegliamo menuDolci:



Quindi confermiamo la richiesta cliccando sul pulsante verde:



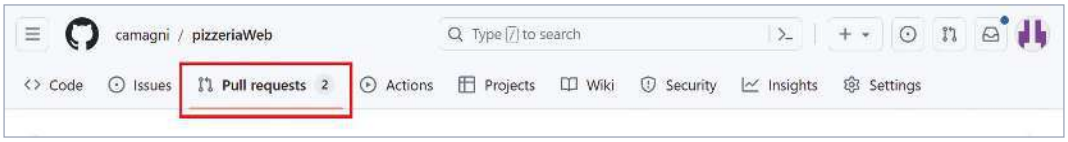
Naturalmente, ci troviamo nella stessa situazione dello sviluppatoreA:



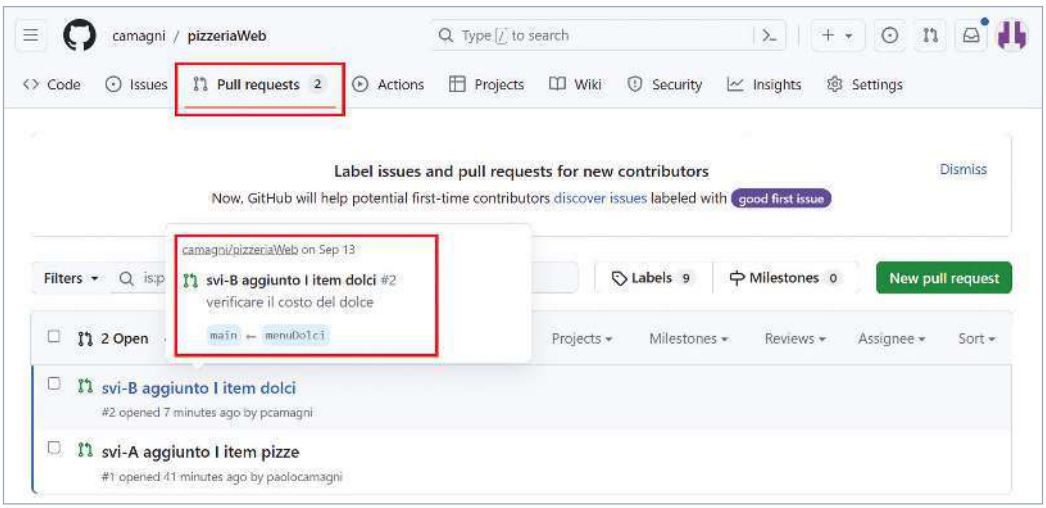
1 ESERCIZI IN LABORATORIO

Anche in questo caso non sono presenti conflitti e, quindi, il nuovo contributo può essere "fuso" con la linea principale, operazione che deve essere eseguita dal PM.

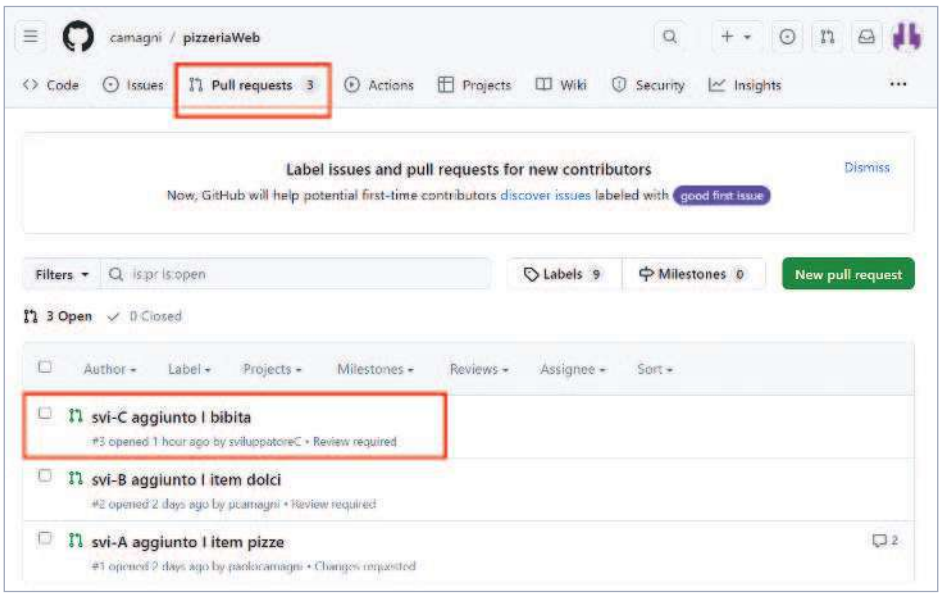
Il PM si accorge della presenza di due **Pull Requests**, dato che esse vengono indicate nella barra superiore:



Al passaggio del mouse viene visualizzato il commento:



Successivamente anche il terzo sviluppatore "consegna" il suo lavoro:



e, quindi, abbiamo tre Pull Request aperte.

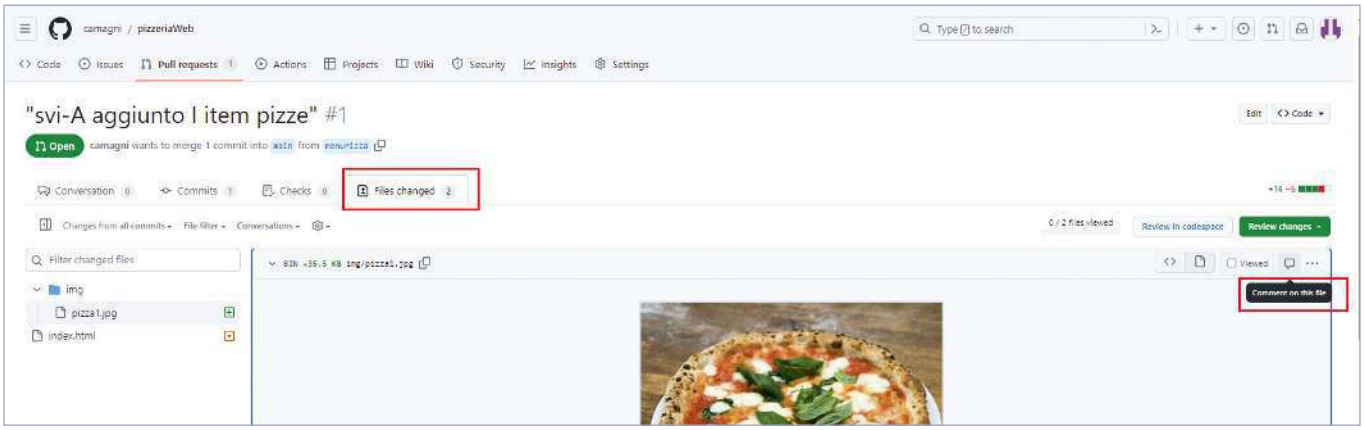
ESERCIZI IN LABORATORIO 1

Aggiornamento e approvazione branch da parte del PM

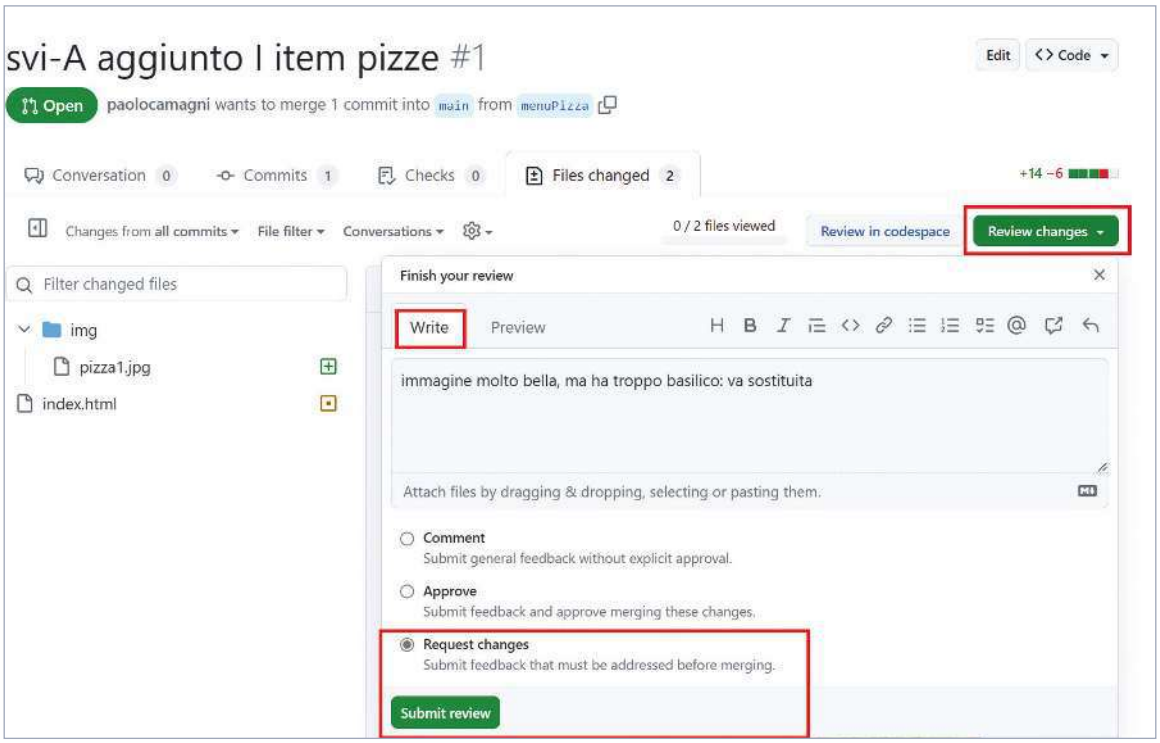
Il PM analizza una **Pull Request**, dove sono presenti tutte le modifiche apportate dallo sviluppatore, in modo da poter revisionare ogni singolo file: nel nostro caso ipotizziamo che l'immagine scelta dallo **sviluppatoreA** per la pizza margherita non sia di gradimento del PM, che decide di richiedere un'azione integrativa con i seguenti passi:

1. seleziona il file da revisionare (l'immagine `pizza1.jpg`);
2. inserisce un commento con le osservazioni per il **DEV** indicando le rettifiche che deve apportare, in questo caso il suggerimento di cambiare l'immagine;
3. seleziona se l'intervento è obbligatorio o solo suggerito e procede con **Submit review**.

Vediamo l'esecuzione dei singoli passi: come PM andiamo a visualizzare l'operato dello sviluppatoreA e, come detto, in particolare l'immagine che ha caricato:

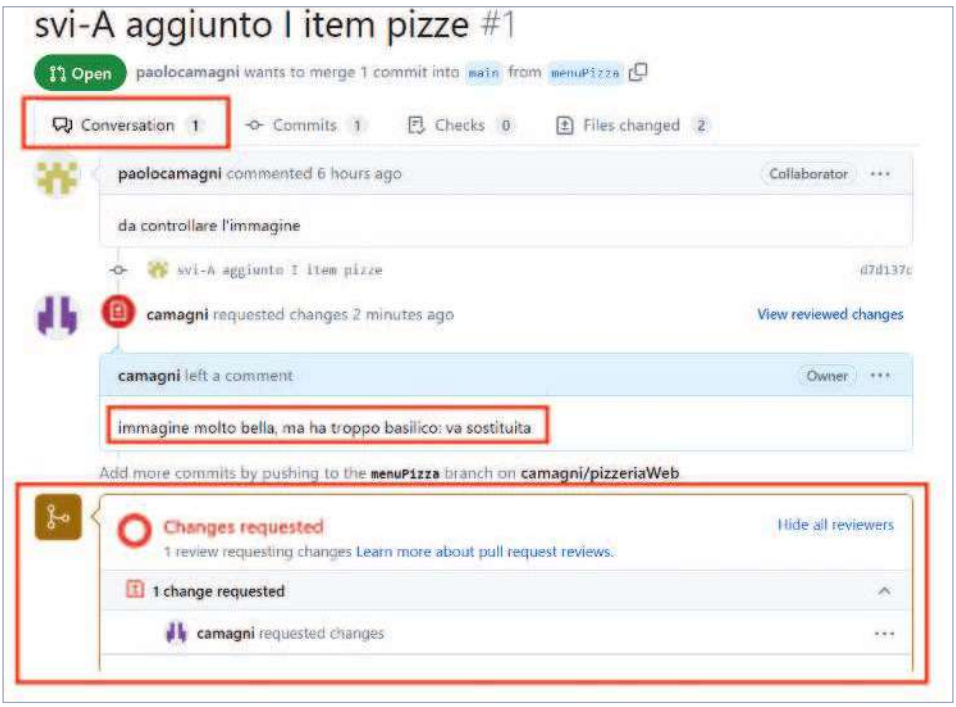


Aggiungiamo un commento, chiedendogli di sostituirla altrimenti non verrà approvato il suo operato:



1 ESERCIZI IN LABORATORIO

Alla conferma veniamo riportati alla pagina principale, dove graficamente viene evidenziato in rosso che il PM ha richiesto un intervento di rettifica che "blocca" l'approvazione del lavoro di svi-A:



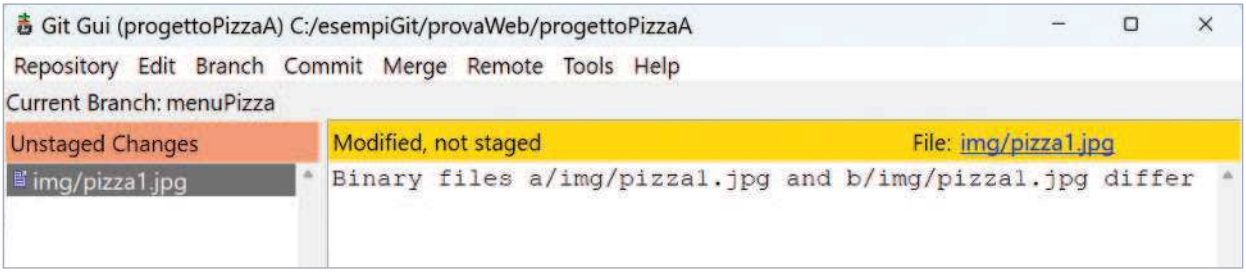
Il sistema provvede a inoltrare una e-mail allo sviluppatore dove viene riportato il testo del commento e il link dove eventualmente replicare:

Azioni correttive dello sviluppatore su richiesta del PM

Lo **sviluppatoreA** comunica di aver ricevuto la segnalazione e che si appresta a fare la sostituzione:

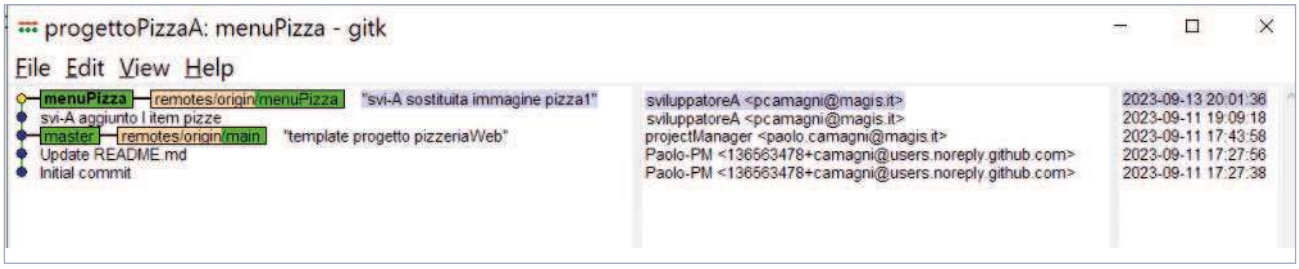
ESERCIZI IN LABORATORIO 1

quindi riprende il progetto, esegue le modifiche del caso sostituendo l'immagine `pizza1.png`:

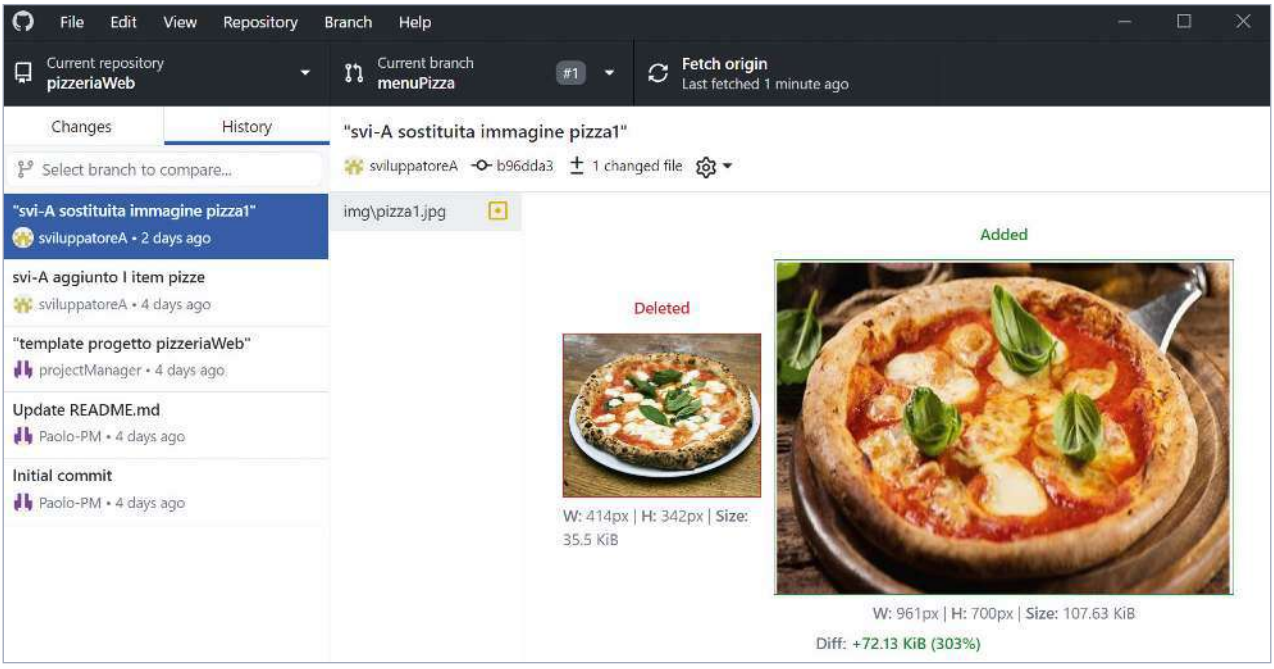


ed effettua il nuovo `add/commit`, eventualmente aggiungendo un commento alla Pull Request, magari menzionando il PM direttamente con il **marker** `@`.

Come ultimo passaggio effettuiamo il caricamento in GitHub semplicemente cliccando su **Push**; ora in GitHub Desktop abbiamo la seguente situazione:



Possiamo anche verificare la nuova situazione presente in GitHub Desktop dove, eventualmente, effettuiamo il fetch per aggiornare la **History** con i nuovi interventi:

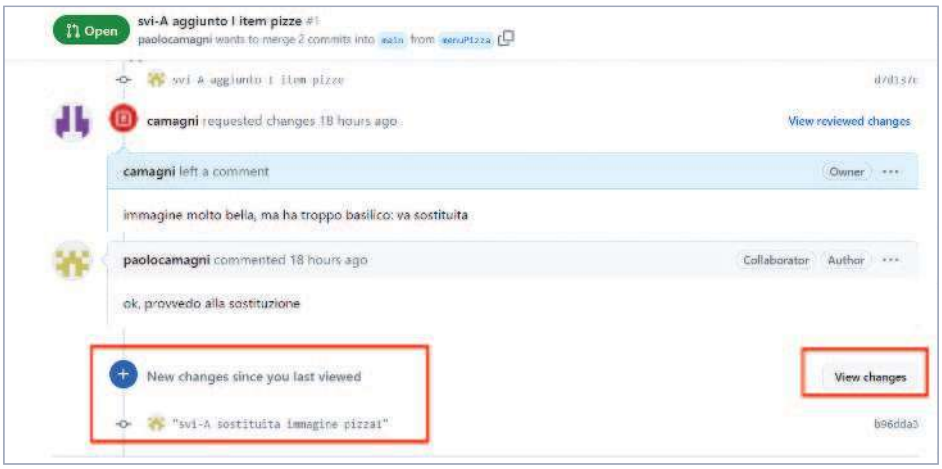


Non è necessario effettuare una nuova pull request dato che stiamo ancora analizzando/risolvendo la prima richiesta.

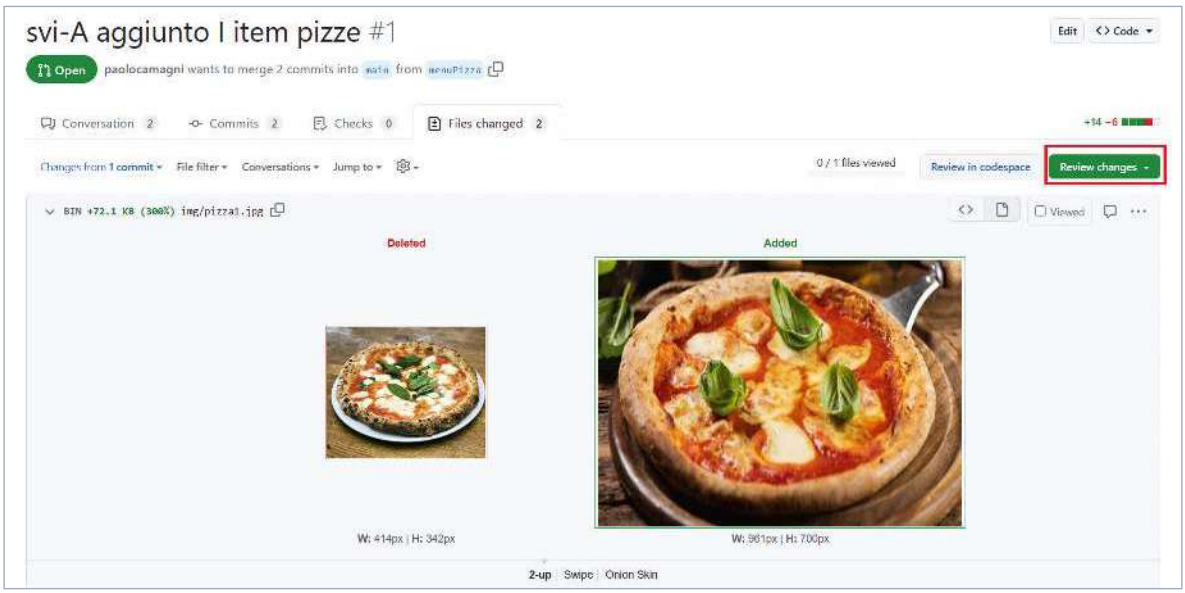
1 ESERCIZI IN LABORATORIO

Replica del PM dopo le correzioni dello sviluppatore

Entriamo in GitHub come PM nel browser e osserviamo la nuova situazione, dove viene indicato al PM che lo sviluppatore svi-A ha sostituito l'immagine della pizza.



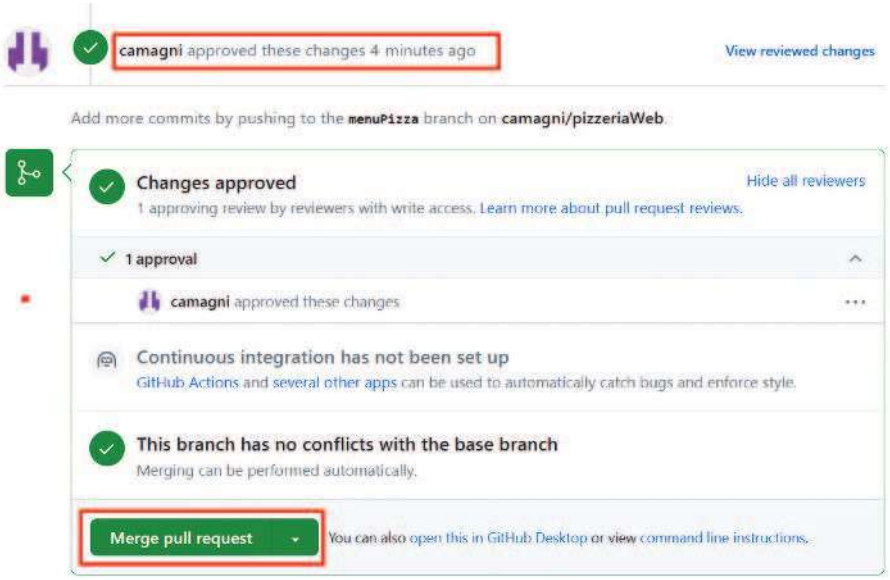
Andiamo a vedere le rettifiche: troviamo entrambe le immagini, così da poterle confrontare:



Se l'intervento è stato di nostro gradimento, approviamo il cambiamento selezionando la prima opzione presente nell'apposito menu:

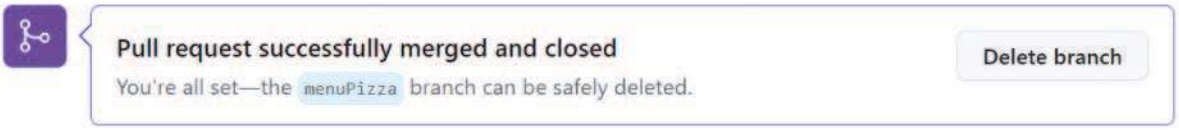
ESERCIZI IN LABORATORIO 1

Approvato il cambio, possiamo procedere con il **merge** in modo da fare confluire l'apporto dello sviluppatore A nel **branch principale**:



Ci viene mostrato cosa stiamo per fondere, cioè l'aggiunta del primo item del menù Pizze:

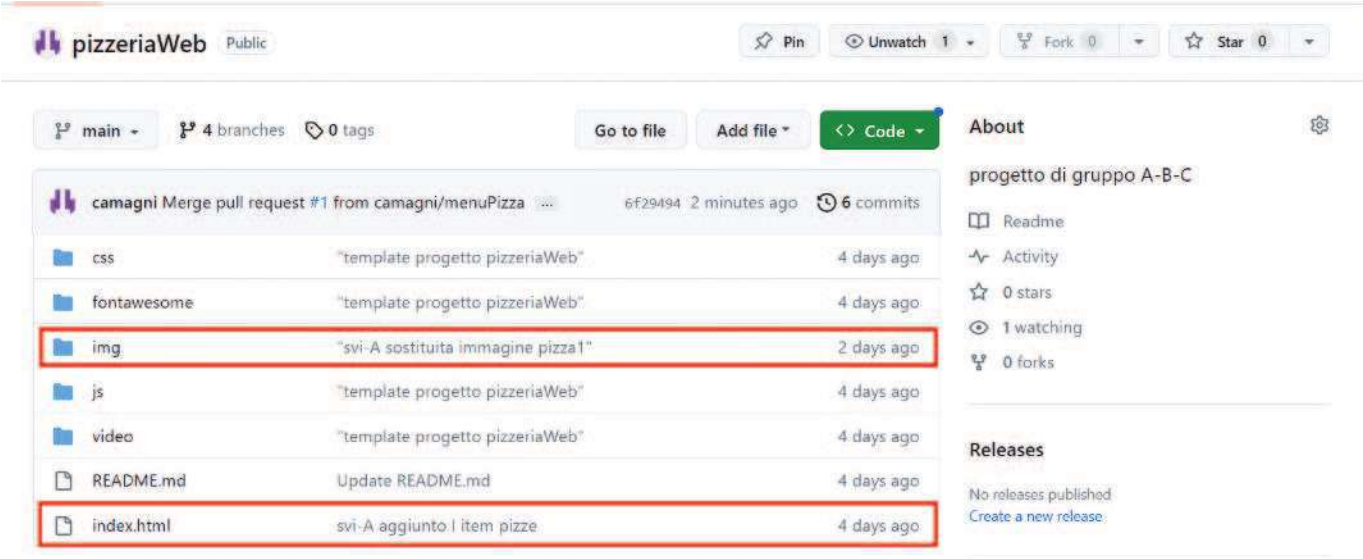
Premiamo l'apposito pulsante e dopo qualche secondo abbiamo il responso:



In questo caso il merge è stato eseguito correttamente, senza che il PM debba compiere interventi ulteriori per conflitti o altro: ora nella linea principale abbiamo la situazione riportata in figura.

1 ESERCIZI IN LABORATORIO

dove possiamo verificare il nuovo elenco dei file presenti nel **branch principale**:



Ora tutti gli sviluppatori possono aggiornare il loro Git locale scaricando questo stato di avanzamento del progetto approvato da PM, che diviene il progetto "pubblicato".

Situazione di conflitto durante il merge

Può però esserci un problema di **conflitto** tra la nuova versione appena modificata da uno sviluppatore e la linea; quindi il PM deve intervenire per due motivi indicati:

- **Review required**, cioè rivedere il lavoro dello sviluppatore;
- **This branch has conflicts that must be resolved**, cioè risolvere il conflitto presente.

Ipotizziamo di effettuare l'approvazione della Pull Request dello SviluppatoreB: come fatto in precedenza, entriamo nella **review** e approviamo le modifiche: quindi clicchiamo su **Resolve conflicts** dove ci viene visualizzato ed evidenziato il segmento di codice da disambiguare.
In questo caso, avendo inserito lo **SviluppatoreB** un `<div>` nelle stesse linee di codice "appena approvate" come modifiche dello **SviluppatoreA,** il sistema segnala che è necessario l'intervento umano per risolvere il problema.

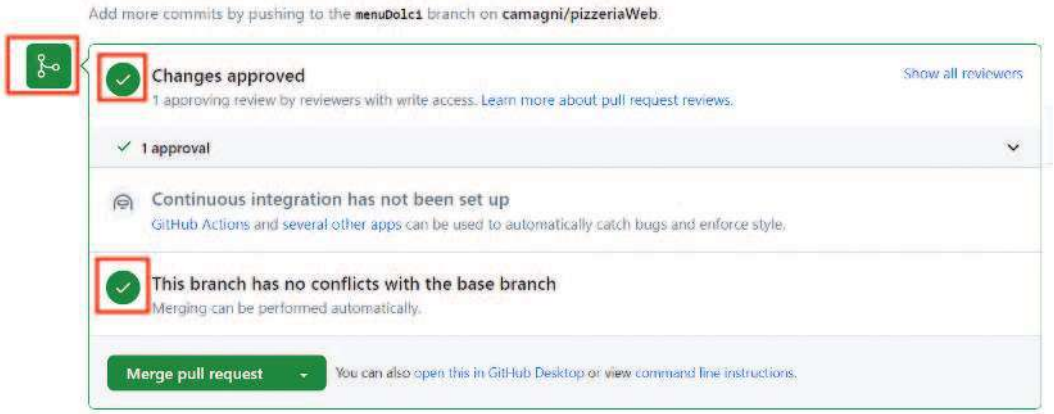
```
56                <div class="tm-list">
57  <<<<<<< menuDolci
58  =======
59                  <div class="tm-list-item">
60                    <img src="img/pizza1.jpg" alt="Image" class="tm-list-item-img">
61                    <div class="tm-black-bg tm-list-item-text">
62                      <h3 class="tm-list-item-name">Pizza Margherita <span class="tm-list-item-price">€10.00</span></h3>
63                      <p class="tm-list-item-description">Pomodoro, basilico e mozzarella di bufala.</p>
64                    </div>
65                  </div>
66
67  >>>>>>> main
68
69                </div>
70              </div>
```

ESERCIZI IN LABORATORIO 1

Dopo aver risolto i tre conflitti portando modifiche al codice del file `index.html`, provvediamo a "marcare" il file come corretto cliccando su **Mark as resolved**:

Ci viene mostrata una spunta verde vicino al file e viene attivato il pulsante per eseguire il merge:

Lo clicchiamo. Ora compare una spunta verde vicino al file ed attivato il pulsante per approvare il merge e veniamo posizionati sulla pagina con i flag verdi, a indicare che abbiamo risolto tutti i problemi e/o situazioni in sospeso:

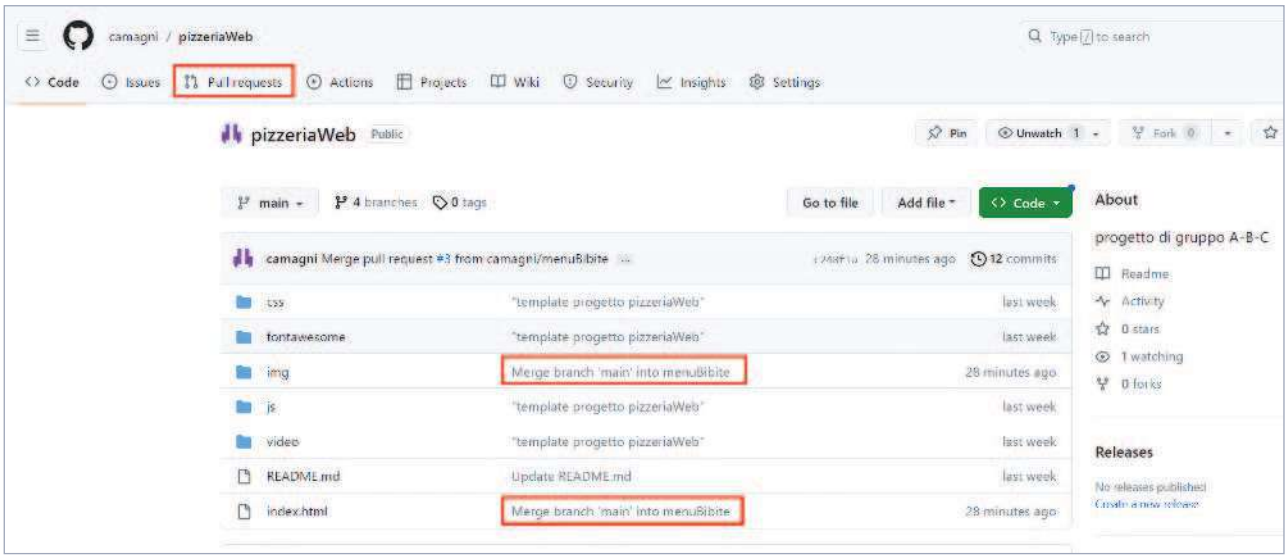


Possiamo procedere confermando l'esecuzione del merge.

L'esito ci viene successivamente confermato da questo messaggio:

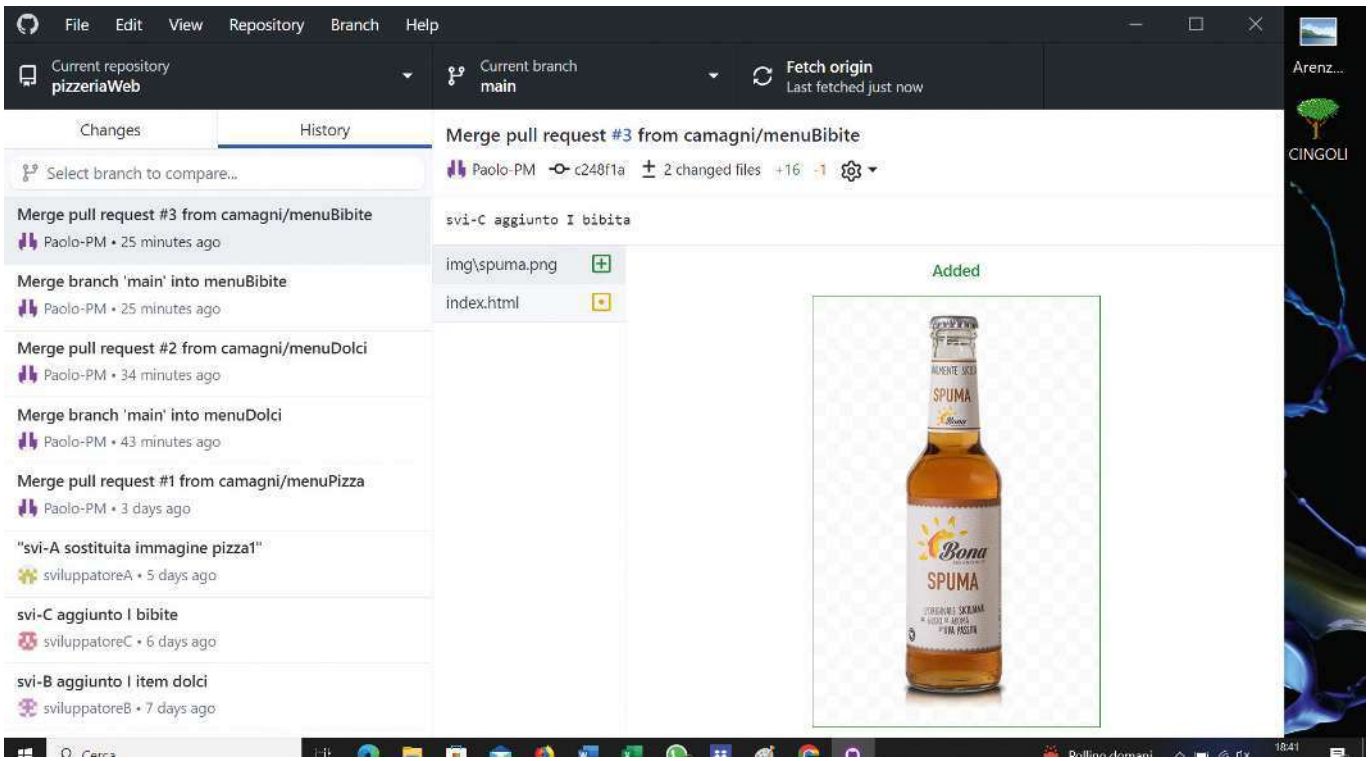
1 ESERCIZI IN LABORATORIO

Lo stesso procedimento viene ripetuto per confermare il merge dello **SviluppatoreC**: alla fine ci troviamo in questa situazione:



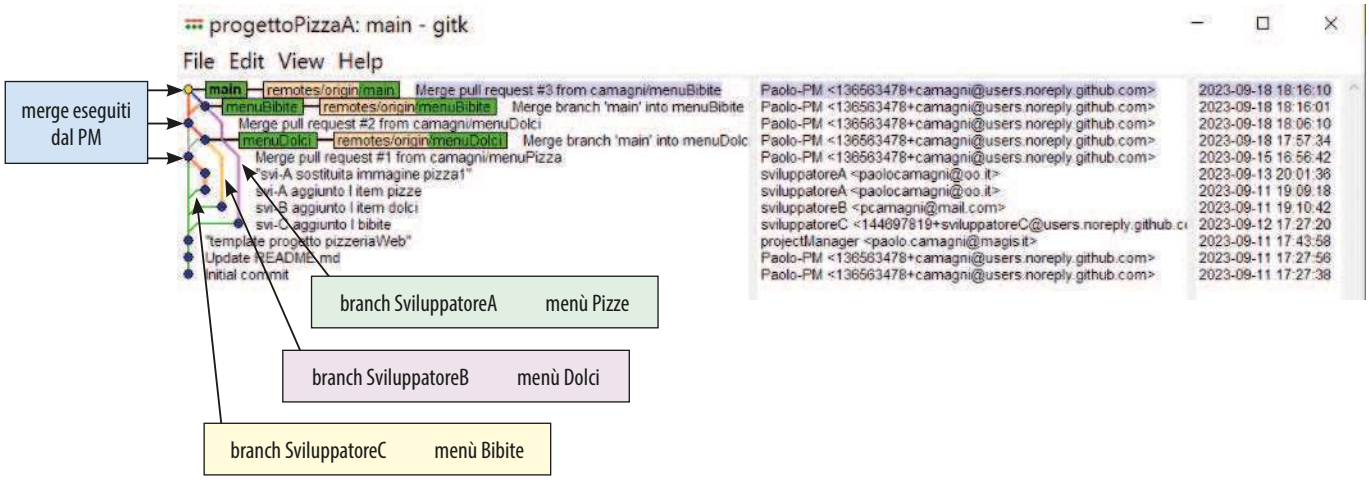
Aggiornamento in locale dopo una approvazione del PM

Ora gli sviluppatori possono aggiornare il loro Git locale scaricando le modifiche apportate e confermate dal PM. Entriamo in **GitHub Desktop** e ci viene segnalato di eseguire una **Fetch origin**: quindi possiamo visualizzare tutte le operazioni effettuate dei membri del progetto, individuabili dalle loro icone collocate a sinistra del corrispondente nome.



ESERCIZI IN LABORATORIO 1

Successivamente entriamo in **Git GUI** dello **sviluppatore A** e osserviamo il grafico dei branch e merge effettuati fino a questo punto del progetto.



Nel caso in cui l'aggiornamento non sia fatto in automatico è sufficiente cliccare su **Fetch from origin**:

Secondo item aggiunto dagli sviluppatori

Il gruppo di sviluppo procede ad aggiungere un nuovo item per ciascun menù, il PM verifica i branch e approva il merge fino a che sono risolti i conflitti: il progetto che ci eravamo proposti di realizzare risulta essere completato con una situazione simile alla seguente.

1 ESERCIZI IN LABORATORIO

METTITI ALLA PROVA

Dopo aver formato un gruppo di sviluppo e assegnato i ruoli, replica quanto descritto fino a questo punto della trattazione.

Quindi completa il progetto facendo aggiungere per ogni menù uno/due item da parte ci ciascuno sviluppatore, sempre procedendo come indicato con pull request da parte degli sviluppatori e controllo/merge da parte del PM.

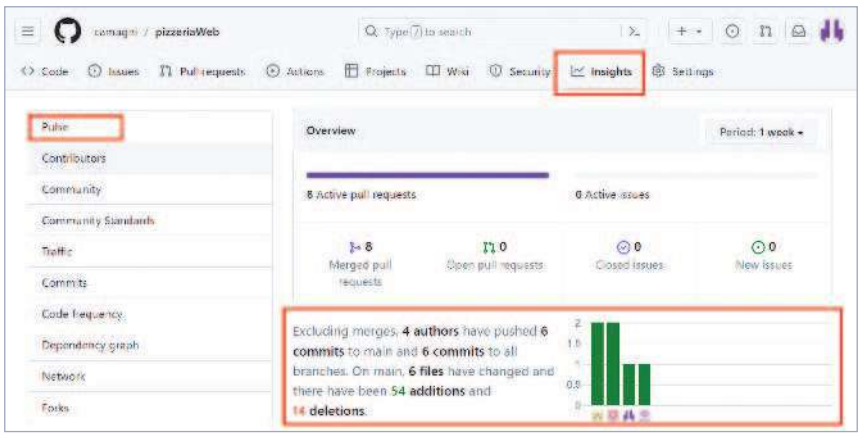
Alcune funzionalità di GitHub

Le possibilità offerte da **GitHub** sono numerose e sarebbe necessario un corso specifico per poter conoscere completamente questo straordinario strumento per lo sviluppo delle applicazioni condivise.
Oltre alle opzioni che abbiamo utilizzato, GitHub offre un insieme di opzioni che consentono agli sviluppatori di collaborare, gestire e condividere il codice sorgente dei loro progetti.
Ne riportiamo sinteticamente le principali, riprendendo il menu generale che è così composto:

1. **Code:** è il punto di accesso principale per il codice sorgente di un repository, offre diverse opzioni per visualizzare, interagire con il codice del progetto, aprirlo in GitHub Desktop oppure in **Visual Basic**, clonarlo oppure scaricarne una copia compressa nel PC locale.
2. **Issue**: sono "la memoria" del progetto, dove si tiene traccia delle questioni/discussioni o problemi sorti durante la progettazione; possono essere aperte dagli sviluppatori stessi o dagli utenti del progetto e possono essere assegnate a specifici collaboratori per la risoluzione.
3. **Pull Request**: è una proposta di modifica aperta da uno sviluppatore per richiedere la revisione e l'approvazione delle modifiche da parte del PM.
4. **Actions**: è possibile creare script personalizzati che consentono di automatizzare il flusso di lavoro di sviluppo del software: questi possono essere eseguiti in risposta a eventi specifici nel repository, per esempio quando viene inviata una pull request o quando viene effettuato un push nel repository, per eseguire test, compilare il codice, distribuire il software ecc.
5. **Projects** è un ambiente visuale e interattivo che consente di organizzare e gestire il lavoro di un repository in forma di progetti creando elenchi di attività, tenendo traccia dei progressi, assegnando compiti agli utenti, offrendo la possibilità di utilizzare diagrammi temporali simili al **Gantt** per le singole attività e per le fasi dei cicli di sviluppo e collaborare in modo efficace.
   È particolarmente utile per progetti con più attività parallele in corso e per promuovere la collaborazione e la trasparenza all'interno del team di sviluppo.
6. **Wiki**: GitHub offre la possibilità di creare una wiki all'interno di un repository per condividere documentazione con gli utenti del progetto.
7. **Security:** mette a disposizione strumenti e funzionalità per migliorare la sicurezza del repository e proteggere il codice sorgente e i dati sensibili; con le sue opzioni il PM può organizzare e risolvere potenziali problemi di sicurezza, mantenere le dipendenze aggiornate e implementare procedure di sicurezza per proteggere il progetto.
8. **Insights** è una funzionalità di **GitHub** che fornisce analisi e informazioni dettagliate sulle prestazioni e l'utilizzo dei repository; vediamo per esempio i dati riepilogativi e le statistiche con il resoconto delle operazioni effettuate nel progetto appena realizzato.
   Nella colonna di sinistra sono elencate le possibili visualizzazioni disponibili: esaminiamo i più significativi.
   Pulse, il primo nell'elenco, visualizza un grafico temporale delle attività nel repository, mostrando il numero

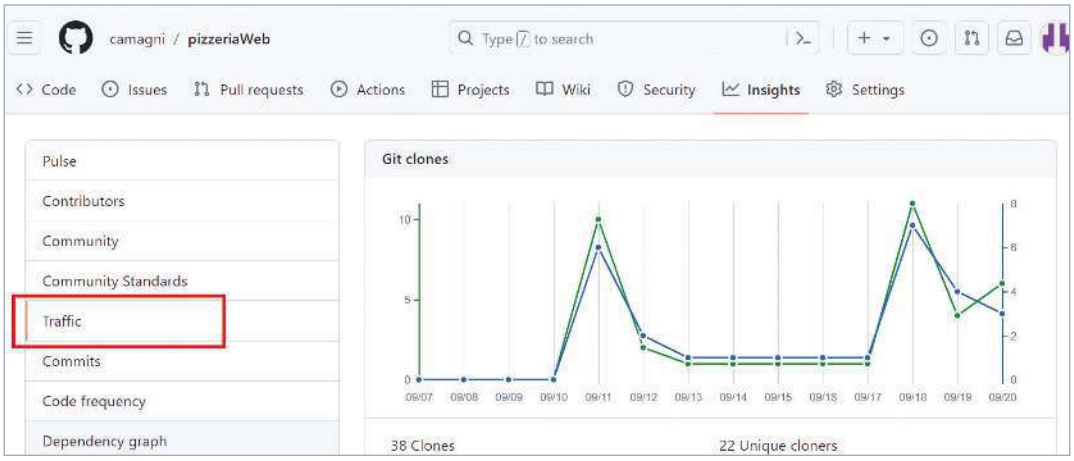
ESERCIZI IN LABORATORIO 1

di problemi aperti/chiusi, richieste di pull aperte/fuse e commit effettuati; può essere utilizzato per tracciare l'andamento del progetto nel tempo e ottenere una visione generale delle attività degli sviluppatori.



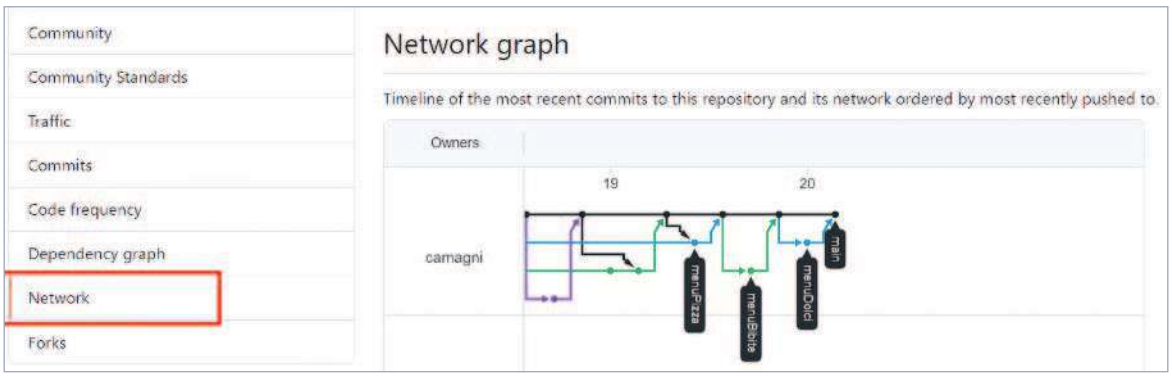
Il secondo, **Contributors**, presenta i grafici di contribuzione con il numero di contributi effettuati da ciascun collaboratore del repository nel corso del tempo.

Traffic presenta l'analisi del flusso di lavoro in modo che il PM possa monitorare le azioni eseguite nel repository: sono evidenziati i i tempi di esecuzione delle azioni, il numero di successi/fallimenti e altre informazioni utili per il monitoraggio e l'ottimizzazione dei flussi di lavoro.

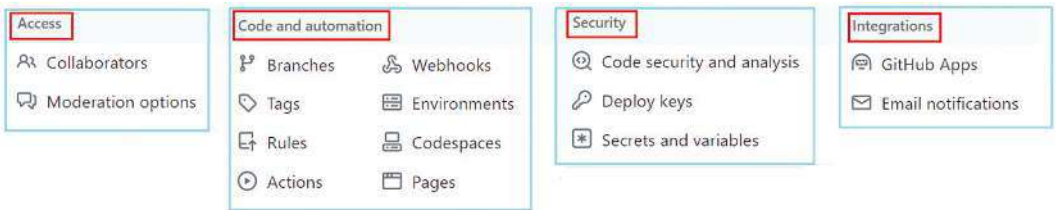


1 ESERCIZI IN LABORATORIO

Network visualizza graficamente la timeline delle operazioni effettuate dai singoli utenti; in particolare sono evidenziate le **Pull Request** e le operazioni di **merge** eseguite dal PM.



Settings, infine, raccoglie tutte le funzioni per configurare le diverse opzioni e personalizzare i ruoli, i permessi di accesso degli utenti al repository. È diviso in quattro sezioni (**Access**, **Code and automation**, **Security** e **Integrations**); in ciascuna sono presenti ulteriori dettagli e sottosezioni per consentire una personalizzazione più approfondita del repository in base alle esigenze specifiche del progetto.



METTITI ALLA PROVA

Dopo aver completato il progetto, analizza tutte le opzioni presenti nella sezione **Insights** e realizza una presentazione da condividere con tutti i membri del gruppo di lavoro.

Quindi analizza la sezione **Settings,** in particolare le opzioni presenti in **Access** e **Security**, in modo da impostare le opzioni per assegnare ruoli e permessi ai futuri collaboratori di un nuovo progetto che richiede una maggiore riservatezza.

Conclusioni

Abbiamo visto come i sistemi di controllo di versione aiutano a soddisfare un'importante esigenza degli sviluppatori: poter identificare senza problemi ogni punto di tutta la storia di un progetto e poter tornare a qualsiasi punto in qualsiasi momento; quindi, per soddisfare questa esigenza, abbiamo visto come **Git** mette a disposizione innumerevoli strumenti che rendono la vita degli sviluppatori più semplice. Non per niente Git dimostra oggi di essere lo strumento di versioning più gettonato nella comunità dei programmatori.
Le possibilità offerte da Git sono molteplici: noi abbiamo semplicemente "introdotto" i concetti essenziali, rimandando alla documentazione ufficiale lo studio completo del prodotto.
Un altro prodotto di versioning popolare quasi quanto Git è **Bazaar**, scaricabile dall'indirizzo

http://bazaar.canonical.com/en/

Essendo Bazaar molto giovane, i suoi progettisti hanno potuto imparare dagli errori commessi nel passato ed evitarli, così come hanno potuto rendere agevole la portabilità e l'interoperabilità con altri sistemi di controllo di versione.